Petri nets for Situation Recognition
Petri nets for Situation Recognition
Abstract

Situation recognition is a process with the goal of identifying a priori defined situations in a flow of data and information. The purpose is to aid decision makers with focusing on relevant information by filtering out situations of interest. This is an increasingly important and non-trivial problem to solve since the amount of information in various decision making situations constantly grow. Situation recognition thus addresses the information gap, i.e. the problem of finding the correct information at the correct time. Interesting situations may also evolve over time and they may consist of multiple participating objects and their actions. This makes the problem even more complex to solve.

This thesis explores situation recognition and provides a conceptualisation and a definition of the problem, which allow for situations of partial temporal definition to be described. The thesis then focuses on investigating how Petri nets can be used for recognising situations. Existing Petri net based approaches for recognition have some limitations when it comes to fulfilling requirements that can be put on solutions to the situation recognition problem. An extended Petri net based technique that addresses these limitations is therefore introduced. It is shown that this technique can be as efficient as a rule based techniques using the Rete algorithm with extensions for explicitly representing temporal constraints. Such techniques are known to be efficient; hence, the Petri net based technique is efficient too. The thesis also looks at the problem of learning Petri net situation templates using genetic algorithms. Results points towards complex dynamic genome representations as being more suited for learning complex concepts, since these allow for promising solutions to be found more quickly compared with classical bit string based representations.

In conclusion, the extended Petri net based technique is argued to offer a viable approach for situation recognition since it: (1) can achieve good recognition performance, (2) is efficient with respect to time, (3) allows for manually constructed situation templates to be improved and (4) can be used with real world data to find real world situations.

Keywords: Situation recognition, Petri nets, situation assessment, information fusion, rule based, Rete algorithm, genetic algorithms.
Sammanfattning

Situationsigenkännning syftar till att hjälpa beslutsfattare att hitta instanser av kända typer av intressanta situationer i stora mängder information. Detta är ett viktigt problem att lösa, givet de stora mängder information som finns tillgänglig i diverse övervakningssystem. Tekniskt stöd för situationsigenkännning angriper således det informationsgap som kan uppstå, d.v.s. att hitta rätt information vid rätt tidpunkt. Intressanta situationer kan utspelas över tid och bestå av många olika objekt samt deras handlingar. Dessa aspekter bidrar till att problemet inte är trivialt att lösa.


Sammanfattningsvis så anses den utökade Petri-nät-baserade tekniken utgöra en hållbar grund för situationsigenkännning eftersom den kan användas för att effektivt känna igen situationer samt att definitioner av intressanta situationer kan anpassas efter den information som inhämtas. Utöver detta så har det även visats att tekniken kan användas för att känna igen existerande situationer som utspelar sig i verklig data.
dedicated to my sister

Jeanette Hilmersson

1969.05.07 – 2005.10.19
Acknowledgements

I am most grateful to my supervisor Lars Niklasson for his never ending support during these five years of thesis joy and frustration. Without your support the story would have been shorter. I could not have hoped for a better supervisor with advise about everything from structural issues and beer decisions to more important decisions at a grander scale. Thank you. It has been good fun.

Many thanks to my co-supervisors: Göran Falkman for many fruitful discussions and valuable feedback, and to Amy Loutfi for much appreciated feedback. I am also grateful to my external reviewer Thorsteinn Rögnvaldsson for very valuable feedback that has led to a better thesis. I would also like to express my gratitude to members from our external research partner Saab AB: Håkan Warston, Martin Smedberg and Thomas Kronhamn, for appreciated feedback, many discussions and for support during the thesis work.

Many thanks to my colleagues in Skövde: Maria Riveiro, Fredrik Johansson, Christoffer Brax, Rikard Laxhammar, Alexander Karlsson, Maria Nilsson, Tove Helldin and Tina Erlandsson. I will certainly miss all the more and less serious discussions we have had during these years. I would also like to thank our Borås partners in fusion: Rikard König, Tuve Löfström and Ulf Johansson. Also, thanks to the Information Fusion Research Program and all its members.

I am also most grateful to my dear Louise who has put up with me during these five years of structured thesis work, but unstructured life. Thank you very much for your support. I would also like to thank my mother Ann, my father Göran, my siblings Joakim and Sandra and my niece Sara. Many thanks to my dear friends: Josef Saltell, Mikael Lantz and Mathias Johander. I am not sure I would have had my sanity intact coming out of this process without you. Ding! GZ. The book is now finished and we can again do more fun stuff. ROFL.


Last but not least, thank you Zoega for Mollbergs blanding.
Publications

The work presented in this thesis has also been presented in a number of papers submitted to conferences and journals. During the thesis work a few publications have also been produced which are of less relevance for the specific problem investigated in the thesis. These are presented separately.

Publications with high relevance


Publications with lower relevance


Contents

1 Introduction ........................................ 1
  1.1 Technical support for recognition ............... 2
  1.2 Towards situation recognition .................. 3
  1.3 Problem formulation ............................ 4
    1.3.1 Research questions ........................ 7
    1.3.2 Research objectives ........................ 7
    1.3.3 Research methodology ....................... 9
  1.4 Thesis overview ................................ 11
    1.4.1 Contributions .............................. 12
    1.4.2 Limitations .................................. 13
  1.5 Thesis outline .................................. 14

I Background ....................................... 15

2 Support for situation awareness ................. 17
  2.1 Situation awareness ............................. 17
    2.1.1 The OODA loop .............................. 19
    2.1.2 Situation analysis .......................... 20
    2.1.3 The information gap ......................... 21
  2.2 Information fusion ................................ 22
    2.2.1 The JDL model .............................. 22
    2.2.2 The λ JDL model ............................ 24
    2.2.3 Situation science ............................ 29
  2.3 Representing situations .......................... 31
    2.3.1 Situation calculus .......................... 31
    2.3.2 Situation theory ............................. 33
    2.3.3 Situation management ....................... 35
  2.4 Recognising situations ......................... 38
    2.4.1 Graph matching ............................. 39
    2.4.2 Probabilistic techniques .................... 40
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.3 Temporal constraint based techniques</td>
<td>42</td>
</tr>
<tr>
<td>2.4.4 Rule based techniques</td>
<td>43</td>
</tr>
<tr>
<td>2.4.5 State transition techniques</td>
<td>44</td>
</tr>
<tr>
<td>2.4.6 Plan recognition</td>
<td>45</td>
</tr>
<tr>
<td>2.5 Chapter summary</td>
<td>46</td>
</tr>
<tr>
<td>3 Techniques for situation recognition</td>
<td>47</td>
</tr>
<tr>
<td>3.1 Rule based recognition</td>
<td>47</td>
</tr>
<tr>
<td>3.1.1 Rule based inference and matching</td>
<td>49</td>
</tr>
<tr>
<td>3.1.2 The Rete algorithm</td>
<td>50</td>
</tr>
<tr>
<td>3.1.3 Temporal extensions to the Rete algorithm</td>
<td>54</td>
</tr>
<tr>
<td>3.2 State transition based recognition</td>
<td>58</td>
</tr>
<tr>
<td>3.2.1 Finite state automata</td>
<td>58</td>
</tr>
<tr>
<td>3.2.2 Petri nets</td>
<td>60</td>
</tr>
<tr>
<td>3.3 Genetic algorithms</td>
<td>66</td>
</tr>
<tr>
<td>3.3.1 Fitness and reproduction</td>
<td>67</td>
</tr>
<tr>
<td>3.3.2 Properties of genetic algorithms</td>
<td>69</td>
</tr>
<tr>
<td>3.4 Chapter summary</td>
<td>69</td>
</tr>
<tr>
<td>II Theoretical results</td>
<td>71</td>
</tr>
<tr>
<td>4 Defining situation recognition</td>
<td>73</td>
</tr>
<tr>
<td>4.1 Towards a conceptualisation of the problem</td>
<td>73</td>
</tr>
<tr>
<td>4.2 An abstract view of the world</td>
<td>74</td>
</tr>
<tr>
<td>4.3 Properties and relations</td>
<td>77</td>
</tr>
<tr>
<td>4.3.1 States and events expressed in first order syntax</td>
<td>78</td>
</tr>
<tr>
<td>4.3.2 The abstract observable universe</td>
<td>80</td>
</tr>
<tr>
<td>4.4 Situations</td>
<td>81</td>
</tr>
<tr>
<td>4.4.1 Situations of temporal nature</td>
<td>82</td>
</tr>
<tr>
<td>4.5 The situation recognition problem</td>
<td>85</td>
</tr>
<tr>
<td>4.5.1 Situation templates</td>
<td>85</td>
</tr>
<tr>
<td>4.5.2 Situation recognition defined</td>
<td>86</td>
</tr>
<tr>
<td>4.5.3 Problem complexity</td>
<td>87</td>
</tr>
<tr>
<td>4.6 Situation recognition in a wider context</td>
<td>89</td>
</tr>
<tr>
<td>4.6.1 System overview</td>
<td>89</td>
</tr>
<tr>
<td>4.6.2 Solution requirements</td>
<td>90</td>
</tr>
<tr>
<td>4.7 Chapter summary</td>
<td>92</td>
</tr>
<tr>
<td>5 Petri net based situation recognition</td>
<td>93</td>
</tr>
<tr>
<td>5.1 Iterative situation recognition</td>
<td>93</td>
</tr>
<tr>
<td>5.2 Analysis and problems</td>
<td>95</td>
</tr>
<tr>
<td>5.3 Representation</td>
<td>97</td>
</tr>
<tr>
<td>5.3.1 Tokens</td>
<td>97</td>
</tr>
<tr>
<td>Chapter</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Transitions</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Places</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Representing facts</td>
</tr>
<tr>
<td>5.4</td>
<td>Algorithm</td>
</tr>
<tr>
<td>5.5</td>
<td>Illustrative example</td>
</tr>
<tr>
<td>5.6</td>
<td>Trading memory for speed</td>
</tr>
<tr>
<td>5.7</td>
<td>Chapter summary</td>
</tr>
<tr>
<td>6</td>
<td>Genetic algorithms for learning Petri nets</td>
</tr>
<tr>
<td>6.1</td>
<td>Learning with genetic algorithms</td>
</tr>
<tr>
<td>6.2</td>
<td>Genetic procedure</td>
</tr>
<tr>
<td>6.2.1</td>
<td>The initial population</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Evolution</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Fitness and recombination</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Fitness calculations</td>
</tr>
<tr>
<td>6.2.5</td>
<td>Initial seeding</td>
</tr>
<tr>
<td>6.2.6</td>
<td>Bootstrapping</td>
</tr>
<tr>
<td>6.3</td>
<td>Bit genome representation</td>
</tr>
<tr>
<td>6.4</td>
<td>Complex genome representation</td>
</tr>
<tr>
<td>6.5</td>
<td>Dynamic complex genome representation</td>
</tr>
<tr>
<td>6.6</td>
<td>Chapter summary</td>
</tr>
<tr>
<td>III</td>
<td>Tools for evaluation</td>
</tr>
<tr>
<td>7</td>
<td>A platform for working with situation recognition</td>
</tr>
<tr>
<td>7.1</td>
<td>Motivation</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Towards answering research question 1</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Towards answering research question 2</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Requirements and design goals</td>
</tr>
<tr>
<td>7.2</td>
<td>Desirer overview</td>
</tr>
<tr>
<td>7.3</td>
<td>Framework library</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Fundamental data types</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Observation processing</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Event processing</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Event usage</td>
</tr>
<tr>
<td>7.3.5</td>
<td>Situation recognition in DESIRER</td>
</tr>
<tr>
<td>7.3.6</td>
<td>Genetic algorithms in DESIRER</td>
</tr>
<tr>
<td>7.4</td>
<td>Host application</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Experimentation</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Graphical user interface</td>
</tr>
<tr>
<td>7.5</td>
<td>Chapter summary</td>
</tr>
</tbody>
</table>
# A simulator for situation recognition research

## The need for data

## Requirements analysis

### Purpose

### Requirements

### Enabling technologies

## Architectural design

### Simulator design

### Editor design

## Implementation

### Scripting

### Nodes, components, resources and allocation

### Basic agent behaviours and collision detection

## Example usage: smuggling scenario

### Scenario description

### Component construction

### Scenario construction

## Chapter summary

## Scenarios for evaluation

### Pick-pocket scenario

#### Scenario description

#### Scenario construction

#### Extraction of events

#### Interesting situations

### Piloting situations in real data

#### Scenario

#### Extraction of events

#### Interesting situations

## Chapter summary

# Empirical results

## Recognition results

### Experimental setup

#### Notation

#### Pick pocket scenario

#### Maritime scenario

## Recognition performance

### Method

### Results

### Conclusions

## Time and memory consumption
## CONTENTS

10.3.1 Method .............................................. 186
10.3.2 Results ............................................. 187
10.3.3 Conclusions ......................................... 192
10.4 Applicability ........................................... 193
10.4.1 Method ............................................. 193
10.4.2 Results ............................................. 194
10.4.3 Conclusions ......................................... 195
10.5 Chapter summary ...................................... 196

11 Learning results ......................................... 197
11.1 Experimental setup ................................... 197
11.1.1 Notation ............................................ 197
11.1.2 Scenario properties ............................... 197
11.1.3 Evolutionary process .............................. 198
11.1.4 Evolutionary parameters ......................... 200
11.2 Promoting precision or recall ....................... 200
11.2.1 Method ............................................. 201
11.2.2 Results ............................................. 202
11.2.3 Conclusions ......................................... 203
11.3 Initial seeding ......................................... 203
11.3.1 Method ............................................. 203
11.3.2 Results ............................................. 204
11.3.3 Conclusions ......................................... 206
11.4 Genome representations ............................... 206
11.4.1 Method ............................................. 207
11.4.2 Results ............................................. 208
11.4.3 Conclusions ......................................... 210
11.5 Bootstrapping .......................................... 211
11.5.1 Method ............................................. 211
11.5.2 Results ............................................. 212
11.5.3 Conclusions ......................................... 216
11.6 Chapter summary ...................................... 216

V Conclusion .................................................. 217

12 Conclusions .................................................. 219
12.1 Thesis problem and research objectives ............... 219
12.2 Answers .................................................. 226
12.3 Discussion ............................................... 227
12.4 Future work ............................................. 230
Chapter 1
Introduction

Situation recognition is a process with the goal of identifying a priori defined situations in a flow of data and information. The purpose is to aid decision makers with focusing on relevant information by filtering out situations of interest. This is an increasingly important, and non trivial, problem to solve since the amount of information in various decision making situations constantly grows. As an example, the total amount of created digital information has been estimated to grow by a factor of ten from 2006 to 2011 – from 200 to 1 800 Exabyte’s (Gantz et al., 2008). Luckily, any single decision maker does not need to sift through all that information, of which only a fraction is likely to be relevant, and still, many complex decision making tasks often require the use of some form of machine based support system in addition to technical systems for collection and display. Too much information to be analysed in too many ways is simply not manageable without some form of support, since operators may have difficulties in analysing all information in a timely manner. As an example, the surveillance system at the subway central in Stockholm contains more than 140 individual cameras, but only two operators.

There are today an abundance of surveillance systems in use, in which numerous sensing platforms, such as radars, video cameras, forward looking infrared sensors and GPS tracking systems, are located throughout the environment. Theses sources are used to provide data and information which often is integrated into support systems for command and control. Such systems can be used as a basis for decision making. Endsley (2000) argues that the main problem when using many of today’s systems is not a lack of information, but rather, it lies in the task of finding the information that is needed when it is needed. Endsley refers to this challenge as the information gap.

Situation recognition aims at aiding decision makers by reducing the information gap. Situation recognition is part of the wider concept of situation assessment, which aims at aiding decision makers in achieving enhanced situation awareness. This is depicted as an important precursor to decision making (Endsley, 2000).
1.1 Technical support for recognition

To exemplify the use of situation recognition, consider the following maritime situation. There are at all times, all over the world, multiple vessels of different sizes moving along the coast lines: tankers, RO/RO\textsuperscript{1} ships, container freighters, fishing vessels and ferries, to name a few. Automatic Identification Systems (AIS), radars and other surveillance equipment can today be used to supply tracks of individual detected objects in specific areas that are being surveyed. This information can be used for detecting situations of interest. However, this often requires manual analysis. Picture some form of smuggling situation, a large vessel moves along the coast. A speed boat is deployed from the vessel and another speed boat sets out from the coast. The two boats rendezvous, something is transferred, after which they return to where they came from. Smuggling situations such as this can be thought of in many different ways however, they can be hard for a human operator to detect, due to the massive amounts of information that is available and which needs to be processed.

Machine based support in the task of recognising patterns that have been defined using expertise could serve as a key capability here. It can however be problematic to define exactly what an interesting situation consists of. For this purpose, data driven techniques can be used. Still, it is desirable if existing expertise can be exploited. Furthermore, data driven techniques often require extensive amounts of training examples. This kind of information does however not exist for many interesting situations.

It is thus of interest to develop techniques that allow for temporal constraints and multiple objects to be modelled, knowledge to be used and adaptation of patterns to be carried out with respect to data. Additionally, there are also time constraints on techniques that are used. Available data and information often needs to be processed at least on the average rate at which it is made available. To quote the European Security Research Advisory Board (2006) with respect to detection and identification capabilities, “existing technologies are generally too bulky, too slow, and generate unacceptably high false alarm rates.” Naturally, these aspects need to be adhered in more complex processing technologies as well; they need to be efficient and robust.

Pattern recognition and abnormality recognition are admitted as important capabilities for meeting the challenges that lies ahead in the surveillance domain (European Security Research Advisory Board, 2006). Furthermore, situation recognition has been identified as an important function in these kinds of support systems (Steinberg, 2009; Jakobson et al., 2007). However, although much focus within the information fusion community has been put on accurately tracking, estimating and predicting objects in real time using causality, not very much effort is put into the problem of efficiently recognising complex situations that can be of partial temporal definition.

\textsuperscript{1}Roll-on roll-off.
1.2 Towards situation recognition

Situation recognition can in its essence be seen as a pattern matching problem, where the patterns to recognise represent situation types. Situations are according to Lambert (2003b), essentially collections of spatio-temporal facts, where facts denote relations between objects over space and time. Situation recognition is thus the task of finding instantiations of a priori defined prototypical patterns in sets of facts consisting of relations. This is a very complex problem all in itself. Interesting patterns may however also be of partial temporal order, that is, constraints may be partially temporally ordered with respect to each other. This makes the problem even more complex. At least three important factors need to be considered when addressing this problem.

- It is necessary to have suitable representations of typical situations. These representations need to be understandable by human decision makers, as well as easy to use and modify. It is after all a human decision making process that should be supported, and decision support systems should in general have features for explaining their conclusions (Jensen et al., 1995). Furthermore, Bladon et al. (2002) argue that the reasoning process of situation assessment systems needs to be understandable and verifiable. Moreover, the ability of formulating and modifying definitions of interesting situations based on expertise is highly important since it can be used to define interesting situations in context of decision makers’ goals.

- It is important to have robust, complete and efficient techniques that can perform the complete task within some defined deadline, but which also have sufficient performance. Since the task is to recognise instances of situations, naturally, as many interesting situations as possible should be recognised. Also important however, is that situations that are not considered interesting should not be recognised and classified as interesting, e.g. the false alarm rate should be low. It is however a complex world that is sensed, with continuous streams of data and information being processed. This puts demands on algorithms to be sufficiently efficient in such a way as to be able to process information when it is produced.

- It is vital to know which situation types that are interesting, and how these should be defined. It can be difficult for human experts to precisely define the content of interesting situations. This calls for computer based support in constructing and refining definitions of interesting situations (offline learning). This is also highlighted by Bladon et al. (2002), who argue that it is advantageous for situation assessment systems to be tuneable, e.g. able to improve in accuracy by learning from data. Furthermore, the world is constantly changing, and today’s definitions of what is interesting may have changed tomorrow. This too calls for computer based assistance, but in the task of adapting knowledge with respect to data (online learning).
The problem of recognising complex patterns has in the past been addressed using both deterministic and probabilistic inference methods. Deterministic methods that have been used include for example rule based systems for complex event recognition (Walzer et al., 2007, 2008b), timed automata for diagnosis of discrete event systems (Bouyer et al., 2005; Supavatanakul et al., 2006) and the use of Petri nets for activity recognition (Ghanem et al., 2004; Lavee et al., 2007). While examples of probabilistic approaches include the use of hidden Markov models (HMMs) for dynamic behaviour modelling (Chiao and Xydeas, 2004), Bayesian networks (BNs) for detecting insider threats in information systems and terrorist threats in homeland security applications (Laskey et al., 2004; Laskey and Levitt, 2002) and the use of Markov random fields for doctrinal intent inference (Glinton et al., 2006). For the recognition of simple activities, for which the structure is well known or for which explicit training data exist, HMMs, BNs and similar techniques can be used (Perše et al., 2008). Moreover, probabilistic inference methods seem to be the most commonly used within the information fusion domain, since they allow for uncertainties to be represented and accounted for. In domains with high levels of uncertainty and in which information may be missing, probabilistic approaches have many desirable properties, such as the ability of coping with uncertain, inconsistent and incomplete data. Bladon et al. (2002) argue that these are important factors to acknowledge for building robust systems. Ghanem et al. (2004); Perše et al. (2008) however argue that deterministic approaches seem preferable in the case of recognising patterns, consisting of temporal combinations of subevents, that are only vaguely defined and for which training data does not exist.

1.3 Problem formulation

Similar problems, to the situation recognition problem, have been addressed for nearly forty years in the artificial intelligence community, and more specifically in connection to expert systems. Girratano and Riley (1989) claim that rule based expert systems over the years have been one of the most popular types of expert systems. Rule based systems have also been used as a basis for recognition of complex patterns (Edlund et al., 2006; Schmidt et al., 2008; Walzer, 2009). Girratano and Riley (1989) argue that rules are attractive since they have a modular nature, they provide good explanation facilities, and they are similar to the human cognitive process, thus allowing for humans to more easily understand their content.

A rule based expert system consists of three essential components (Luger, 2002). The first component is a number of rules that encode expert knowledge. These are stored in a knowledge base. Secondly, a number of facts or statements about the world are required. These are inserted into a working memory. Finally, the third component is the inference engine, which in a cyclic fashion matches rules in the knowledge base with facts in the working memory, in order to provide solutions to various problems. The processing in a rule based
system proceeds in cycles consisting of three main steps: recognise, conflict resolution and act. This is referred to as the recognise-act cycle (Luger, 2002; Girratano and Riley, 1989). Naturally, it is the recognise step that is related to the specific problem addressed in this thesis. Recognition may, however, also be carried out in a cyclic fashion, as a result of many simple rules being activated and which in combination can be used to recognise complex patterns. In this case the complete chain of inference is relevant.

Rule based techniques are according to Walzer (2009) often based on the Rete algorithm or variants thereof. It, or extensions to it, has for example been used as the basis in OPS\(^2\), CLIPS\(^3\), Soar\(^4\) and JESS\(^5\). The Rete algorithm was introduced by Forgy (1982) to address efficiency issues coupled to rule based matching. This is done through the use of a matching network which is compiled from rules. The key aspects of the Rete algorithm are that (1) it retains partial matches between consecutive updates due to new facts and (2) it tries to reuse overlapping rules in the matching process. For a thorough description of the Rete algorithm, see Forgy (1982) or Girratano and Riley (1989). Undoubtedly, the Rete algorithm has had a large influence on the development of rule based systems over the years. Moreover, the Rete algorithm is claimed to be an efficient algorithm (Zhou et al., 2008; Sapp, 2009). It has however been argued that rule based systems, and the Rete algorithm, do not provide sufficient capabilities for modelling complex patterns consisting of relative temporal constraints (Walzer, 2009). Several extensions to the Rete algorithm have been proposed for addressing this problem (Maloof and Kochut, 1993; Berstel, 2002; Schmidt et al., 2008; Walzer, 2009), of which the work of Walzer (2009) seems to be the most suitable for the problem addressed here. Although explicitly being able to represent temporal constraints, two problems can be identified when it comes to the task of representing and recognising complex patterns that may be of temporal and concurrent definition: (1) temporal and causal relations are hard to visualise without the aid of other forms of representation\(^6\) and (2) consistency of rules may deteriorate due to their complexity.

Finite state machines and hidden Markov models are representations that are well suited for representing and visualising causality. They do however not lend themselves very well to the problem of recognising patterns that may be of partial causal order, due to their sequential nature. This would require complex state paces to be defined rather precisely. Petri nets, a generalisation of finite state automata, do however have the capability of representing concurrency and temporal synchronisation. In fact, Sowa (2000) argues that these are the main strengths of Petri nets.

---

\(^2\)Official Production System, developed by C. Forgy at Carnegie Mellon University.

\(^3\)C Language Integrated Production System: http://clipsrules.sourceforge.net/

\(^4\)For more information about Soar, visit http://sitemaker.umich.edu/soar/home

\(^5\)Java Expert System Shell, for more information visit http://www.jessrules.com/

\(^6\)Such as for example graphs.
Petri nets have been promoted to constitute a suitable mechanism for representing and recognising complex patterns (Ghanem et al., 2004). Moreover, Ghanem et al. (2004) argue that Petri nets provide a nice graphical representation that is easy to understand, since it has a well defined semantics and only uses few types of elements. As stated, Petri nets also offer a mechanism for explicitly representing and visualising concurrency and temporal synchronisation (Castel et al., 1996; Sowa, 2000). Moreover, they have been used on a wide range of problems, c.f. Jensen (1991). Petri nets are founded on a precise mathematical model that can be used for analysis to detect for example inconsistencies and deadlocks (Ghanem et al., 2004). Additionally, Petri nets are not restricted to deterministic modelling and recognition, but have also been used for probabilistic modelling and inference, see for example (Kudlek, 2005; Laufenbach and Pinl, 2005). The use of Petri nets does therefore not exclude neither of the two approaches to modelling and inference.

It has been reported that the Rete algorithm can be used for Petri nets (Burescu and Brezovan, 2001b,a), and that Petri nets can be used as an underlying mechanism in rule based systems (Hura, 1993; Murata and Yim, 1995; Murata and Zhang, 1988; Li, 1994). It is however not known how efficiently Petri nets on their own can be used for recognising situations consisting of partially temporally synchronised relations of varying arity\(^7\) between objects. The following general problem statement is therefore formulated.

**Problem statement.** Is the Petri net based approach viable for recognising situations of partial temporal definition?

A viable solution to the problem of recognising situations of partial temporal definition can be defined as a solution that: (1) can recognise situations with good performance, (2) is efficient with respect to time, (3) allows for manually constructed situation templates to be adapted and (4) can be used in real world systems. If syntactically correct, deterministic approaches have per definition always perfect performance since they find exactly that which is expressed. Thus, if an interesting situation is syntactically expressed in such a way that it is separable from uninteresting situations, then the first requirement is trivial. However, in a real world setting there is often a discrepancy between what actually is expressed and what is interesting. This generates false alarms. These are however highly dependent on the actual purpose and goals of the system in which situation recognition is used. Hence, recognition performance depends on these aspects too. Requirements one and four are therefore tightly coupled to individual real world system. Naturally, before introducing capabilities in a real world system, it is important to know that they in theory are viable. Thus, before investigating performance and applicability in a real world setting, efficiency and adaptivity can be studied in a simulated setting.

\(^7\)The arity of a relation refers to the number of objects in the relation, e.g. a binary relation operates on pairs of objects.
1.3.1 Research questions

In order to fulfil the second requirement of viability, a Petri net based technique needs to be efficient. It is known that rule based recognition using the Rete algorithm with temporal extensions for temporal relations constitute an efficient approach. This leads to the first research question, which reads as follows.

**Research question 1.** Can Petri nets be used for recognising situations as efficiently as rule based approaches using the Rete algorithm with extensions for explicitly modelling temporal constraints?

Even though Petri nets have the potential to be as efficient as rule based techniques, the situation recognition problem also requires that it is possible to learn templates describing interesting situation types from data, or to adapt definitions of interesting situation types using data. This constitutes the third requirement on a viable solution. The task of learning Petri nets has previously been successfully addressed through the use of genetic algorithms (see for example Mayo and Beretta (2010), Nummela and Julstrom (2005) and Alves De Medeiros and Weijters (2004)). It is, however, unknown to which degree genetic algorithms can be used for learning Petri net situation templates. In order to achieve viable Petri net based situation recognition, it is therefore important to investigate if genetic algorithms can be used to construct and adapt Petri net situation templates. Thus, the second research question reads as follows.

**Research question 2.** Can genetic algorithms be used to successfully learn Petri net based situation templates?

In order to answer the two research questions, five research objectives have been identified. These are important milestones, and prerequisites, in the process of answering the two research questions.

1.3.2 Research objectives

In order to address the situation recognition problem, it is first imperative to fully understand the problem. This puts requirements on knowing what situations are and how they can be represented. Moreover, it also requires that interesting situations can be represented and defined. It also requires that the complexity of the problem is understood, and that additional requirements that can be put on solutions to the problem have been identified. These are important aspects to consider. The first research objective thus reads as follows.

**Research objective 1:** Identify a suitable conceptualisation of the situation recognition problem in literature, and if one does not exist, suggest one. Furthermore, formally define the situation recognition problem and suggest a suitable representation of concurrent and partially temporally synchronised situations of interest. This objective includes analysing and synthesising existing theories.
Although very important, theoretical knowledge and a conceptualisation of the problem do not lead to an answer to any of the research questions that have been formulated. As previously argued, Petri nets seem to be a viable approach for solving the problem of recognising patterns that are partially temporally ordered. It is however not clear to which degree Petri net based techniques fulfil the requirements inherent in real time situation recognition. This calls for analysis and possibly suggestions of modification of existing algorithms and representations. The second objective thus read.

**Research objective 2:** Investigate, develop and suggest extensions to Petri net based recognition, to suit the problem of recognising situations of temporal and concurrent nature.

Theoretical investigations are important, but in order to successfully answer the research questions it is also important that algorithms and representations are investigated and compared empirically. This gives rise to two needs. First, there is a need for data and scenarios that are relevant in the application domain, and which can be used as a basis for comparison. Secondly, there is a need for tools that can be used to easily model and compare specific solutions with each other in a fixed and measurable way. The third research objective therefore reads as follows.

**Research objective 3:** Develop a test environment that contains necessary tools for evaluation. This environment could for example include scenarios, simulators and benchmarking capabilities.

In order to compare the performance and efficiency of Petri net based techniques to the situation recognition problem with rule based approach using the Rete algorithm, it is important to carry out empirical investigations using relevant measures. The fourth research objective therefore reads.

**Research objective 4:** Empirically investigate and compare the efficiency of Petri net based and rule based situation recognition.

Recall, in order for situation recognition to be viable in the long-term, it is important to be able to adapt existing knowledge concerning interesting situation types as more data and information is gathered. Moreover, it is also important to be able to fine-tune manually constructed definitions of what is interesting with respect to data and information, since experts will possibly have problems in defining precisely what they consist of. Genetic algorithms are kinds of techniques that can be used for learning and adapting complex concepts. Genetic algorithms are inspired by evolution in nature and consist of searching for promising solutions by evolving a population of candidate solutions over a number of generations using a set of genetic operators. Genetic algorithms investigate multiple different solutions in parallel. This can result
in efficient searches and increased chances of escaping possible local optima. These can be important aspects when it comes to the task of learning Petri nets, since the search landscape may be complex and not necessarily continuous. It is thus not enough to only rely on hill climbing. Furthermore, genetic algorithms are according to Kamp and Savenije (2006) one of the most successful optimisation techniques within soft computing. As already noted, genetic algorithms have previously been used for successfully evolving Petri nets. It is however of importance to investigate their suitability on the specific problem investigated in this thesis. The fifth and final research objective thus reads as follows.

**Research objective 5:** Analyse and develop algorithms and representations for adapting Petri net based definitions of situation types, and empirically investigate if and how the performance on the situation recognition task can be improved, and/or maintained, through the use of genetic algorithms.

### 1.3.3 Research methodology

The research methodology undertaken in this thesis is manyfold. Literature survey and analysis are relevant for most theses. As a first step, relevant sources were searched for in typical databases using the key word “situation recognition”. This did however only result in a few relevant publications. Another approach for finding relevant sources is to analyse the contents of relevant conference proceedings and journals. This can also be complemented with backwards citation chaining. Both of these approaches are relevant and have been used as a basis for finding sources relevant for this thesis work.

A literature survey has thus been carried out as follows. Situation recognition, as addressed in this thesis, is a problem within the information fusion domain. More specifically, it is related to higher level fusion and situation assessment in particular. The most relevant sources of literature within the information fusion domain consist of two journals \(^8\) \(^9\) and one annual conference \(^10\).

In order to identify relevant literature for the investigated problem, a survey has been carried out in five steps, using the online proceedings from the annual conference. In the first step, the titles of all papers, in relevant tracks, have been inspected and selected for further processing in cases where the paper seemed to be related to situation assessment. In the second step, the abstracts of the selected papers were read, and any paper that seemed related to situations, representation and processing in machines, were selected for further processing. The third step consisted of briefly reading the remaining papers to select those papers that were relevant for recognition. In the fourth step, selected papers were

---

\(^8\) Information Fusion, Elsevier.

\(^9\) Journal of Advances in Information Fusion, ISIF.

\(^10\) The International Conference on Information Fusion, ISIF.
read and analysed more carefully. In the last step, backwards citation chaining was carried out using the references from the identified papers, in order to widen the explored area.

In this thesis, two scenarios are utilised to conduct empirical evaluations of Petri net based situation recognition. These scenarios have been identified in collaboration with experts from our industrial partner Saab AB\textsuperscript{11}. The first of the scenarios is a fictive pick pocket scenario, and the second is a piloting boat scenario. The scenarios are interesting from an industrial perspective since they represent a move from identifying individual objects to recognising complex situations. The scenarios are intended to be used for carrying out both theoretical and empirical investigations. One potential weakness of this approach to research is however that the results may be biased. The validity of results may become biased when working in collaboration with external partners, since their views and preconceptions can affect choices, actions and interpretation of results. The benefits of the approach are however considered to outweigh the potential weakness of bias, since experts in the field have been actively involved in the tasks of identifying problems and developing solutions. They thus provide relevance and feasibility to the results.

The thesis work has been structured into five research objectives. These are important milestones for investigating the problem and for successfully answering the research questions. The nature of the objectives however varies, and thus, different research methods are suitable for each of the objectives. A brief synopsis of the methods that have been used is presented in the following.

Research objective 1 is concerned with a theoretical conceptualisation and definition of the situation recognition problem. The task of identifying concepts regarding situations, suggests the use of literature survey and analysis. Analysis in itself does however not address the full extent of the research objective, but synthesis is also required.

Research objective 2 is concerned with analysis and development of existing approaches to Petri net based recognition of complex patterns. Existing techniques for addressing the problem, or similar problems, needs to be analysed in light of the requirements that can be put on solutions to the problem. The result is expected to be either identified existing algorithms and representations, or suggested extensions to existing algorithms and representations.

Research objective 3 is concerned with the implementation of tools and scenarios that can be used for working with algorithms and representations for solving the situation recognition problem. This objective has two scopes: software and scenarios. The first scope concerns tools, and in order to develop good tools, it can be important to follow existing methodologies used in software engineering. The alternative would be to identify and compare existing tools. The second scope concerns the development of scenarios. The following five step method has been carried out for identifying and implementing suit-

\textsuperscript{11}Former Saab Microwave Systems, now Saab AB, business unit Electronic Defence Systems.
able scenarios: (1) a set of scenarios has been identified in literature, (2) the suitability of the scenarios has been discussed with experts from our industrial partner, (3) the scope and outline of the scenarios have been developed together with the industrial partner, (4) the designed scenarios have been implemented using software and (5) the resulting scenarios have been demonstrated for the experts, which have suggested improvements that have been implemented.

Research objective 4 consists of carrying out empirical investigations of Petri net based situation recognition with respect to efficiency. This may be carried out using quantitative experimental research. Hypotheses are formed. Quantitative experiments are carried out. Results are inspected and analysed. Hypotheses are rejected or accepted. An alternative would be to carry out a theoretical comparison with respect to complexity. Worst case complexities could be derived in this way. However, average complexities are hard to analyse due to their dependence on input data. This calls for empirical investigations where relevant data is used.

Research objective 5 is concerned with investigations into the use of genetic algorithms for learning and adapting Petri nets for situation recognition. This involves identifying suitable mechanisms in literature (which previously have been applied on similar problems), suggesting modifications of these approaches to suit the problem at hand, and lastly, to evaluate the chosen approaches and their suitability for the problem. The last step includes carrying out quantitative experiments, where hypotheses are: formed, investigated, and lastly, either accepted or rejected.

1.4 Thesis overview

This thesis carries out theoretical and empirical investigations with respect to the problem recognising situations using Petri nets. A holistic view is advocated throughout the thesis, in which object level data is analysed to extract relational information, which in combination with contextual information and a priori definitions of interesting situations, successfully is used for recognising situations in an efficient manner. The world is however constantly changing and today’s definitions of interesting situations may have changed tomorrow. This issue is also addressed through the use of genetic algorithms for learning definitions of interesting situations. These techniques may however also be used for refining manually defined situations of interest, with respect to data. This is also an important aspect, as it may be hard to specify exactly what an interesting situation consist of. Investigations are carried out when using artificial data and complex and dynamically generated situations of interest, as well as when using real world data for recognising real world situations.

A viable solution to the situation recognition problem has been defined as a solution that: (1) is able to achieve good performance, (2) is efficient, (3) allows for adaptation and (4) is relevant from a real world systems perspective. The investigations carried out in the thesis address each of these requirements
to some extent. The results point towards Petri nets as a viable solution to the situation recognition problem. The results of this thesis thus shows that it in the next generation of surveillance systems should be possible to include techniques for recognising patterns of partial temporal order in a timely fashion. By including such functions, new capabilities can be offered. This can in turn relieve operators who may focus their energy on even more complex tasks, thus making even better decisions.

1.4.1 Contributions

Three main contributions have been identified as an outcome of this thesis.

1. A conceptualisation and a definition of the situation recognition problem.

Many views and solutions for situation assessment and related topics exist in literature. Moreover, the situation recognition problem has previously been identified as important in the information fusion domain. There is however no clear conceptualisations and definitions that addresses the problem of recognising situations that may be of partial temporal definition. A conceptualisation and definition has been presented in this thesis and is argued to be a contribution to the information fusion domain.


An extended Petri net based technique for solving the real time situation recognition problem has been suggested in the thesis. It has been shown that this technique can be as efficient as approaches based on classical rule based techniques. Besides having a well founded mathematical theory, Petri nets however also have the benefits of allowing concurrency and temporal synchronisation to be easily represented and visualised. These are important aspects when building systems that should support human decision making.


A study on the use of genetic algorithms for learning Petri nets has been carried out. This study serves as a proof of concept that it is possible to improve the quality of complex structures using only limited data and information.

A number of minor contributions can however also be identified as a result of this thesis. The thesis presents a holistic view and a proof of concept of a situation recognition system based on relations extracted from track data. The design and architecture of a simulator for supporting research on situation recognition is presented. Moreover, the design and architecture of a platform for working with situation recognition is presented. This platform is also available as open source for other researchers to use. Lastly, the thesis suggests a dynamic complex genome representation for representing complex Petri nets
in genetic algorithms. It has been shown that this representation may have significant benefits, with respect to the time that needs to be spent on finding promising individuals, compared to classical bit string based genomes.

1.4.2 Limitations

As argued, the results in this thesis point out that Petri nets may be used as a viable technique for efficiently carrying out situation recognition. It has however not been shown that Petri nets actually do provide any significant benefits compared with rule based recognition based on the Rete algorithm, with respect to what can be represented and with respect to recognition performance. Furthermore, there may also exist other techniques that can be more efficient, e.g. the temporal constraint propagation technique proposed by Dousson et al. (1993); Dousson (2002); Dousson and Le Maigat (2007). Naturally, in future work it is interesting to compare with such techniques as well.

It is in this thesis assumed that Petri nets offer a more understandable form of representation for capturing complex situations that develop over time. Some support for this assumption can also be found in the work of Castel et al. (1996), who claim that Petri nets allow for concurrency and temporal synchronisation to be easily represented and visualised. A limitation in this work, however, is that this aspect has not been studied and verified empirically. The benefit of Petri nets with respect to understandability clearly needs to be verified with human user studies in future work.

The Petri net based technique has only been compared with one instance of a Rete based rule recognition technique (based on the extensions for temporal constraints proposed by Walzer (2009)). However, there has been much research on the Rete algorithm, since the early 1980’s, and it is possible that there are extensions that outperform Petri nets. There are also other extensions for including temporal aspects in the Rete algorithm, such as (Maloof and Kochut, 1993; Berstel, 2002; Schmidt et al., 2008).

Another limitation is that the two investigated techniques have only been compared on two example scenarios. There may very well be other scenarios for which the outcome would have been different. Still, this argumentation holds in many cases. The “no free lunch theorem” (Wolpert and Macready, 1997) is likely applicable to situation recognition too.

A limitation of the suggested technique is that it does not allow for explicit time intervals to be specified, e.g. event $A$ must occur at least 5 minutes before event $B$. Such constraints are possible in the extended Rete algorithm suggested by Walzer (2009). Still, extensions to the Petri net based technique, which would allow for such constraints, would not require too much alteration. Furthermore, the scalability of the Petri net based technique has not been examined to an extent that satisfies all possible doubts. For example, it has not been investigated how the number of interesting patterns affects the performance.
1.5 Thesis outline

The rest of the thesis is structured in five parts that outlined as follows.

Part I — Background

Chapter 2 presents an overview related to support for situation awareness. This chapter contains background material from a fusion perspective and can be skipped by readers that are familiar with this area.

Chapter 3 presents two types of techniques that can be used for situation recognition. The content include rule based systems and the Rete algorithm, and state transition techniques with a focus on Petri nets. The chapter also presents an overview of genetic algorithms. The chapter may be skipped by readers that are familiar with these topics.

Part II — Theoretical results

Chapter 4 provides a conceptualisation and a definition of the situation recognition problem.

Chapter 5 presents a Petri net based technique for situation recognition. This technique builds upon and extends existing approaches for recognition using Petri nets.

Chapter 6 suggests an approach for using genetic algorithms to learn Petri net situation templates. The chapter also suggests three different genome representations for evolving Petri nets for situation recognition.

Part III — Tools for evaluation

Chapter 7 presents a platform and framework for working with algorithms and representations for situation recognition. This tool is also used for comparing the performance of different algorithms with each other.

Chapter 8 presents a simulator for constructing data in support of research on situation recognition.

Chapter 9 outlines two scenarios, from data to events and recognition. These scenarios are used in subsequent empirical investigations.

Part IV — Empirical results

Chapter 10 presents empirical results related to recognition. It thus addresses the first research question.

Chapter 11 addresses the second research question and presents empirical results related to learning.

Part V — Conclusion

Chapter 12 concludes the thesis, discusses the work that has been carried out and outlines future work.
Part I

Background
Chapter 2
Support for situation awareness

This chapter discusses the topics of situation awareness and information fusion for supporting decision making. More specifically, the main focus is put on higher level fusion and on situations in particular. The chapter is extensive due to a broad problem and that there in literature exist many views of it.

2.1 Situation awareness

Within the surveillance domain, command and control (C²), command, control, communications, computers and intelligence (C⁴I), and other concepts with similar abbreviations, are typically seen as key enablers for using and consuming information for improved decision making in organisations of various types. Even though no precise agreement exists on what command and control really is (Roman, 1997), the view of Wallenius (2004) is adopted in this thesis. Wallenius (2004) puts forth that it concerns the recursive act of fulfilling a task assigned to an organisation by means of subtasks and available resources on lower levels of abstraction in the organisation.

Large amounts of information are often available in many organisations, and decision makers at each instance need to find the information about the present situation that is of importance for their specific subtask. This information needs to be analysed in light of experience and goals, with a purpose of establishing some form of awareness of how various pieces relate to each other and to the goal. This form of awareness can be used for decision making, and it is also known as situation awareness.

Many definitions of situation awareness have been proposed in the literature (Royal Aeronautical Society, 2003). The perhaps most well established definition however, is provided by Endsley who describes it as follows:
“Situation awareness is the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future.” (Endsley, 1988, p. 97)

Three levels of awareness are depicted in Endsley’s (2000) definition: perception, comprehension and projection. Perception is according to Endsley concerned with the perception of information from cues. Comprehension is concerned with meaning and encompasses how people combine, interpret, store and retain information, thus involving integration of multiple pieces of information and the determination of their importance with respect to goals. Projection is concerned with anticipating possible future situations and implications, based on the present situation and its dynamics. Endsley points out that experienced operators rely heavily on future projections: it is the mark of skilled experts.

Endsley (2000) argues that in its essence, situation awareness is concerned with knowing what is going on around you and knowing which information that is important. Temporal aspects of situations are also important in the notion of situation awareness. Endsley argues that perception of time and temporal dynamics associated with events is an important when forming situation awareness: (1) how much time is available until some event will occur or until some action needs to be taken, (2) how soon will a perceived element have an impact on the operator’s tasks and goals, and (3) the real world is dynamic and constantly changing, thus, an operator’s situation awareness must also constantly change, or be rendered outdated or inaccurate.

Situation awareness is by Endsley (2000) depicted as the decision maker’s internal model of a situation, acting as a precursor to decision making. Achieving and maintaining situation awareness involves many cognitive processes and is not something that directly can be provided by technical support systems (Endsley, 2000). The essential cognitive mechanisms involved in the formation of situation awareness, discussed by Endsley, are illustrated in Figure 2.1.

As can be seen in Figure 2.1, there are many cognitive processes involved in maintaining an internal representation of the situation: working memory and attention, goals and expectations, automacity, long term memory and mental models, and pattern matching. Pattern matching is perhaps the most relevant aspect for the work presented in this thesis. Endsley (2000) argues that there is considerable evidence that experienced decision makers make use of pattern matching to recognise situations as being of certain classes. This is closely related to theories of mental models and schemas, which may be coupled to specific situations that quickly can be recognised. Moreover, Endsley and Bolstad (1994) claim that there is evidence for the importance of pattern matching when distinguishing between fighter pilots with high and low levels of situation awareness. For a comprehensive discussion about the other concepts, see Endsley (2000).
2.1. SITUATION AWARENESS

Although situation awareness is very important, Endsley points out that the quality of decisions and outcomes are not necessarily tied to the awareness. It is possible to make incorrect decision even with perfect awareness and it is also possible to make good decision, even with poor awareness. Nonetheless, Endsley argues for the importance of situation awareness in decision making, and that there in many situations is a strong linkage between the two.

2.1.1 The OODA loop

Decision making can at a high level of abstraction be viewed as a continuously revolving loop, in which a decision maker observes information, analyses its meaning and impacts, decides on what to do, and finally, implements the decision. After this, a new revolution is started. Boyd (1987, 1996) captured this behaviour in a cyclic loop named the Observe - Orient - Decide - Act (OODA) loop, of which a generalisation is illustrated in Figure 2.2.

Figure 2.1: Illustration of the key cognitive mechanisms involved in achieving situation awareness (adapted from Endsley, 2000).

Figure 2.2: Boyd’s OODA loop (adapted from Boyd, 1996).
The OODA loop consists of four phases that illustrate different kinds of processes that are carried out when performing decision making:

- **Observe.** Data and information regarding the situation is gathered from many different sources in the observe phase.

- **Orient.** In order to establish some form of awareness, the data and information is analysed in the orient phase. Past information, culture, heritage and experience play key roles here.

- **Decide.** In the decide phase, some decision making paradigm is used to decide on what to do, in light of the established awareness.

- **Act.** Finally, the decision is carried out in the act phase, having an impact on the world, which again can be observed (although usually after some delay, since most actions are not carried out instantaneously\(^1\)).

### 2.1.2 Situation analysis

Situation awareness can be depicted in the OODA loop as a result of the processes occurring in the observe and orient phases. Roy (2001) makes this connection and refers to these two phases of the OODA loop as situation analysis. Situation analysis is by Roy defined as a process that provides and maintains a state of situation awareness, and it captures the phases of the decision making cycle that are concerned with understanding the world (Roy, 2001). Roy’s connection is illustrated in Figure 2.3.

![Figure 2.3](image)

**Figure 2.3:** Illustration of the connection between the OODA loop, situation analysis and situation awareness (adapted from Roy, 2001).

A situation model is continuously developed within the mind of a decision maker in the observe and orient phases of the OODA loop. The situation model is an abstraction of the world, and it forms the basis for situation awareness,

---

\(^1\)The OODA loop can in many ways be considered as being too simplistic as it for example does not capture the delays that are often experienced in connection to command and control situations, *c.f.* the work by (Brehmer, 2005), who uses the concept of sense making.
which in turn can be used to decide and act appropriately. Roy (2001) views the situation analysis process as consisting of many different tasks or subprocesses, with a purpose of continuously answering a plethora of questions such as what, who, why, when how, where, etc. Simply put, the situation analysis process can be broken down into many subprocesses, at many levels of abstraction, which together maintain the situation model. Roy (2001) argues that in order to create a meaningful synthesis of a situation, it is not enough to capture a representation with elements of interest in an environment, but also to capture how they relate. The situation model needs to capture a comprehension of the situation. This is a hard task in an ever increasing flow of information.

2.1.3 The information gap

The amount of data and information that is available in many complex decision making situations is today growing rapidly. Numerous systems, and subsystems thereof, are producing large quantities of information that is or can be made available to decision makers. More information does however not necessarily result in better or more efficient decisions. The situation is rather the opposite, or to put it in Naisbitt’s words:

“We are drowning in information but starved for knowledge.” (Naisbitt, 1984, p. 17)

A similar problem is identified by Endsley (2000) who refers to it as the information gap. Decision makers have huge amounts of data available at their disposal, but there are so many pieces that the important ones are missed. Decision makers are having difficulties in navigating through the available data to find the pieces that are needed, when they are needed. According to Endsley (2000), it is also becoming widely recognised that more data does not result in more information. Figure 2.4 illustrates one aspect of the information gap.

Figure 2.4: Illustration of the problem of finding the correct pieces of information that are needed (adapted from Endsley, 2000).
Naturally, the information gap needs to be closed in order to perform decision making more efficiently. It can therefore be concluded that there is a growing need for support in many complex decision making situations, such as for example search and rescue operations, civilian law enforcement, stock market trading, logistics and so on. By supporting decision makers in complex situation such as those, the decision cycle can be performed more efficiently and to a higher precision.

2.2 Information fusion

Techniques for information fusion have been identified as key enablers for providing decision support in the information age (Bossé et al., 2007). As the name suggests, information fusion involves a process that yields information of higher value by integrating and combining different pieces of information. For example, consider your average GPS receiver. A position estimate on its own does not constitute the greatest of support, but when combined with previously known map information, users can generally appreciate the system to a larger extent.

The concept of fusion is not a new one: corporate fusion, gene fusion, nuclear fusion and, even, Gillette fusion, are some examples. In nature we have it as the provider of life through the sun, which indeed is one massive fusion “reactor”. The concept however remains the same: individual pieces are fused together with a result that is of higher value than the constituent parts. Information fusion, as addressed in this thesis, is an offspring from the community of sensor-data fusion, or data fusion, where information from several sources generally are combined to predict the state of some aspect of interest to a higher degree of confidence. Data fusion is defined by Steinberg et al. as:

“Data fusion is the process of combining data to refine state estimates and predictions.” (Steinberg et al., 1999, p. 433)

The underlying assumption of data fusion is that synergistic differences in overlapping data can be exploited so that the resulting information is of higher value than otherwise would be possible (Steinberg et al., 1999; Dasarathy, 2001). Moreover, Lambert (2001) argues that machine data fusion delivers a technological basis for situation awareness.

2.2.1 The JDL model

Motivated by a general confusion within the community, the Joint Directors of Laboratories (JDL) introduced the JDL model of data fusion to clearly distinguish what it is (Llinas et al., 2004). The JDL model, as presented by Steinberg et al. (1999); Llinas et al. (2004), is illustrated in Figure 2.5.

In its original conception, the JDL model is a functional model that groups various aspects of fusion with respect to the types of processes that are carried
2.2. INFORMATION FUSION

Figure 2.5: The JDL model of data fusion (adapted from Steinberg et al., 1999).

out (Steinberg, 2009). The five levels (0-4) of the JDL model are by Llinas et al. (2004); Steinberg et al. (1999) defined as follows:

- **Level 0 processing.** *Sub-object assessment* concerns estimation and prediction of states of sub-object entities such as signals and features.

- **Level 1 processing.** *Object assessment* is concerned with estimation and prediction of states of discrete physical objects, such as vehicles, pedestrians and buildings.

- **Level 2 processing.** *Situation assessment* is concerned with estimation and prediction of relations among entities, such as aggregates, intent, acting on and context.

- **Level 3 processing.** *Impact assessment* concerns estimation and prediction of impacts, such as consequences of events on one’s own assets and goals.

- **Level 4 processing.** *Process refinement* focuses on management and change of the various processes, such as adaptive data acquisition and processing. This is also often viewed as a meta process controlling processes at the other levels (Hall and McMullen, 2004).

The JDL model attempts to: (1) provide a common frame of reference for discussing fusion related topics, (2) categorise different types of fusion processes, (3) help in recognising problems for which data fusion is applicable and (4) provide aid in extending previous solutions to new problems (Llinas et al., 2004; Steinberg and Bowman, 2004). According to Llinas et al. (2004), the basis for the JDL model was derived from how human beings organise and fuse data and information. Furthermore, the levels of the JDL model are a result of a partitioning that is based on the effects of changing levels of abstraction and changing levels of problem space complexity (Llinas et al., 2004).
2. SUPPORT FOR SITUATION AWARENESS

2.2.2 The λ JDL model

Although the JDL model was never intended to be a prescription for how to build a system (Steinberg and Bowman, 2004; Steinberg et al., 1999), quite naturally, it is often interpreted in that way. It does after all recognise the major aspects that are needed for assisting human decision makers in achieving situation awareness. We need to know what entities there are, where they are, what they are doing, what they will do and how this will affect us.

The JDL model has also greatly guided the community in advancing the foundations of data and information fusion. Lambert (2003b) however argues that the JDL model only offers limited instruction beyond its distinction of its levels and their content. Furthermore, Lambert (2003b, 2009) claims that the JDL model only provides a weak framework for fusion and that there is a demand for fusion systems involving both humans and machines. The original JDL model mostly addresses machine based issues (Lambert, 2009), and Lambert (2003b) therefore provides a decomposition in which level 0 is incorporated into level 1, and in which level 4 is absorbed by the remaining levels: 1, 2 and 3, respectively. Lambert (2001) connects the remaining three levels of the decomposed JDL model to the three levels of situation awareness defined by (Endsley, 2000), thereby making way for semi automatic fusion systems\(^2\). Lambert refers to this as the λ JDL model.

In the λ JDL model, Lambert (2003b) clearly distinguishes between the process of fusion and its results. The λ JDL model proposed by Lambert (2003b), is a process model, and reads as follows:

- **Object fusion** is the process of utilising one or more data sources over time to assemble a representation of objects of interest in an environment, and an **object assessment** is a stored representation of objects.

- **Situation fusion** is the process of utilising one or more data sources over time to assemble a representation of relations between objects of interest in an environment, and a **situation assessment** is a stored representation of relations between objects.

- **Impact fusion** is the process of utilising one or more data sources over time to assemble a representation of effects of situations in an environment, relative to our intentions, and an **impact assessment** is a stored representation of effects of situations.

**Object fusion**

Object fusion typically addresses three distinct problems: association, correlation and combination of data from single or multiple sources to produce refined

---
\(^2\)Similar reasoning have been expressed in our work (Niklasson et al., 2007, 2008).
state estimates and predictions (Hall and Llinas, 2001). Typically, sensor observations resulting from level 0 signal processing are associated with previously recognised objects. The outcome of this association process can result in new estimates of already recognised objects, new objects being recognised and thus estimated, or an acknowledgement that previously recognised objects no longer can be observed (Lambert, 2003b). An estimation process is also commonly used, in which future states of previously estimated objects is predicted at the current time step. In case a new observation and a predicted state lay within some bounded acceptance gate, then the new observation is associated with the previously recognised object, which is again estimated (Lambert, 2003b). Together, the association and estimation processes allow for states of objects to be tracked over time (Lambert, 2003b).

Lambert (2003b) states that the estimation process tends to be dominated by Kalman filters (Kalman, 1960), which gives a computationally efficient solution to linear filtering problems (Welch and Bishop, 1995). The Kalman filter provides a recursive solution to the problem of estimating the true state of a discrete time controlled process that is governed by a linear stochastic difference equation (Welch and Bishop, 1995). Kalman filtering typically proceeds in two steps that can be described with a feedback control loop. In the first step, the filter estimates the state of some process at time \( t \) (given the estimated state at time \( t-1 \) and a behaviour model). In the second step, the estimated state of the process at time \( t \) is corrected using feedback from measurements of the state at time \( t \) (Welch and Bishop, 1995). Other techniques that have been gaining interest include, but are not restricted to, particle filtering (Arulampalam et al., 2002; Gustafsson et al., 2002), and multi hypothesis tracking (Blackman, 2004).

In addition to estimating the true states of observables, object fusion may however also involve inferring more complex state variables, such as behaviour classes. Here we for example find work using hidden Markov models (HMM) for dynamic behaviour modelling (Chiao and Xydeas, 2004). Also relevant is the problem of anomaly detection, e.g. the detection of objects behaving outside of the norm, as defined by some model of normality. Techniques for addressing the anomaly detection problem are often based on machine learning techniques. Examples include the use of self organising maps and Gaussian mixture models (Riveiro et al., 2008), conformal prediction (Laxhammar and Falkman, 2010), domain specific abstract state space modelling (Brax et al., 2008), modified Fuzzy ARTMAP neural networks (Rhodes et al., 2005) and trajectory clustering (Piciarelli et al., 2005; Snidaró et al., 2006; Dahlbom and Niklasson, 2007).

Object fusion can today be considered quite mature. Higher level fusion, e.g. levels 2 and 3, has however received much less attention (Lambert, 1999, 2003b; Toth et al., 2008). We need to advance beyond the fact that we are interested in relations amongst entities, and effects of events on goals and assets.
2. SUPPORT FOR SITUATION AWARENESS

Situation fusion

Nowak and Lambert (2005) claim that object assessments typically consist of measurable properties that describe objects in the real world. Situation assessments are however usually based in the domain of symbols (Lambert, 2006b), consist of relations among objects and describe events in the world (Nowak and Lambert, 2005). Steinberg et al. (1999) argue as follows about the aggregates that are considered to be the result of level 2 fusion:

“The state of the aggregate is represented as a network of relations among its elements. We admit any variety of relations to be considered - physical, organizational, informational, perceptual - as appropriate to the given system’s mission.” (Steinberg et al., 1999, p. 435)

Situations are thus viewed as aggregates represented as networks of relations among elements. Similar structure can however also be represented in formal languages such as first order predicate calculus. This is also conveyed by Lambert (2003b), who argues that situations can be represented using expressions. As an example, within Lambert’s State Transition Data Fusion (STDF) model (Lambert, 2006b), situations are understood as “transitions in situation instances over time”. A situation instance is in turn understood as the state of affairs $\Sigma(k)$, at time $k$, consisting of a set of statements about the world in some formal language. Consequently, a situation at time $k$ is a set of states of affairs $\Sigma(k) = \{\Sigma(t) \mid t \in Time \land t \leq k\}$, e.g. a set of sets of statements about the world. Either way, however, there is a move from numeric representations to symbolic representations (Lambert, 2006b). This move gives rise to one of the grand challenges for information fusion, the semantic challenge:

“What symbols should be used and how do those symbols acquire meaning?” (Lambert, 2003c, p. 216)

This problem is similar to the classical symbol grounding problem (Harnad, 1990). Lambert (2003c, 2006b) argues that three dimensions are required for addressing the semantic challenge: philosophical, mathematical and computational. A philosophical theory is required, which properly specifies the domain of interest. A mathematical theory is required to impose formal structure on the specified philosophical theory. A computational theory is required in which the mathematical theory is implemented. Work on formally transcending these dimensions can be found in for example Nowak and Lambert (2005) and in Lambert (2006a).

Yet another highly related grand challenge for information fusion is identified by Lambert, the epistemic challenge:

“What information should we represent and how should it be represented and processed within the machine?” (Lambert, 2003c, p. 217)
In contrast to the semantic challenge, the epistemic challenge is according to Lambert (2003c) concerned with the choice of knowledge representation and suitable content. Analytic solutions where mathematical models are developed and used are popular in the fusion community for solving problems at the object level (Lambert, 2003a). Gaussian distribution and linearity are two often required assumptions for mathematical solutions to be feasible, and solutions without such approximations are often intractable and undecidable (Lambert, 2003a). Heuristics are thus needed to cope with the complexities of the environment at the object level. The complexity of higher level fusion however greatly exceeds the complexity of object fusion, which make purely analytic solutions even more difficult (Lambert, 2003a).

Lambert (2003a) instead suggests the use of cognitive solutions: find someone who knows how to solve the problem and develop a mathematical model of the process undertaken to solve it. Lambert (2003a) argues that a combination of analytical and cognitive solutions provides a suitable strategy for addressing the epistemic challenge. Lambert instead promotes functionalism (as suggested by Block (1980)). Lambert advocates that solutions does not necessarily need to mimic the precise processes undertaken by humans or the representation schemes used internally in the mind, but should instead aim at being functionality equivalent given some input and desired output (Lambert, 2003a). Subsequently, Lambert (2003a) argues for the use of a system founded on propositional attitude expressions: Attitude. Another example of a formal approach is the Situation Awareness Assistant (SAWA) (Matheus et al., 2005), which makes use of formal ontology to reason about situations. We will return to this in Section 2.3.2.

Less formal representation schemes are however also possible for representing the knowledge in some domain of discourse. One example is the technique using situation trees proposed by McMichael et al. (2004); McMichael and Jarrad (2005). McMichael and Jarrad suggest a technique where a situation tree is constructed recursively using sequence sets, and it is a mathematical formalism to represent battle, from tracks to higher level behaviours, by evolving force structures over time. It thus represents a knowledge base of the semantics of battle. Naturally, there are many other forms of representations that could be used as well; however, a suitable set of symbols is required as a foundation.

In conclusion, a suitable set of symbolic relations are needed, by which an abstraction of the domain of interest can be captured. Steinberg (2009) identifies five categories of relations: logical/semantic, physical, functional, conventional and cognitive. Similarly, Lambert (2003c) identifies five levels of relations: social, intentional, functional, physical and metaphysical. For more information about types of relations, see Lambert (2003c); Steinberg (2009).

The primary task for level 2 fusion is to infer the defined relations in various ways and to structure these into structures termed situations. For properties of objects, it is quite straightforward to construct symbolic representations through the use of unary predicates or similar (Lambert, 2009; Steinberg,
Given an object in an object assessment $o_1 :<x,y,z>$, three predicates can be formed $X(o_1) = x$, $Y(o_1) = y$, and $Z(o_1) = z$. Symbolic representations of other properties can be constructed in a similar fashion. For higher level relations it however becomes more complex. Some approaches that have been suggested in literature include: link analysis, clustering, inference and reasoning and procedural approaches.

*Link analysis* can be used for discovering associations amongst multiple entities using linkages in data, c.f. (Steinberg, 2009). *Clustering* can be used to analyse properties of observed objects in light of some mathematical model. An example includes the work of Das et al. (2006), who present two techniques for spatio-temporal clustering. An excellent review of data clustering can be found in Jain et al. (1999). *Inference and reasoning* can be used for inferring new relations from existing relations. This has also been promoted as a key capability for building generic systems for situation awareness (Matheus et al., 2005; Kokar et al., 2009). For a more detailed description, see Kokar et al. (2009). *Procedural* techniques can be used to derive relations using specific procedures that are hard coded. An example can be found in the work of Edlund et al. (2006), which in an agent based architecture use specific relation finder agents to extract relational information.

Naturally, relational information may also be provided in the form of facts, provided by external sources. In the military settings, human intelligence (HUMINT), communications intelligence (COMINT) and open source document intelligence (OSINT), are key sources of information, which involve information generated by humans and which often also have been interpreted and evaluated by human analysts (Steinberg, 2009).

**Impact fusion**

In the original versions of the JDL model, level 3 was envisioned as threat refinement (Hall and Llinas, 2001). This typically involves establishing the level of threat posed on one’s own assets by objects detected with level 1 fusion. Other types of threats may however also be considered, c.f. Steinberg (2009) and Looney and Liang (2003). The conception of level 3 has in later refinements of the JDL model changed to impact assessment, which instead aims at inferring possible impacts of the present situation on one’s own goals. Threat assessment can, according to Steinberg (2009), be seen as a class of impact assessment. Although Steinberg argues that threat, or impact, assessment is “a formally ill-defined and underdeveloped discipline”, much work has been carried out which is relevant to both the former and latter notions. Some examples are: the use of Bayesian networks (Laskey and Laskey, 2002; Johansson and Falkman, 2006, 2008a), the use of fuzzy belief networks (Looney and Liang, 2003), the  

These sources may indeed consist of human experts.
use of fuzzy logic (Johansson and Falkman, 2008b) and the use of game theory (Brynielsson, 2006).

Steinberg (2009) argues that level 3 fusion has been distinguished from level 2 fusion in at least three different ways: temporal, epistemic and ontological. A temporal distinction is promoted by Salerno (2007), who view level 2 fusion as being concerned with estimating the present situation, at time $t$, and where level 3 fusion is concerned with projecting the current situation $t + n$ steps into the future. Steinberg (2009) promotes an epistemic distinction, where level 2 fusion is concerned with inferring situations directly from observations, and where level 3 fusion is concerned with estimating situations indirectly from observations. Finally, Steinberg (2009) argues that Lambert, although estimating future states, promotes an ontological distinction where it is the utility of future situations, with respect to some agent, that is estimated.

In Lambert’s view, the task of impact fusion is twofold: (1) assessing situations that could or will arise in the future and (2) finding out any, possible, impacts that these future situations will have on our own intentions (Lambert, 2003b). In terms of Lambert’s STDF model, given a state of affairs $\Sigma(k)$ at time $k$, impact fusion attempts to model future expected states of affairs at times $> k$. For this purpose, (Lambert, 2006b) defines scenarios, which are expressed as sets of projected situations (state of affairs). From this, estimation of utility is depicted as follows. An agent $A$ has a number of intended effects, where each intended effect $\Psi_A(t \mid k)$, is defined in terms of future state of affairs that the agent $A$ at time $k$ intends to be the case at time $t$. Intended effects can be compared with predicated state of affairs, and the highest utility will be achieved if selecting course of action that minimises the difference between all intended effects and the predicted states of affairs, c.f. Lambert (2006b, 2007).

### 2.2.3 Situation science

Situation fusion/assessment spans many different topics, ranging from semantics and logic to relations. In an attempt to clarify matters, Steinberg (2009) proposes a taxonomy of what he calls situation science. This taxonomy consists of functions related to situation assessment, and it is illustrated in Figure 2.6.

At the top level, Steinberg (2009) divides situation science into situation semantics and situation assessment. Situation semantics is according to Steinberg (2009) concerned with defining situations in the abstract. The two subdisciplines of situation semantics concern the topics of ontology and logic. The topic ontology is the study of the categories of things that exist or may exist in some domain (Sowa, 2000). In the view of situations, an ontology specifies, according to Steinberg (2009), the types of entities that may be involved in situations in a domain of interest, as well as their dependencies towards each other. Logic concerns the use of for example first order logic for reasoning about instant-

---

4A situation may in itself be regarded as an entity, depending on the formal theory that is used.
2. SUPPORT FOR SITUATION AWARENESS

![Diagram of situation science taxonomy]

Figure 2.6: Taxonomy of situation science (adapted from Steinberg, 2009).

Situated entities defined in a formal ontology. An example of work related to situation semantics, is the work of Kokar et al. (2009), which defines a general ontology for situation awareness, using concepts developed in situation theory (Barwise and Perry, 1983; Barwise, 1989; Devlin, 2008). We shall return to this in Section 2.3.2. Moreover, we also find the work of Little and Rogova (2005) which defines a formal ontology for situation and threat analysis (STA) in crisis management.

The second subclass of situation science is situation assessment, which according to Steinberg (2009) is concerned with estimating and predicting structures of parts of reality. In comparison to the classical notion of level 2 fusion, situation assessment is here depicted as aggregating relational information into meaningful structures, which can be termed situations. Taxonomically, situation analysis matches the situation analysis process described by Roy (2001), for establishing a situation model. Steinberg (2009) however refers to it as being concerned with estimating and predicting relations among entities. Situation analysis is further decomposed into object- and relationship assessment, where object assessment refers to the classical notion of object assessment described in the JDL model. Steinberg (2009) however argues that it also concerns projections of entity states, thus incorporating parts of what classically is seen as impact assessment. Relationship assessment is according to Steinberg (2009) focusing on estimating the presence and characteristics of relations among entities in situations. This corresponds to the notion of situation assessment in the original JDL model (Steinberg et al., 1999; Llinas et al., 2004). Also depicted under situation assessment, is projection of relationships, which again incorporates parts of what generally is known as impact fusion.

5 Situations are by Steinberg (2009) seen as structures of parts of reality, and is based on themes developed in situation theory. Situation theory is presented in Section 2.3.2.
2.3. REPRESENTING SITUATIONS

Besides situation analysis, situation assessment is also divided into recognition, characterisation and projection of situations. Situation recognition is concerned with classifying situations according to situation types, and situation characterisation is concerned with estimating salient features of situations (Steinberg, 2009). In comparison with situation recognition, situation characterisation primarily involves abduction, whereas situation recognition primarily involves deduction (c.f. Steinberg (2009)). Finally, situation projection mainly concerns the estimation of future situations on the basis of present situations. Steinberg (2009) however also briefly discusses the use of counterfactual data for inferring past and present situations. In the classical notion, situation projection falls within the domain of impact assessment.

In conclusion, situation fusion is (if subtracting object and impact fusion) concerned with estimating relations amongst objects (relationship assessment), aggregating these relations into structured parts of reality, e.g. situations, determining if these structures corresponds to previously known situation types (situation recognition), or determining if they can be characterized and classified using salient features (situation characterization). Nowak and Lambert (2005), however, argue that ability of reasoning also is important.

2.3 Representing situations

As we have seen, object fusion targets the information gap with its goal of assembling a representation of objects of interest in an environment. Besides association, correlation and combination of data, object fusion is clearly also concerned with focusing on interesting objects. Similarly should situation fusion target the information gap by assembling a representation of situations of interest in an environment, and besides inferring and analysing relational information, it should necessarily also be concerned with focusing on interesting situations. This is also the focus of situation recognition, which has been pointed out as an important function of situation assessment (Steinberg, 2009), as well as a core component in situation management (Jakobson et al., 2007).

In order to carry out the task of recognising situations, it is necessary that we properly understand what situations are. Besides being an ordinary word in many languages, formal definitions of the situation concept have been defined within many contexts, of which situation calculus (McCarthy, 1963; McCarthy and Hayes, 1969), perhaps is the earliest (Jakobson et al., 2007).

2.3.1 Situation calculus

Situation calculus (McCarthy and Hayes, 1969) is according to Russel and Norvig (1995) a particular approach for describing change in first order logic. In situation calculus, a situation \( s \) is considered to be the complete state of

---

6Lambert’s terminology will from here on be used to clearly separate processes from their results.
the universe at an instant of time. However, since the universe is too large, situations are according to McCarthy and Hayes never completely described. Rather, partial information about situations is given using fluents, which are used to state facts about situations. As an example sunny(x) is a fluent such that sunny(x, s) is true if it is sunny at location x in situation s. Fluents are functions over the domain of situations\(^8\) S with a range of either \(S\) (situational fluents) or of \((true, false)\) (propositional fluents). Causality can be expressed using a fluent \(F(\pi)\), where \(\pi\) is a propositional fluent, and where \(F(\pi, s)\) states that the situation s will be followed by a situation which satisfies \(\pi\). An important part in situation calculus is, according to McCarthy and Hayes, the situational fluent result\((p, \sigma, s)\), where \(p\) is an agent, \(\sigma\) an action or a strategy, and where \(s\) is a situation. The value of result\((p, \sigma, s)\) is the resulting situation of \(p\) carrying out \(\sigma\) in \(s\). The result fluent may be used to express laws of ability with effect axioms, which in turn allows for plans to be conceived for achieving future situations. It is however also necessary to state which fluents that stay the same when actions are carried out, i.e. which fluents that are action invariant. This is achieved using frame axioms, which in turn gives rise to the frame problem (c.f. McCarthy and Hayes (1969)). The frame problem refers to the potentially large number of frame axioms needed to properly describe actions and effects.

Reiter (1991) provides a solution to the frame problem for deterministic actions\(^9\), using successor state axioms. In subsequent work, a slightly different view on situation calculus is offered (c.f. Levesque et al. (1998)). A situations \(s\) is here viewed as a history consisting of a finite sequence of primitive actions \(a\). Levesque et al. (1998) define \(L_{sitcalc}\) as a second order language\(^{10}\) with three disjoint sorts: action, situation, and object, for actions, situations, and everything else, respectively. Besides the standard logical connectives, countably infinitely many variable symbols are allowed of each sort, two function symbols of sort situation are admitted: \(S_0\) denoting the initial situation, and binary function do : action \(\times\) situation \(\rightarrow\) situation denoting successor situation do\((a, s)\) resulting from carrying out \(a\) in \(s\), a binary predicate \(\sqsubseteq\) : situation \(\times\) situation, which defines and ordering on situations such that \(s \sqsubseteq s'\) means that \(s\) is a proper subhistory of \(s'\). A binary predicate Poss : action \(\times\) situation is also used, for which Poss\((a, s)\) means that action \(a\) can be carried out in situation \(s\). Also admitted are for each \(n \geq 0\) countably infinitely\(^{11}\) many: predicate symbols with sort \((\text{action} \cup \text{object})^n\) denoting situation independent relations such as cat(Felix), function symbols of sort \((\text{action} \cup \text{object})^n \rightarrow \text{object}, den-

---

\(^7\)If and only if.

\(^8\)\(S\) is by McCarthy and Hayes (1969) defined as the set of all situations.

\(^9\)According to Russel and Norvig (1995) a similar solution to the frame problem, or representational frame problem, was simultaneously and independently suggested by Elkan (1992).

\(^{10}\)The original situation calculus (McCarthy, 1963) is a first order language.

\(^{11}\)Any set whose members can be identified one at a time, through some prescription, is called a countable infinite set (Wolfram Math World, 2010).
noting situation independent functions such as \( \sqrt{x} \), function symbols of sort \((\text{action} \cup \text{object})^n \rightarrow \text{action}\), denoting actions such as \(\text{fire}(x, y)\), predicate symbols with arity \(n + 1\) and sort \((\text{action} \cup \text{object})^n \times \text{situation}\), denoting relational fluents such as \(\text{married}(\text{Mary, John, } s)\), function symbols of sort \((\text{action} \cup \text{object})^n \times \text{situation} \rightarrow \text{action} \cup \text{object}\), denoting functional fluents such as \(\text{population}(\text{Sweden, } s)\). Additionally, a number of axioms and theorems are defined, as well as basic theories and regression (Pirri and Reiter, 1999; Levesque et al., 1998). Furthermore, a logic programming language based on situation calculus, GOLOG, was introduced by Levesque et al. (1997), and it can be used to for example implement controllers in dynamic domains (Pirri and Reiter, 1999).

### 2.3.2 Situation theory

A slightly different view on situations is provided by Barwise (1981); Barwise and Perry (1983), who in their situation semantics look at situations from the perspective of understanding language using logic (Devlin, 2008). Situations come in many flavours: “events and episodes are situations in time, scenes are visually perceived situations, changes are sequences of situations, and facts are situations enriched (or polluted) by language” (Barwise and Perry, 1980)\(^{12}\). The underlying mathematics in situation semantics is called situation theory (Devlin, 2008). Although its name suggests it is a theory, Devlin points out that situation theory rather is a set of mathematically based tools for analysing context and information. Situations are in situation theory, according to Barwise (1989), the name used for portions of reality that agents find themselves in and about which they exchange information. Moreover, Barwise (1989) argues that one important aspect in situation theory is that situations can be regarded as first class objects; they can have properties and stand in relations.

Information is in situation theory, according to Devlin (2008), always information about some situation, and the information an agent has about a given situation will only be a limited part of all the information that is theoretically available. Information about situations is according to Devlin given using infons, which are discrete informational items. An infon is defined as \(\sigma = \langle\langle R, a_1, ..., a_n, 0/1 \rangle \rangle\), where \(R\) is an \(n\)-place relation, \(a_1, ..., a_n\) are objects appropriate for \(R\), and where \(0/1\) indicates the polarity of the infon (1: the objects stand in the relation in a given situation, 0: the objects do not stand in the relation in the given situation). Devlin argues that infons are not themselves true or false, but instead, information may be true or false about a specific situation. The proper terminology is, according to Devlin (2008), that a situation \(s\) supports a specific infon \(\sigma\). This is denoted as: \(s \models \sigma\)\(^{13}\). Infons may be recursively combined using conjunction, disjunction, and situation-bounded

\(^{12}\)This citation is requoted from Devlin (2008, p. 2), since the original publication by Barwise and Perry (1980) has not been found.

\(^{13}\)\(\models\) is not to be interpreted in a model theoretic view.
existential and universal quantification, to construct compound infons. Similar to the revised situation calculus (Reiter, 1991), situation theory is many sorted\textsuperscript{14}. The various sorts of objects consist of *individuals, relations, spatial locations, temporal locations, types, parameters*, and *situations*.

Devlin (2008) argues that an aspect of intelligent behaviour is the capability of an agent to individuate between different types of objects, various activities and so on. This is according to Devlin captured in situation theory through the use of a number of basic types, loosely corresponding to the sorts, with the addition of types for *infons* and *polarities*. An infinite number of basic parameters are allowed for each basic type, except for the type of parameters. Parameters are used as place holders for specific entities (including instances of situations), and anchoring is used to link parameters to actual entities (Devlin, 2008).

Inference is enabled in situation theory through the use of constraints. Constraints are links between situation types, such as natural laws, logic and rules (c.f. Devlin (2008)). For example, *rain requires clouds*, can be represented with a constraint $S \Rightarrow S'$, where $S$ is the type of situation where it is raining, and where $S'$ is the type of situation where it is cloudy. An agent can then infer that it is cloudy, by observing that it is raining. Formally, if there is a situation $s$ of type $S$ and $S \Rightarrow S'$, then there is also a situation $s'$ which is of type $S'$, where $s$ and $s'$ are situation instances and $S$ and $S'$ are situation types.

Devlin (2008) argues that the purpose of situation semantics is to study utterances. These are analysed using three classes of situations: utterance situations, resource situations, and focal situations. *Utterance situations* denote the context in which an utterance is made and received. *Resource situations* denote external situations referred to when describing a situation. *Focal situations* denote the situation that is being described. For example, if Kermit tells Miss Piggy: *the chef that I heard mumbling in yesterday’s show is on the stage now*, the utterance situation is the context in which Kermit tells Miss Piggy, the resource situation is the situation referred to that occurred yesterday, in which Kermit heard a chef\textsuperscript{15} mumbling in the show. The focal situation is the situation presently being described: the chef on stage now.

Perry (1999) argues that although situation semantics, and situation theory, has been used to analyse many linguistic phenomena, the liar paradox\textsuperscript{16}, heterogeneous reasoning and representation, and the nature and structure of information and action, it has been more accepted for its main themes: partiality, realism and the relational structure of meaning. These are concepts that “have been incorporated into the (generally) received wisdom of philosophy and linguistics.” Perry (1999)

\textsuperscript{14}The universe is not considered to only consist of a homogeneous collection of objects, but may instead consist of multiple collections of objects of different sorts.

\textsuperscript{15}Obviously, I am referring to the Swedish chef.

\textsuperscript{16}If, *this sentence is false*, is true, then it is false, resulting in that it is true, ad infinitum.
2.3. REPRESENTING SITUATIONS

**Situation theory ontology**

The work of Kokar et al. (2009) builds upon interpretations and extensions to situation theory as put forth by Devlin (2008). Kokar et al. argue for a general ontology for situation awareness, which they call the Situation Theory Ontology (STO). Situation theory is in STO expressed\(^\text{17}\) as an ontology in the Web Ontology Language (OWL)\(^\text{18}\). Kokar et al. claim that an advantage of an ontology based approach is that it allows for inference of new facts from existing facts. The particular choice of OWL is that it is one of few standardised languages that is supported by common software (Kokar et al., 2009).

STO implements the basic types of situation theory as classes in OWL. As an example, there is a class of type individual which can be extended to more specific types of individuals through the incorporation of subclasses. Similarly, classes for the other basic types are also implemented, and these can again be further divided into subclasses of subtypes. Constraints can in STO be modelled using subclass relationship between classes. If for example situation of type \(S_1\) is the case whenever a situation of type \(S_0\) is the case, then this can be modelled with a constraint: \(S_1 \Rightarrow S_0\). Such constraints can in STO be modelled by having class \(S_1\) as a subclass of \(S_0\).

In earlier work (Matheus et al., 2005) on their situation awareness assistant, SAWA, the Semantic Web Rule Language\(^\text{19}\) (SWRL) is also used as a complement to the basics of OWL. SWRL is an extension used on top of OWL for defining semantic rules that can be used for inference. As an example, OWL does not allow for concepts such as: 
\[
\text{Male}(x) \land \text{Sibling}(x, y) \land \text{Child}(x, z) \Rightarrow \text{Uncle}(x, z)
\]

This is however possible when using SWRL. Matheus et al. however state that SWRL is limited to using binary predicates. In later work (Kokar et al., 2009) another rule language and inference engine is reported to be used, which is called BaseVISor.

### 2.3.3 Situation management

A closely related discipline to information fusion, and to situation fusion in particular, is that of situation management (Jakobson et al., 2007). Situations are in situation management viewed as objects of management. Jakobson et al. view situation management as:

> “a framework of concepts, models and enabling technologies for recognizing, reasoning about, affecting on, and predicting situations that are happening or might happen in dynamic systems during pre-defined operational time.” (Jakobson et al., 2007, pp. 18-19)

\(^{17}\)Kokar et al. (2009) mention that STO does not faithfully implement situation theory, rather resemble it, since STO is based on OWL, which is based on model theory.

\(^{18}\)More information about OWL is available at: http://www.w3.org/TR/owl-features/.

\(^{19}\)For more information about SWRL, visit: http://www.w3.org/Submission/SWRL/.
The core in situation management consists of situation modelling, situation recognition and situation reasoning. Some examples of applications areas for situation management are according to Jakobson et al. (2007): tactical battle space management, disaster situation management, network and systems management, and infrastructure and cyber security management. In comparison with information fusion, the focus of situation management is on higher level topics. Less focus is thus put on signals and objects. Furthermore, situation management also model the complete OODA loop, from sensing to actions.

Jakobson et al. (2007) depict three important aspects of situation management, which depend on the goals of the system: investigative, predictive and control. The investigative aspect is concerned with retrospective analysis of causality, and aims at determining why certain situations have occurred. The predictive aspect of situation management aims at projecting possible future situations, and the control aspect aims to achieve or keep goal situations.

In the control mode, Jakobson et al. depict a deliberative control loop involving four steps: (1) sensing and information collection, (2) perceiving and recognising situations, (3) analysing past situations and predicting future situations, and (4) reasoning, planning and implementing actions so that desired goal situation is reached. The control loop is illustrated in Figure 2.7, and it is according to Jakobson et al. based on the architecture of intelligent control systems discussed by Albus and Meystel (1996). It however also has high resemblance with the OODA loop (recall Figure 2.2 and Figure 2.3).

Figure 2.7: A general process loop in situation management (adapted from Jakobson et al., 2007).
Because of the large amounts of raw data being collected in many domains, event streams need to be processed and correlated to produce situational events, i.e. events at the domain level. This is carried out in an information correlation stage, which is part of a sensing and collection step. Additional dimensions of situation management are according to Jakobson et al.: situation awareness (see Section 2.1), situation resolution, situation acquisition and situation learning. Situation resolution is concerned with action planning and implementation for achieving goals. Situation acquisition and learning are the main sources for building the knowledge structures required for situation management processes.

Jakobson et al. also provide a general framework for situation modelling, in which they separate between structural, dynamic and representational components. Structural components identify individual objects and systems in the world, and include entities, attributes, classes and relations. The dynamic components define the behaviour of the structural components in time, using the concepts of events, basic situations and compound situations. Lastly, the representational component is depicted as being orthogonal to the two former components and it consist of the languages, interpreters and environments for defining concepts.

Situations in situation management

Jakobson et al. (2007) consider a world\(^\text{20}\) that can be sensed, perceived reasoned about and affected. This world is considered to be populated with entities that are engaged in any number of relations (structural, causal, spatial, temporal, etc.), and which possess separable attributes and associated values. Moreover, entities can interact with each other and they can change their state over time. An entity is defined as a “thing of significance, either real or abstract that has distinctive existence.” Formally, an entity \(e\) is by Jakobson et al. represented with a set of attributes \(\{a_1, a_2, \ldots, a_n\}\), where each attribute \(a_i\) is a collection of properties of interest, such as name, type, value, default value, etc. The value property describes the actual attribute value, and is represented as a triplet: <value, certainty estimation, time>\(^\text{21}\). Jakobson et al. also define situational attributes as a subset of entity attributes. These are declared, computed or defined depending on application context. Furthermore, entities are considered as being dynamic time dependent objects with a time of creation \(t'\) and a time of clear \(t''\), resulting in a lifespan \(\delta = (t', t'')\) for entity \(e\). Naturally, any attribute value of an entity may only be defined within the lifespan of the entity.

Relations are by Jakobson et al. considered to be mental abstractions that link a certain number of entities with each other. Formally, a relation is defined as \(R \subseteq E_1 \times \ldots \times E_m\), where \(E_i \subseteq U\) is a class of entities in the universe \(U\).

\(^{20}\)This world can be real or imaginary (Jakobson et al., 2007).

\(^{21}\)The time property is either a point in time or an interval of time.
Jakobson et al. argue that it in many practical applications can be required to view relations and entities. In this view, relations are characterised by sets of attributes \( \{b_1, b_2, \ldots, b_h\} \), as well as all features that are attached to attributes of entities. Similar to entities, relations are also considered to be dynamic time dependent objects with a lifespan \( \delta = (t', t'') \). Furthermore, the lifespan \( \delta \) of a relation \( e_iRe_j \) should be a subset of the union of the lifespans of its linked entities: \( \delta \subseteq \delta_i \cap \delta_j \).

Three types of base situations are considered by Jakobson et al. (2007):

- **Entity based situation**, \( S_e(d) \), is defined over the set of situation attributes of entity \( e \) and have a duration \( d \subseteq \delta \), where \( \delta \) is the lifespan of entity \( e \). More specifically, an entity based base situation is a collection of time stamped attribute vectors \( \{a_1, \ldots, a_n\} \) (the situation attributes) of an entity \( e \), in which all the values are the same.

- **Relational entity based situation**, \( S_R(d) \), is defined over the set of situational attributes of relation \( R \), and have a duration \( d \subseteq \delta \), where \( \delta \) is the lifespan of relation \( R \). Similarly, a relational entity based situation is a collection \( \{b_1, \ldots, b_h\} \) situational attributes of relation \( R \), in which the attribute values are the same.

- **Relational situations**, \( S_{e_i,e_j}(d) \), is defined over relations between objects: \( e_iRe_j \), where \( d \subset \delta \) is the lifespan of the situation, which is a subset of the union of the lifespans of the entities: \( \delta = \delta_i \cap \delta_j \). Relational situations are situations where only relations between entities are considered, and not any attributes of the relations themselves.

Additionally, Jakobson et al. also consider *compound situations*. These are complex situations that can be constructed from other situations through the use of the set theoretical union and intersection operations. Lastly, Jakobson et al. also consider events. These define transitions of a system from one state to another state, or in terms of situation management, from one situation to another situation. In a practical view, an event is considered to be a time stamped piece of information representing a change in state of an object or similar.

### 2.4 Recognising situations

The act to *recognise* is defined by Merriam-Webster online as: “to perceive to be something or someone previously known”, by YourDictionary.com as: “identification of some person or thing as having been known before or as being of a certain kind”, and by The Free Dictionary as “to know or identify from

---

22 Jakobson et al. (2007) argue that it in most practical applications is enough to consider binary relations.
past experience or knowledge\textsuperscript{23}. Naturally, something in this case referring to situations or types of situations, a priori defined or experienced.

Situations can according to Steinberg (2009) be implied by the states of a set of objects and relationships that may exist among them. The black king at G8, the white queen at F6, a white tower at H2, a white bishop at B3, ..., may be an indication of check mate. Steinberg (2009) formally defines situation recognition as:

\begin{equation}
\exists x_1, \ldots, \exists x_n [R_1^{(m_1)}(y_{11}, \ldots, y_{m_1}) \land \ldots \land R_k^{(m_k)}(y_{1k}, \ldots, y_{m_k}) | y_{ij} \in x_1, \ldots, x_n, 1 \leq j \leq k, 1 \leq i_j \leq m_j] \Rightarrow s,
\end{equation}

where $s$ is a situation of some type, $R_i^{(j_i)}$ are relations of arity $j_i$, and where $x_k$ are objects. Although Steinberg (2009) argues that there in general are too few ontologies that would enable such rules to be used, it is questionable if ontologies alone would provide efficient solutions for recognising situations. In its essence, situation recognition however concerns the problem of matching relational structures found in data with stored structures representing defined classes of interesting situations (c.f. Steinberg (2009)).

Similar problems have been addressed within many domains and have, possibly, been termed as many different names. Some of these names include: situation recognition, multi agent activity recognition, complex event recognition, scenario recognition and chronicle recognition. In the following, a brief recapitulation of some techniques used for addressing these, different, but very similar problems, is presented.

### 2.4.1 Graph matching

Graph matching is according to Steinberg (2009) a frequently used technique for situation recognition. Graph matching basically consist of two steps: (1) extract a relational structure in the form of a semantic net from data (termed data graph) and (2) comparing the extracted structure with a set of stored structures of interesting situations (termed template graphs) to find the best match (Steinberg, 2009). Graph matching problems are typically NP-complete (Sudit et al., 2006), and Stotz et al. (2009) thus argue that graph matching algorithms in general have problems with performance, which makes them suitable only for forensic type applications and retrospective analysis. Nonetheless, much work has been carried out on graph matching for many years (for a comprehensive review see Conte et al. (2004)).

In the fusion community we for example find the work of Sudit et al. (2005, 2006), which presents a framework, INFERD\textsuperscript{24}, for addressing the problem of 23Several more definitions are available, such as for example: “To accept officially the national status of as a new government” (The Free Dictionary). However, such definitions are of less interest in this thesis.

24INformation Fusion Engine for Real time Decision making.
real time detection of multi stage coordinated cyber attacks. Templates are defined a priori for different types of cyber attacks, and the real time task is to process information and to evaluate if the behaviours defined in any of the templates is possibly occurring. To do this, they calculate credibility values, which together with a ranking of templates provide security analysts with a situational assessment of the network environment. Moreover, Nagi et al. (2006) describe INFERD as a suitable formalised graph matching technique for addressing asymmetric warfare problems. Sudit et al. (2006) also present TruST25, which builds upon graph matching techniques. TruST is a search heuristic intended for forensic graph matching.

Also related to graph matching is the work of (Aleman-Meza et al., 2005), which addresses the problem of finding complex illegal activities such as money laundering, theft operations and terrorism activities. Templates are defined as directed graphs, where nodes represent different entities and edges represent relationships between these entities. Core templates are defined to be a non empty subset of a template, and should capture the most important aspects of an activity. A two step procedure is used, where the first step concerns matching core templates with data. In the second step, matched core templates are expanded to include all aspects of interest in its associated template. Similarity is calculated using node and edge similarity.

Stotz et al. (2009) present an incremental subgraph matching approach, which is shown to generate solutions of similar quality to regular graph matching; however, it lowers the computational demands of the matching procedure. The running time is shown to be relative to the size of the change in input of the input data graph. By using techniques such as the one presented by Stotz et al., graph matching can be used in applications with real time demands.

Besides the potential problems with complexity and real time constraints, another problem with graph based approaches may be that it can be hard to represent and recognise situations consisting of relations between objects, where there also may be temporal constraints between relations.

2.4.2 Probabilistic techniques

Probabilistic approaches have become very popular within the fusion community, to for example recognise situations, threats, groups, scenarios, and so on. One of the main strengths of probabilistic approaches is that they allow for reasoning and inference in uncertain environments. In the Bayesian view, Bayes’ theorem is typically the cornerstone. It is according to Russel and Norvig (1995, p. 426), the foundation of all modern AI systems for probabilistic inference. Bayes’ theorem allows one to calculate the posterior probability of some hypothesis $h$ to be true, given observed data $D$, prior probability of $h$ being

---

25Truncated Search Tree.
2.4. RECOGNISING SITUATIONS

true, probability of observing $D$ given $h$, and probability of observing $D$ (c.f. Mitchell (1997)). Formally, Bayes’ theorem is defined as:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)},$$

where $P(x|y)$ denotes the conditional probability of $x$ given $y$. Through the use of Bayes’ rule, unknown probabilities can be computed from known probabilities. In practice, the combination of many pieces of evidence may however require that a very large number of conditional probabilities are evaluated, since variables in the domain may be causally dependent on each other. However, the use of conditional independence assumptions amongst variables can significantly lower the complexity of the problem (Russel and Norvig, 1995).

Bayesian networks provide a mechanism for specifying conditional dependencies amongst variables, and thus also for modelling conditional independence. A BN is a directed acyclic graph, where nodes represent variables and where edges denote conditional dependence between variables (for a general introduction see for instance Mitchell (1997)). For each node, a conditional probability table is formed, which describes the probability distribution of the node given probabilities for its direct ancestors. Given evidence for some variables that have been observed, the posterior probability distributions for other variables can be determined.

In the maritime domain, we for example find the work of Fooladvandi et al. (2009), which uses BNs for signature based detection of maritime situations based on AIS data. A BN is used to specify the signature of an activity using simpler components, for which probabilities are derived procedurally. Given the probabilities of each ingoing component, the probability of a situation with the specified signature taking place, can be inferred.

Wright et al. (2002) argue that most applications using BNs are based on template models consisting of a static set of variables and their dependencies. This is however considered insufficient in large and dynamic domains, and Wright et al. (2002) argue for the use of BN fragments (BNFrags), which should capture meaningful pieces that can be parameterised. Moreover, Wright et al. (2002) argue for the use of a knowledge base consisting of multiple BNFrags, which can be combined to build appropriate BNs depending on context. Laskey et al. (2001) describe Multi Entity BNs (MEBNs) which are an extension to BNs, for representing knowledge in domains where for example the number and types of entities cannot be specified in advance, but are situation dependent. Situation specific MEBNs are built from BNFrags. These concepts and technologies have been used for detecting insider threats (Laskey et al., 2004) and terrorist threats (Laskey and Levitt, 2002).

Besides BNs, there are other probabilistic approaches that can be used for recognition. For example, Glinton et al. (2006) use Markov random fields for finding potential instances of doctrinal templates in a set of data. Although
their focus is on intent inference, the matching procedure maps quite nicely to the situation recognition problem. Glinton et al. (2006) cast the problem as a template matching problem, in which both the input data and templates are considered to be relational structures that connect a set of nodes through relations. Nodes can be both events and entities, and edges are relations such as for example spatial proximity and temporal ordering. The problem they face is to identify an optimal mapping between relational structures in the input stream and in the templates. To achieve this, Glinton et al. use simulated annealing and Markov random fields. A potential problem is however that this method requires that we have access to a complete set of input data when performing the matching. This is not often the case, as data continuously are reported from various sensors, resulting in that the assessment needs to be carried out in its full extent whenever new information should be included.

2.4.3 Temporal constraint based techniques

Temporal constraint based techniques aim at explicitly reasoning about time or intervals of time, in order to carry out scheduling or recognition of chronicles, scenarios or situations. These problems are characterised by the existence of distinct temporal constraints between various components. One way of modelling such a domain is to use temporal constraint networks (Dechter et al., 1991). Temporal constraint networks provide a means of specifying temporal constraints using a graphical structure. Temporal constraint networks can amongst other things be used for representing and addressing simple temporal problems. In a simple temporal problem, there is a set of time point variables \( X = \{x_1, \ldots, x_n\} \), representing for example a number of events, a set of binary constraints over the time points \( C = \{c_1, \ldots, c_m\} \), where each \( c_i \) defines a temporal constraint between two variables \( x_i, x_j \) using two time points. A consistent solution to the simple temporal problem is a set of time points for which all of the constraints are fulfilled. As mentioned, temporal constraints can be used for recognising situations that take place over time.

In their work on situation recognition for environment surveillance, Douson et al. (1993) addresses the situation recognition problem as an online search for specific patterns of activity in an event stream. The basis of their approach is propositional reified logic, in which a set of propositions are temporally qualified (true, false, on, off), and in which time points are used as the elementary primitive. The authors use event patterns to denote change in truth value of propositions, and events for denoting time stamped instances of event patterns. In their work, a situation model is a set of event patterns and a set of temporal constraints (an event occurring before another event, the time between two events being less than some value). Furthermore, a situation model can also describe contextual constraints that need to be true (for example a robot being in a specific room before even looking at a pattern where it picks up a box). The situation model thus describes what they are interested in recognis-
2.4. RECOGNISING SITUATIONS

The problem they face is that of identifying subsets of the event stream, which matches the set of event patterns, temporal constraints and contextual assertions of a situation model. The heuristic in the process for recognising situations that Dousson et al. (1993) present is based on temporal constraint propagation. This is used for managing the potentially large number of instantiations of situation models.

In later work (Ghallab, 1996; Cordier and Dousson, 2000; Dousson, 2002; Dousson and Le Maigat, 2007) the focus has shifted to recognising chronicles in network surveillance applications. The terminology has been changed accordingly, to chronicle recognition. Moreover, extensions for garbage collection, hierarchical recognition based on subpatterns and learning of chronicles is also proposed. Chronicle recognition, as discussed in Ghallab (1996), is also exploited in the work of Heintz (2009), which addresses the problem of recognising traffic situation based on information provided by a set of unmanned aerial vehicles (UAVs) monitoring some area. A typical situation in this setting is a wreck-less overtaking, where one car passes another in a less than optimal way.

2.4.4 Rule based techniques

In rule based techniques, a set of if then rules are used for denoting various patterns of interest. Rules may be complex and describe a complete pattern of interest. Rules may however also be simple, but when used in combination they can be used to find an interesting pattern. The use of simple rules is for example investigated by Ehlert et al. (2002), who look at two different approaches for recognising situations in their work on human like flight bots. The underlying parameters that are used in their rules include for example air-speed, vertical speed, throttle, brakes status and gear status. Combinations specific values of these parameters are used as an indication of a situation in progress.

A potential problem is that situations of temporal nature cannot be encoded. This is addressed by for example Edlund et al. (2006), who suggest using temporal information in rules. This, however, has the side effect of making rules more complex. A more thorough description for including temporal constraints in rules is for example presented in the work of Walzer (Walzer, 2009; Walzer et al., 2007, 2008c,b,a), which incorporates a rule based approach using the Rete algorithm, for complex event recognition. In that work, the Rete algorithm is extended to allow for temporal constraints to be modelled. As noted, however, a potential problem with complex rules can be that it may be hard to easily understand and define rules containing causal or temporal relations in the premises. We shall cover rule based techniques in more detail in Chapter 3, since these are used for evaluation purposes in the thesis.
2. SUPPORT FOR SITUATION AWARENESS

2.4.5 State transition techniques

State transition techniques such as finite state automata, HMMs and variants thereof, are often used for various recognition tasks. Timed automata are for example used for diagnosis of discrete event systems (Bouyer et al., 2005; Supavatanakul et al., 2006). This is a task that involves recognition of typical system states. Moreover, Meyer-Delius et al. (2007) use HMMs for probabilistic recognition of situations in their work on intelligent driving assistants. In their approach, dynamic Bayesian networks26 are used to estimate a system state $x_t$ from observations $z_t$, at time $t$. On top of this, relational logic is used for capturing an abstract system description $o_t$ of $x_t$. Relations in the relational logic are defined using state variables in $x_t$, thereby allowing the construction of $o_t$ from $x_t$. Examples of relations extracted from $x_t$ include if two cars are close and if one car is behind another. In the approach, hidden Markov models are used for describing abstract situations consisting of sequences of relations (as defined in the relational logic). When new relations are extracted from observation, new candidate HMMs are instantiated and tracked as hypotheses over time. The system keeps track of a set of active situation hypotheses and uses Bayes factors for determining which of the currently active hypotheses that best explains the presently observed sequence of relations.

Situation recognition is however a task that may involve recognising partially temporally ordered patterns consisting of relations between multiple objects. Three main problems thus present themselves when using strictly sequential techniques for situation recognition. Firstly, it can be problematic to consistently model the state space of an activity containing multiple objects, when the number of objects and their relative order is a priori unknown. This problem may however be addressed through the use of hierarchically decomposed models, see for example the work of Liu and Chua (2006).

The second problem lies in the sequential nature of the approaches. Since the approaches are sequential, they do not have means of modelling concurrency and partial temporal order. Instead, all possible variations must be modelled, possibly resulting in very large state spaces. This problem can be addressed by using for example Petri nets, which is a state transition technique whose main strength lies in representing concurrency and temporal synchronisation (Sowa, 2000). Petri nets have for example been used in (Ghanem et al., 2004; Lavee et al., 2007; Perše et al., 2008), for recognising complex activities and events in video surveillance and analysis applications.

Coloured Petri nets (CPN), an extension to regular Petri nets, have also been used by for example Wagenhals et al. (1998); Wagenhals and Levis (2002); Wagenhals and Wentz (2003) for assessing situations and for comparing possible courses of actions, in work on effects-based operations. Their work integrates influence nets, a probabilistic modelling technique, with discrete event systems, to assess courses of action at operational and strategic levels in military war

\[26\text{For more information on dynamic Bayesian networks (DBNs), see Murphy (2002).}\]
games. Influence nets are used for modelling causal relationships between actions and effects. These are then used to construct CPNs for incorporating timing and sequence information. Nodes in the influence net correspond to transitions in the Petri net, and tokens are used for carrying probabilities.

Another interesting approach for modelling partial temporal order between components is the use of weighted ranked tree automata (WUTA) (Högberg and Kaati, 2010). Högberg and Kaati propose the use of WUTA for constructing automated systems for threat detection in a plan recognition setting. Högberg and Kaati argue that tree automata are suitable since they allow for patterns to be defined hierarchically. Goals and subgoals are defined top-down, to at the bottom be realised/coupled to actions/observations in the world. Tree automata also allows for partial temporal order of nodes, and partial inclusion of subgoals. Lastly, tree automata are founded on a well researched theory. These aspects should make them suitable for situation recognition too. In this thesis the focus is however on Petri nets, wherefore the use of tree automata is left for future work.

Thirdly, although HMMs, FSA and similar approaches are often depicted as changing state upon observing various symbols, the problem in situation recognition involves a scenario where the symbols are predicates of varying arity, and where the terms in predicates refers to variables for objects. This thus also involves solving a problem similar to unification.

We shall return to state transition based techniques, and Petri nets in particular, in Chapter 3, since this is the main topic addressed in this thesis.

### 2.4.6 Plan recognition

Although not a technique, rather a field of study, plan recognition shares many similarities with situation recognition. Plan recognition has been defined as the process of inferring future actions, intent or goal of an agent by observing its actions (Carberry, 2001; Fagan and Cunningham, 2003). To model the domain, a plan library consisting of goals and actions for reaching them are typically used. A goal is a state and actions describe changes to the state space. By applying many different actions (can be at different levels of abstraction), an agent tries to move from the current state to the goal state. In this view, the objective of plan recognition becomes to infer possible future goals and the intent of an agent by observing its actions and matching these with the knowledge in the plan library. Techniques used in the plan recognition literature include but is not limited to: BNs, classical abduction, inductive concept learning, machine learning, grammatical parsing methods and case based reasoning. For a review of work within plan recognition, see for example Carberry (2001). For information about tactical plan recognition in uncertain environments, see for example the work of Mulder and Voorbraak (2003).

There are common denominators between the plan and situation recognition problems. In both cases we are usually dealing with data that to some
extend already has been refined into symbols. Secondly, we try to classify the
current set of actions/relations, in order to find what we are looking for. How-
ever, we are not trying to infer intent for specific agents; rather we are looking
for predefined patterns of activity that could be interesting for operators to be-
come aware of. Furthermore, we do not have extensive knowledge about what
constitutes an enemy plan, what available actions there are, and what goals
they seek. Instead, a solution to the situation recognition problem needs to be
conceptualised based on what we are able to observe.

2.5 Chapter summary

In this chapter we have looked at background material related to situation
awareness, data fusion, situation fusion/assessment, and various views on sit-
uations and recognition. Needless to say, the world and our interactions with
it are often viewed as a revolving loop in which we must perceive and inter-
pret information in order to decide on suitable actions for achieving our goals,
e.g. the OODA loop. Data fusion provides technical support for interpreting
and aggregating data and information, and can be used for achieving improved
situation awareness. An important part of situation awareness is the ability of
matching present information with patterns of previously known situations.
Situation recognition serves the purpose of providing technical support for this
process, and can be considered of high importance for aiding decision makers
in recognising complex patterns when there is little time available.

Two main themes on situations have been covered. The aim in situation
calculus is to be able to reason in certain situations, and to be able to achieve
certain goals given where we are now. Situation calculus is also partly the basis
of modern AI planning techniques (Russel and Norvig, 1995), such as STRIPS\textsuperscript{27}
(Fikes and Nilsson, 1971; Nilsson, 1998) and GOAP\textsuperscript{28} (Orkin, 2003). In situ-
atution theory on the other hand, the aim is more philosophical and concern the
representation and meaning of language and information. Nonetheless, situation
theory has been promoted as a suitable basis for understanding situations
within the information fusion community.

Problems similar to situation recognition have previously been addressed
with a large variety of techniques that do not involve explicit reasoning or
philosophy. These include, but are not restricted to, graph matching, Bayesian
networks, Markov random fields, hidden Markov models, rule based forward
chaining, Petri nets and temporal constraint propagation.

In Chapter 3 we shall take a closer look at two of these approaches for
efficient recognition of complex patterns. Chapter 3 will also look briefly at
the use of genetic algorithms for possibly learning such complex patterns using
data.

\textsuperscript{27}STanford Research Institute Problem Solver.
\textsuperscript{28}Goal Oriented Action Planning.
Chapter 3
Techniques for situation recognition

This chapter presents background material related to two different kinds of techniques that should be able to address the situation recognition problem efficiently. We shall start with rule based systems, which have been used on a very large number of problems in the area of artificial intelligence (AI) over the past four decades. More specifically, the focus will be put on the Rete algorithm (originally suggested by Forgy (1982)) and extensions that allow for modelling of temporal constraints. Secondly, we focus on state transition based techniques, and more specifically, finite state automata and Petri nets. State transition techniques have also been used extensively for pattern matching. Also important in the notion of situation recognition however, is patterns of what to recognise. To end the chapter we thus look at genetic algorithms, which possibly can be used for defining interesting patterns using data.

3.1 Rule based recognition

Rule based systems have been used extensively over the past forty years within the area of AI, and more specifically in connection to expert systems. In fact, Girratano and Riley (1989) argue that rule based expert systems have been one of the most popular types of expert systems. Expert systems, or knowledge based systems, is a branch of AI that makes use of specialised knowledge, e.g. domain knowledge, to solve problems at a level comparable to human experts. Rules may however also be used for defining interesting patterns that can be used for recognising situations. Rules have according to Girratano and Riley a number of attractive properties that have made them popular:

- Modular nature. This allows for rule bases to be easily extended and it also enables for incremental development.
• **Explanation facilities.** By keeping track of the premises in all rules that have been activated, the chain of reasoning can be presented.

• **Similarity to the human cognitive process.** Rules have been shown to constitute a natural way of modelling how humans solve problems.

Rule based systems have their roots in the work of Newell et al. (1959) and Newell and Simon (1961), who presented the general problem solver as a generic approach for problem solving and as a theory how human problem solving. This work has strongly influenced the artificial intelligence community over the years, and it also serves as the intellectual precursor of modern expert systems (c.f. Luger (2002)). The basis of the general problem solver\(^1\) (Newell et al., 1959) was the use of means ends analysis. This however changed (Newell and Simon, 1972) to instead favour the use production systems as a basis (c.f. Luger (2002)). A production system\(^2\) consist of three essential components: two memories and an inference engine. The first memory is referred to as a production memory, and it consist of rules (condition action pairs, also called productions) such as:

\[
\text{if size > 10 then weight = large.}
\]

The part of a rule between the *if* and *then* is often referred to as the left hand side (LHS)\(^3\) and the part to the right of *then*, is often referred to as the right hand side (RHS)\(^4\) (Girratano and Riley, 1989). The second memory is called a working memory, and it contains facts such as:

\[
\text{size: 20; weight: small; color: orange.}
\]

Facts in the working memory, working memory elements (WMEs), can be matched with rules in the production memory, resulting in new facts being inserted, facts being removed or other actions being carried out. This inference process typically proceeds in a cyclic fashion called the recognise-act cycle (Luger, 2002; Girratano and Riley, 1989). The recognise-act cycle typically consist of three steps: rule matching, conflict resolution and act. Facts are in the recognise step matched with rules, and all matching rules are put in a list. In the conflict resolution step, it is decided which rule (if more than one was matched) that should be allowed to fire. Finally, in the act step, rule consequences are carried out. Naturally, it is the recognise step that is most related to the situation recognition problem. An important aspect of the production system approach is that there is a clear separation of knowledge and control (Luger, 2002). Figure 3.1 illustrates the components and the processing of a production system.

---

\(^1\)The general problem solver later evolved into Soar (Laird et al., 1987).

\(^2\)Production systems have their roots in the work of Post (1943), which used them in symbolic logic (Girratano and Riley, 1989).

\(^3\)Other common names include antecedent, conditional part, pattern part and premise.

\(^4\)Consequent, conclusion and action are other names for this.
3.1 RULE BASED RECOGNITION

3.1.1 Rule based inference and matching

As we have seen, the inference engine determines which rules that are activated. In a larger context, a query or problem is generally given to the system, and after a number of cycles have been carried out by the inference engine, a solution can perhaps be provided. A group of inferences that together connects a problem with its solution is termed a chain (Girratano and Riley, 1989), and the inference process is called chaining. Chaining can essentially be carried out in two different ways: forward chaining and backward chaining. Reasoning is in forward chaining carried out from facts to conclusions; from problem to solution. In backward chaining, reasoning is carried out in reverse, e.g. from hypotheses to facts. An alternative view of a backward chain is in terms of goals that can be achieved through subgoals.

The problem addressed with rule based matching consists of finding all rules for which facts exist that fulfil the premises. In cases where the matching process only has to occur once, then the solution is quite straightforward: iterate through all rules and facts and find all matches (Girratano and Riley, 1989). This is, however, seldom the case in rule based system, where the matching process instead is repeated in each iteration of the recognise-act cycle. Between iterations, new facts may be added, existing facts may be updated or old facts may be removed. Changes to the fact base may result in changes to the satisfaction of rules: some patterns that previously were satisfied may no longer be, and patterns that previously were unsatisfied, may have become satisfied. Matching is thus an ongoing process and the satisfaction of rules must be maintained for each iteration. For a more detailed description, see for example Girratano and Riley (1989).

A problem with the straightforward matching approach is that it may have a rather high time consumption. The number of rule unifications that has to be
carried out for solving a specific problem can be calculated as $wrnc$, where $w$ is the number of facts in working memory, $r$ is the number of rules, $n$ is the number of elements in the LHS of each rule, and where $c$ depicts the number of cycles required to solve the problem (Russel and Norvig, 1995). As an example, consider a situation where there are 100 facts in working memory, 50 rules in the rule base and where each rule has 2 elements. This would in each iteration result in 10000 unifications. Change one fact and 10000 new unifications are required. Add one fact and 10100 more unifications are required.

Girratano and Riley (1989) claim that most rule based systems have a property called temporal redundancy, that is, only a small percentage of the size of the working memory is added, updated or removed in each cycle. Moreover, having rules to guide the search for facts implies many unnecessary computations since most rules are likely to find the same facts in the present iteration, as was found in the previous iteration. This can however be avoided by remembering partial matches from iteration to iteration, and by only deriving changes in partial matches due to added, updated or removed facts. The rules remain static and the facts change. Girratano and Riley thus argue that the facts preferably should find the rules, and not the other way around.

### 3.1.2 The Rete algorithm

The Rete algorithm was introduced by Forgy (1982) as a means to address efficiency issues coupled to rule based matching. The Rete match algorithm is according to Forgy, an algorithm for computing the conflict set in a production system. The algorithm can be seen as a black box, where the input consist of changes to working memory, and where the output consist of changes to the conflict set (Forgy, 1982). The algorithm is designed to exploit the temporal redundancy often exhibited in rule based systems, and it does so by saving the set of all partial matches of the matching procedure, from cycle to cycle (c.f. Girratano and Riley (1989)). Instead of carrying out the complete matching procedure in a cycle, it only computes changes to the set of partial matches for each change that is made to the working memory. As an example, if a rule finds two out of three facts in the first cycle, then these two facts does not need to be found in the next consecutive iteration. It is only the third fact that needs to be found. To allow for this, the first two facts are stored in something termed a partial match, which is associated with the rule that was checked. A partial match is a set of facts that partially matches the patterns of a rule, i.e. some subpatterns\(^5\) are matched.

In order to properly model a complete space of partial matches, all combinations of sets of facts that partially matches a rule needs to be stored. A partial match that contains facts that satisfies all parts of the pattern in a rule

\(^5\)The premise part of a rule may consist of many conditions. Each of these conditions is in this thesis referred to as a subpattern.
is a complete match, and when such partial matches are found, rule activation occurs. Changes to the space of partial matches are only carried out when facts are added or removed\(^6\). When facts are removed, all partial matches containing those facts are also removed.

Girratano and Riley (1989) argue that one disadvantage of the Rete algorithm is that it can be very memory intensive. In general, however, they point out that this trade-off between speed and memory is worthwhile. Another issue is that poorly written rules can lead to considerable amounts of memory being consumed. It is thus of importance that rule designers pay attention to details. The Rete algorithm however also improves performance in rule based matching by exploiting structural similarity of rules. Several rules can consist of similar subpatterns. For example, consider the following two rules:

Rule1: if \( x = 10 \land y = 20 \) then class is tree  
Rule2: if \( x = 10 \land z = 10 \) then class is bush.

These rules share the same subpattern \( x = 10 \). Such structural similarities are exploited in the Rete algorithm by grouping rules that share parts of their patterns with each other.

The Rete algorithm is based on a directed acyclic graph of nodes, which is divided in two parts for taking care of two different problems (c.f. Forgy (1982); Girratano and Riley (1989)). The first problem is to match subpatterns and the second is to assert consistent variable bindings across patterns. For example, given the following pattern:

\[
\text{if } \text{approach}(x, y) \land \text{intercept}(x, z) \text{ then stressed}(x),
\]

then the first problem lies in finding the pattern:

\[
\text{approach}(\_\_\_\_ \land \text{intercept}(\_\_\_\_),
\]

and the second problem lies in assuring that \( x \) in the first subpattern is the same as \( x \) in the second subpattern. The first problem is taken care of in a pattern network and the second problem is taken care of in a join network.

The pattern network

The pattern, or alpha, network is used for determining which facts that matches subpatterns of rules (Girratano and Riley, 1989). The pattern network is structured in a tree, where nodes are connected in a linear fashion from the root node. Nodes represent tests between a field in a subpattern and a field in a fact. The leaves in the tree represent complete matches with subpatterns. The

\(^6\)Updating a fact is thus carried out by first issuing a fact removal command and then issuing a fact addition command (Forgy, 1982).
pattern network can also be seen as a discrimination network (Walzer, 2009). The pattern network is only concerned with intra element checking, that is, tests only involve single WMEs (Forgy, 1982). Variable bindings are thus not checked across subpatterns involving different WMEs and different subpatterns (Girratano and Riley, 1989).

The root node in a network is used as input to the black box, and it simply passes WMEs on to successor nodes. Nodes in the alpha network, except the root node, are also referred to as one input nodes or alpha nodes, and they only have one parent node, but may have multiple successor nodes. Alpha nodes contain expressions to be evaluated (Girratano and Riley, 1989) and are used to carry out simple conditional checks (Walzer, 2009). A special type of alpha node may also be used, type nodes. These have a single input, and are used to distinguish between different types of facts. They thus act as filters that only pass along facts that match the type of WME being processed in the subtrees that follows (Walzer, 2009). At the end of the pattern network a number of alpha memories are locate. More specifically, every leaf node is associated with an alpha memory. All patterns that have passed all the conditional tests on the path up to a leaf node are stored in its associated memory (Girratano and Riley, 1989). Alpha memories are sometimes also referred to as a right memories.

Structural similarity between different rules, and between different parts of rules, is exploited in the pattern network. Similar parts are shared by being associated with the same part of the tree, from the root up to the point where there are differences. At points of difference, the tree is split and multiple paths follow from the split node, one for each differing pattern, or rule.

The join network

As noted, the end of the pattern network consists of a number of alpha memories. These contain matches between all subpatterns of rules and facts in the working memory. The next step is to assert that variable bindings across subpatterns are consistent (Girratano and Riley, 1989). This is taken care of in the join, or beta, network. Forgy (1982) refers to this part of the networks as being concerned with checking inter element features and that it involves tests including more than one WME (variable consistency across facts are thus asserted).

The join network also consists of a number of nodes. However, these nodes have two inputs and are termed two input nodes, join nodes or beta nodes. Beta nodes are used to combine facts to produce joined subpatterns (Walzer, 2009). Each beta node represents a join operation between partial matches arriving at each of its two inputs, which are referred to as its left and right inputs, respectively. Successful join operations (WME combinations with valid variable bindings) results in new partial matches. These are stored in an output memory, or beta memory. All valid combinations are however also forwarded to child nodes following the beta node. Each of the inputs in a beta node is associated with a memory (alpha or beta). In the view of a beta node, these memories are
3.1. RULE BASED RECOGNITION

referred to as the right and left input memories, respectively. Typically, one of
the memories is associated with an alpha memory in the pattern network, and
the other is associated with a beta memory from previous join nodes.

A beta node is activated when a new partial match arrives at one of its
inputs\(^7\). When this happens, it checks for valid combinations of the new partial
match with all partial matches existing in the input memory associated with
its other input. Generally, both alpha- and beta memories contains matches
between parts of rules and the contents in working memory.

Leaf nodes in the join network are termed \textit{terminal nodes}, and they equal
the number of rules in the knowledge base (c.f. Walzer (2009)). Partial matches
arriving at terminal nodes, from the join network, represent complete matches
between rules and these thus results in new items being put on the agenda.
Structural similarity is also exploited in the join network by having join nodes
shared by different rules or patterns. For an extensive description see for in-

\textbf{Rete construction and example}

The construction of a matching network proceeds in two steps. When the com-
piler processes the LHS of a rule, it starts with building the pattern network
for intra element features, by determining the intra element features of each
rule and then by adding corresponding nodes to the network (Forgy, 1982).
The result is a linear sequence of nodes, where each node tests for the presence
of one feature. When the compiler is finished with intra element features, it
proceeds by building nodes that test inter element features (Forgy, 1982). Each
inter element feature is represented with a join node that joins two paths. One
path represents the linear sequence of inter element checks and the other path
represents previous joins. Finally, after all paths have been joined, the compiler
builds the terminal node to represent the rule (c.f. Forgy (1982)).

A short example is here provided for the construction of a Rete matching
network. The example is adapted from Walzer (2009). Picture a warehouse
with different cold storage rooms for keeping fruits. In each of the rooms, three
temperature sensors have been installed. These sensors can be used for making
sure that the temperature in a room is within limits. The sensors are however
of low quality and may be malfunctioning from time to time. To be sure that
the false alarm rate is kept low, an alarm should only be raised if three sensors
in a single room indicate that the temperature is too high. The sensors provide
objects of different types: sensor1, sensor2 and sensor3. Each object has a value
property which indicates the temperature reading on the sensor, and a room
property, which denotes the room in which the sensor is located. A rule for
raising an alarm based on these criteria can be written as in Listing 3.1.

\(^7\)A new partial match arriving at its left input is called a left activation, and a new partial match
arriving at its right input is called a right activation (Walzer, 2009).
3. TECHNIQUES FOR SITUATION RECOGNITION

Listing 3.1: Example rule (adapted from Walzer, 2009).

```java
rule "Raise temperature alarm"
    IF ( sensor1.value > 4 AND
         sensor2.value > 4 AND
         sensor3.value > 4 AND
         sensor1.room == sensor2.room AND
         sensor2.room == sensor3.room
    )
    THEN
        Raise alarm for room sensor3.room
    END
```

A Rete match network that implements this rule is illustrated in Figure 3.2. The pattern network consists of a single root node followed by three type nodes, one for each sensor type. Following the type nodes are three alpha nodes, each checking for the condition `sensorX.value > 4`. At the end of the pattern network there is an alpha memory for each subtree. The join network consists of two beta nodes, one beta memory and one terminal node.

![Rete match network diagram](image_url)

Figure 3.2: Illustration of an example Rete match network (adapted from Walzer, 2009).

### 3.1.3 Temporal extensions to the Rete algorithm

Organisations are today using many different systems for managing day to day business. Some systems are rule based and use the Rete algorithm, whilst others are based on complex event processing (CEP). This results in users having to resort to different tools for managing different tasks in their operations. Naturally, if the two paradigms could be combined, then time, money and learning time could be saved. Walzer (2009) thus investigates the combination of rule based systems using the Rete algorithm, and complex event processing systems. There are according to Walzer many similarities between the two paradigms, such as the use of logical operators for combining single facts or events to detect complex patterns. There are, however, also a number of differences, where the

---

8 There are of course systems based on other paradigms as well.
main difference is that rule based systems using the Rete algorithm are not designed for processing events per se (Walzer, 2009). Besides this core difference, Walzer analyses the properties of the two paradigms:

- **In a rule based system** facts are only inserted, removed and updated infrequently, e.g. there is a low update rate. Furthermore, there is not any predefined notion of time with respect to facts. They are assumed to be true when they are in the system. This can be compared with events, which occur at a point in time or in an interval of time. Facts are on the other hand valid until they are manually retracted. Facts are also put into the working memory in an unordered fashion. There is thus no information about the order in which facts were added or updated. This prevents temporal relations between facts to be modelled.

- **In a complex event processing system** there is an unbounded continuous stream of events that needs to be processed. Events are produced at very high update rates (100 events / s is one example given by Walzer). Events cannot be changed or retracted. They simply occur. Moreover, events are put into the system in an ordered fashion since the input stream consists of a sequence of ordered items that varies over time. This can be exploited to for example model temporal relationships between events. Lastly, events cannot be stored indefinitely due to the large number of them. There are simply too many. This in turn requires that the lifetime of events is properly modelled, e.g. for how long should they remain in the system.

In order to address these problems and to allow for the two paradigms to be merged, Walzer identifies three important extensions to the Rete algorithm: (1) introduction of a windowing concept, (2) introduction of explicit temporal operators and (3) introduction of garbage collection. These problems are addressed in Walzer et al. (2007, 2008c,b,a); Walzer (2009). The situation recognition problem, as addressed in this thesis, shares many similarities with both rule based systems and complex event processing systems. The extensions proposed by Walzer (2009) are thus of importance from a situation recognition perspective too. The most important for this thesis is considered to be the introduction of explicit temporal relations between events. This is thus the topic in the rest of this section.

**Modelling temporal relations**

Complex events are characterised as being composed of many subevents that have occurred at different points in time. For example, the fact that event A occurred 20 minutes before event B could indicate that machine failure C is about to occur. The need for modelling temporal relations can be essential when working with situation recognition too. A smuggler boat deployed from mother
ship before meeting a contact boat. When modelling complex events, or situations, it is thus of importance to be able to specify temporal relations between different subcomponents.

There are essentially two different approaches for modelling time: time point semantics and temporal interval semantics (Mörchen, 2007). In the former, a single point in time denotes the occurrence of an event or similar. In temporal intervals on the other hand, two points in time are used for denoting an interval, the start time and the end time. Time point semantics allows for three explicit temporal relations: before, after and equals. For more complex temporal relations, two or more basic relations need to be combined. Time point semantics can however easily be used for expressing quantitative temporal relations, such as \( A \) 20 minutes before \( B \). The other choice is to use some form of temporal interval semantics, such as Allen’s (1983) interval algebra. Temporal intervals can be used for expressing more complex temporal relations. They are however restricted to expressing qualitative relations, e.g. \( A \) contains \( B \). In case of building complex event processing systems, or as in the case of this thesis, situation recognition systems, then the aim is to recognise complex patterns. Recognised patterns can in turn be pushed back to the system. However, they take place over time and are thus preferably modelled using interval semantics and not time point semantics.

Allen (1983) introduced a temporal interval algebra that contains thirteen temporal operators for relating temporal intervals to each other. These relations are illustrated in Table 3.1, in which \( X \) and \( Y \) are intervals of time and where \( i \) represents the inverse of a temporal interval relation. A property of the temporal relations in Allen’s temporal interval algebra, is that they are exclusive, e.g. if \( X \) equals \( Y \), then \( X \) cannot be before \( Y \).

<table>
<thead>
<tr>
<th>Relation</th>
<th>Illustration</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X = Y )</td>
<td>.xxxx... .yyy...</td>
<td>( X ) equals ( Y )</td>
</tr>
<tr>
<td>( X &lt; Y )</td>
<td>xxx....</td>
<td>( X ) before ( Y )</td>
</tr>
<tr>
<td>( Y &gt; X )</td>
<td>.....yyy</td>
<td>( Y ) after ( X )</td>
</tr>
<tr>
<td>( X m Y )</td>
<td>.xxx....</td>
<td>( X ) meets ( Y )</td>
</tr>
<tr>
<td>( Y m i X )</td>
<td>......yyy</td>
<td>( Y ) met by ( X )</td>
</tr>
<tr>
<td>( X o Y )</td>
<td>.xxxx...</td>
<td>( X ) overlaps ( Y )</td>
</tr>
<tr>
<td>( Y o i X )</td>
<td>.yyyyy...</td>
<td>( Y ) overapped by ( X )</td>
</tr>
<tr>
<td>( X d Y )</td>
<td>.xxx...</td>
<td>( X ) during ( Y )</td>
</tr>
<tr>
<td>( Y d i X )</td>
<td>.yyyyy...</td>
<td>( Y ) contains ( X )</td>
</tr>
<tr>
<td>( X s Y )</td>
<td>.xxx....</td>
<td>( X ) starts ( Y )</td>
</tr>
<tr>
<td>( Y s i X )</td>
<td>.yyyy...</td>
<td>( Y ) started by ( X )</td>
</tr>
<tr>
<td>( X f Y )</td>
<td>....xxx</td>
<td>( X ) finishes ( Y )</td>
</tr>
<tr>
<td>( Y f i X )</td>
<td>.yyyyy...</td>
<td>( Y ) finished by ( X )</td>
</tr>
</tbody>
</table>

Table 3.1: Allen’s temporal relations (adapted from Allen, 1983).
Already, Allen’s temporal relations depicted in Table 3.1 can be expressed in rules in case the events, or corresponding elements, have associated start and end times. Listing 3.2 gives an example for expressing that $B$ occurs during $A$, e.g. $B \text{ d} A$.

**Listing 3.2:** Using explicit temporal intervals in a rule (adapted from Walzer, 2009).

```plaintext
IF ( A.startTimeStamp > B.startTimeStamp AND
     A.endTimeStamp < B.endTimeStamp )
THEN
  carry out action
END
```

Each of the thirteen relations can be implemented in a similar fashion. However, the start and end times of the two intervals need to be compared in different ways. Walzer (2009) argues that it soon becomes tedious to explicitly specify the various temporal relations, since they all require different combinations of tests. Furthermore, a Rete match network for modelling the rule described in Listing 3.2 would require two beta nodes and a beta memory. Instead, Walzer argues that it is preferable to explicitly use temporal relations and to have specialised relative temporal constraint (RTC) nodes in the Rete match network. Walzer thus proposes an extension to the Rete match algorithm, which consists of adding a new type of beta node, an RTC node. An RTC node implements the temporal relation operators of Allen’s (1983) thirteen temporal relations. In all other respects, RTC nodes operate as regular beta nodes.

The temporal relations of Allen do not allow for quantitative temporal relations to be modelled, and this can be a limitation in some systems. Walzer thus suggests extensions to the relational algebra, to also gain the benefits of using quantification. This is done by accepting parameters to temporal relations, such as $X < (10, 20) Y$, which would denote the before relation, but with the restriction that $X$ must have occurred 10 to 20 minutes before $Y$\(^9\). For details on the quantitative extensions, please refer to Walzer (2009). Lastly, Walzer discusses the use of specialised memories for RTC nodes, which uses various indexing schemes for improved efficiency.

Other extensions to the Rete algorithm for explicitly modelling temporal relations have also been suggested. Berstel (2002) suggests the use of time point semantics, Maloof and Kochut (1993) also suggest the use of temporal intervals and operators before, after and during, and Schmidt et al. (2008) propose the use of two separate networks, one for facts (without a notion of time) and one for events (with a notion of time). The work of Walzer (2009) is however considered to be the most suitable for the work carried out in this thesis.

\(^9\)A similar approach is described in Meiri (1996).
3.2 State transition based recognition

State transition techniques such as finite state automata (FSA) and hidden Markov models (HMMs) have become very popular for recognition of patterns in sequences of symbolic input. Typically, the perceived state of some process is traced as specific information is discovered. State transition techniques model how some process transits between states, for various types of emitted symbols. These techniques are also sometimes referred to as state transition diagrams. In this section we shall recapitulate on finite state automata and Petri nets. The former technique is very common when it comes to recognition tasks, whilst the latter is more commonly used for modelling complex concurrent systems. As we shall see however, Petri nets can also be suitable for recognition tasks.

3.2.1 Finite state automata

Finite state machines (FSMs) are according to Sowa (2000) the simplest and most widely used type of state transition diagrams. The building blocks in finite state machines are a finite set of states, a designated starting state, an input alphabet and a transition function that maps each pair of state and letter in the input alphabet to a next state (Rosen, 1999). Finite state machines are often used for modelling the behaviour of different kinds of systems. In the past they have often been used to model various parts of computers and computer software, and they have also been extensively used when for example implementing regular expression matching (see for instance Thompson (1968); Aho et al. (1986)). In recent years they have also gained a lot of interest in the computer games industry, where they are the most common structure for constructing artificial agents with seemingly intelligent behaviours (Buckland, 2004; Schwab, 2004).

There are according to Rosen (1999) basically two different kinds of FSMs. These are separated with respect to if they provide output or not. In finite state machines with output, a transition from one state to the next can be complemented with a symbol being emitted from the machine. These machines are also known as Mealy machines\textsuperscript{10}. Another kind of finite state machine with output is the Moore machine\textsuperscript{11}, in which output only depends on the current state. Finite state machines without output are more commonly known as finite state automata (automaton in singular). In finite state automata, specific states are designated as final states, and when reaching such states, the input has been recognised. In the rest of this section we shall focus on finite state automata since recognition, and not output, is the main focus of this thesis.

Rosen formally defines a finite state automaton as

\[
M = (S, I, f, s_0, F),
\]

\textsuperscript{10}Named after George H. Mealy.

\textsuperscript{11}Named after Edward F. Moore.
where $S$ is a finite set of states, $I$ is a finite input alphabet, $f$ is a transition function that assigns a next state for every input state pair, $s_0$ is an initial state, and $F$ is a subset of $S$ with final states. These kinds of automata are also referred to as deterministic automata since they map each input state pair to exactly one consecutive state. To capture behaviour that differs slightly, it can be beneficial to be able to accept varieties in the state transition paths. The function $f$ would in this case point to a set of possible consecutive states. These kinds of machines are more formally known as non-deterministic finite state automata. For a comprehensive overview see Rosen (1999).

Rosen (1999) describes two ways of illustrating finite state automata: state tables and state diagrams. Figure 3.3 shows both a state table and a state diagram for a deterministic finite state automaton, which has the purpose of recognising the string “010” in the input, using an alphabet consisting of 0 and 1. The initial state is shown by an arrow from nowhere, and final states are highlighted with double circles.

Finite state automata are however not directly suitable for recognising situations, since these are characterised by many different objects, over which many different relations can be true simultaneously, and in which both objects and relations change over time. In other words, the input at our disposal does not represent a sequentially evolving process. Furthermore, our view of the world is seldom perfect and we often miss pieces or experience uncertainty.

Related to FSA are hidden Markov models. These are statistical models for capturing the behaviour of various systems. They can thus be used for recognition when there is uncertainty\(^\text{12}\). Similar to FSA however, HMMs are designed for sequential input and do not directly offer any means for detecting parallel input streams that can be coordinated at select points.

\(^\text{12}\)They have previously been used extensively in pattern recognition/matching tasks, such as for example speech recognition. In the past couple of decades they have also gained much popularity within the area of bioinformatics, to for example perform “gene finding” in DNA sequences, multiple sequence alignment and protein structure prediction (De Fonzo et al., 2007).
3.2.2 Petri nets

Petri nets\textsuperscript{13} are according to Murata (1989), a graphical and mathematical tool suitable for modelling and studying systems that are characterised as being concurrent, distributed, asynchronous, non deterministic or stochastic. According to Sowa (2000) Petri nets are useful when representing causes and effects and when simulating processes and causal dependencies. Furthermore, Sowa argues that Petri nets are a generalisation of state transition diagrams, and that their major strength is their ability of representing parallel and concurrent processes.

Murata describes Petri nets as directed weighted bipartite graphs in which nodes are either places or transitions. Sowa claims that places correspond to states in finite state machines and transitions correspond to events in flow charts. In a finite state automaton, a single token is used to represent the current state of the process being modelled. In Petri nets however, different places can contain multiple tokens, simultaneously, representing any number of substates or processes. The global\textsuperscript{14} state of a Petri net is called its marking, and it consists of all tokens in all places in the net (Murata, 1989).

As in any bipartite graph, there are two disjoint sets of edges. In Petri nets these are edges from places to transitions, also referred to as input edges, and edges from transitions to places, also referred to as output edges. A Petri net can according to Murata formally be defined as a 5-tuple

$$\text{PN} = (P, T, F, W, M_0),$$

where $P = \{p_1, \ldots, p_m\}$ is a finite set of places, $T = \{t_1, \ldots, t_n\}$ is a finite set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs, $W : F \rightarrow \{1, 2, \ldots\}$ is a weight function and $M_0 : P \rightarrow \{0, 1, 2, \ldots\}$ is an initial marking of the Petri net such that $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

Transitions change the state of a Petri net by consuming tokens from their input places and by producing tokens at their output places. Moreover, a transition is activated in case the correct number of tokens exists in each of its input places. When a transition is activated, the number of tokens represented on each input arc is removed from each corresponding input place and the number of tokens represented on each output arc is inserted to each output place. Places can thus be seen as conditions for transitions, and for a transition to fire (be activated), each input place must contain the number of tokens that are required. In graphical notation, places are represented by circles, transitions are represented by vertical bars and tokens are represented as dots inside places. Figure 3.4 illustrates a Petri net for a chemical reaction resulting in water.

Places and transitions may have many different interpretations in different contexts. We have in this section however only been concerned with an inter-

\textsuperscript{13}Petri nets were invented in 1939 by Carl Adam Petri.

\textsuperscript{14}It should be noted that Murata (1989) do not refer to a global state, however, in case we choose to see every token as a substate, or local state, of some process, then it only seems suitable to refer to the composite of all local states as a global state.
3.2. STATE TRANSITION BASED RECOGNITION

Figure 3.4: Example of a Petri net describing the chemical reaction $2H_2 + O_2 \rightarrow 2H_2O$ (adapted from Murata, 1989). The dots in the places represent tokens that in this case refers to the availability of a chemical compound.

...creation that is relevant for this thesis. For more details concerning Petri nets and for different interpretations of places and transitions, the reader is referred to Murata (1989), who offers an excellent review and an introduction to the theory behind Petri nets.

Coloured Petri nets

Tokens are in classical Petri nets used for representing some substate of what is modelled. Tokens are represented with numbers, one for each place, and the number associated with a place denotes the number of tokens in the place. In coloured Petri nets (Jensen, 1991) tokens are also used as carriers of information. They may contain any sort of additional information that can be used for building a more abstract and compact system model. A coloured Petri net is a form of high level Petri net\textsuperscript{15}. The information carried by a token in a coloured Petri net may be arbitrarily complex, such as an integer, a list of tuples, a text string, or any other complex data structure (Jensen, 1991). The data attached to a token is referred to as the token colour.

A coloured Petri net consist of three parts: net structure, declarations and inscriptions. Similar to regular Petri nets, the net structure consist of places, transitions and arcs that are connected to each other to form a directed graph. Declarations are used to define variables and colour sets, where each colour set contains a number of colours. Jensen argues that the use of colour sets is analogous to the use of types in programming. Similar to data types, colour sets does not only define the actual colours in the set, but they also define operations and functions that can be applied to colours. Jensen also argues that colour set definitions also often implicitly introduce additional operators and functions, such as addition, subtraction and multiplication if numbers are used.

\textsuperscript{15}Another form of high level Petri net is called predicate transition nets.
Inscriptions are used for defining and restricting the behaviour of a coloured Petri net. For example, each place has an associated colour set, meaning that it can only contain tokens having a colour belonging to the associated colour set. Another example is the use of guards on transitions. Guards are Boolean expressions that must evaluate to true for the associated transition to fire. In addition, arc expressions and initialisation expressions are also used for defining the behaviour of a coloured Petri net. Lastly, an extension to coloured Petri nets is called hierarchical coloured Petri nets (Jensen, 1991). These allow for multiple different subnets to be related to each other in a formal way.

**Petri nets for recognition**

Petri nets are not usually thought of as mechanisms for recognition. Castel et al. (1996) however argue that they are quite suited for this task since they allow for sequencing, parallelism and synchronisation to be easily represented and visualised. Furthermore, they are also a generalisation of finite state automata, which apparently is used quite extensively for recognition. Hence, Petri nets should be suited for this task too. Lavee et al. (2007) identify two main approaches for using Petri nets to recognise complex patterns. The first approach is called *object Petri nets*, and in these, tokens correspond to detected objects and places represent particular states for objects. The second approach is called *plan Petri nets*, and in these, each place corresponds to a subevent, and a token in a place denotes the occurrence of that particular event. These two types of Petri nets for recognition are now presented in more detail.

*Object Petri nets* represent the approaches used by Ghanem et al. (2004); Ghanem (2007) and Borzin et al. (2007). As stated, a token in an object Petri net represents an object and places represents states of objects (Lavee et al., 2007). Object Petri nets can thus be seen as a type of coloured Petri net. A special type of transition is used, which is enabled when certain conditions with respect to properties of objects are met. Moreover, an interesting pattern can be seen as a transition in itself, thereby allowing for hierarchical deconstruction of patterns (c.f. Lavee et al. (2007)). *Plan Petri nets* represent the work presented by Castel et al. (1996). As noted, places correspond to subevents, and the occurrence of a subevent is represented with a token in a place. Transitions in plan Petri nets are enabled when certain scene states are true, and moreover, a token in a place represents the existence of a scene state. A limitation of plan Petri nets, is that places only have a capacity for one token.

A particularly appealing aspect of object Petri nets is that multiple instances of a complex pattern can be represented in a single Petri net. This is not the case for plan Petri nets, where each instance of a complex pattern is represented by an instance of a Petri net. Lavee et al. (2007) argue that object Petri nets are more robust than plan Petri nets. However, due to complexity of facilitating

---

16Names of the various components are also inscriptions, but they have no formal meaning and can thus be omitted (Jensen, 1991).
3.2. STATE TRANSITION BASED RECOGNITION

multiple patterns in a single network in object Petri nets, plan Petri nets may be advantageous in settings where humans are to be a part of the construction phase. Lavee et al. present a methodology for building Petri nets from semantic descriptions, for both types of Petri nets for recognition. Moreover, various constructs of Petri nets for capturing various concepts are illustrated in two video surveillance scenarios for both types of Petri nets. In this view it may indeed be important for understandability reasons, to keep Petri nets describing complex concepts as simple as possible. From a recognition perspective, object Petri nets can be argued to be superior due to their more compact representation.

Object Petri nets for recognition

In the work presented by Ghanem et al. (2004); Ghanem (2007), high level Petri nets are used for recognising complex events in video data. A number of primitive events are depicted as input. These are extracted by analysing video streams. The primitive events are then fed to Petri nets that represent interesting complex activities. This proceeds by having specific transitions associated with each primitive event. Ghanem et al. extend the work of Castel et al. (1996), by representing instances of complex events using tokens, compared to using a single Petri net for each instance. Ghanem et al. claim that this keeps the number Petri nets fixed and small.

In the approach, Ghanem et al. introduce the concept of conditional transitions. These are transitions that have additional constraints in the form of conditions put on input tokens. Thus, in addition to asserting the correct number of tokens in each input place, a transition also tests for conditions with respect to variables in tokens. Ghanem et al. also argue for the use of hierarchical transitions. Hierarchical Petri nets are discussed by Jensen (1991), who claims that they can be used for decomposing Petri nets into different parts. One way is to exchange a transition with a complete Petri net, or the other way around, abstracting some part of a Petri net with a single transition in a Petri net at a higher level. Hierarchical transitions in object Petri nets are however only allowed to have one immediate input and one immediate output (c.f. Ghanem et al. (2004)). Lastly, Ghanem et al. makes use of coloured tokens to represent different actors and information about these actors. All together, the object Petri net approach proposed by Ghanem et al. is a specialised hierarchical coloured Petri net, as presented by Jensen, for recognising complex events.

Conditional transitions are in object Petri nets coupled to primitive events that are externally invoked. Furthermore, each primitive event has a number of actors. These are put in the resulting token of the conditional transition. As the matching of a complex event progresses, new actors are added to the tokens that are consumed. Moreover, primitive events can be coupled to multiple conditional transitions. The occurrence of a primitive event in this case, results

\[17\] The guard concept from coloured Petri nets is thus used.
in all associated transitions being activated (in case their input edge restrictions are also fulfilled). The result is that all permutations of actor combinations are put into the net. To exemplify, consider the following pattern: car A enters a parking lot, car B enters the same parking lot, person C exits car A, and then, person C enters car B. Figure 3.5 illustrates.

Figure 3.5: Illustration of an object Petri net for recognising a parking lot situation (adapted from Ghanem et al., 2004).

In case a primitive event concerning a car entering the parking lot is detected, then two tokens would be created, one in place 1 and one in place 3. Both however having bound the car as a colour in the token. This is illustrated in the figure with one token in each of the initial places: \( t_0 \) and \( t_1 \). Next, consider a primitive event where a person leaves the car. The Petri net is now updated by consuming the token in place 1, \( t_0 \), and by producing a token in place 2. The resulting token in place 2 also has the person added to its list of actors.

The updating procedure described by Ghanem et al. (2004) proceeds as follows. For each complex event that is modelled, i.e. for each Petri net, there is a list of enabled transitions. These are transitions for which the input edge constraints are fulfilled, but for which a primitive event has not yet been detected. The transition is thus in a state of waiting, where the only missing information is a primitive event. Consequently, the marking of the Petri net cannot change unless primitive events are detected. Therefore, the Petri net is only re-evaluated once new primitive events are recognised. When a new primitive event is detected, the list of enabled transitions is checked to test if there is a transition waiting for the occurred event. If this is the case, then actor combinations are evaluated to test for consistency. In case valid combinations are found, then the corresponding tokens in input places are removed and new tokens are produced at output places. Finally, the list of active transitions is updated. Ghanem et al. (2004) argue that the use of an activation list constitutes an efficient implementation of the recognition procedure.

Perše et al. (2008) build upon the work of Ghanem et al. (2004) and Lavee et al. (2007) for recognising play tactics in basketball games. A methodology for automatically constructing Petri nets from activity templates is suggested.
Moreover, Perše et al. extend the formalism with capabilities for determining the degree of similarity between observed data and evaluated play tactics. This is done by adding another colour to tokens, which describes penalties with respect to detected subevents.

**Representing logical and temporal relations**

An essential aspect when constructing patterns of interesting behaviour is the possibility of representing various relations between events, complex events (or situations) and facts. Logical relations such as $x \land y$ and $x \lor y$ are important building blocks. Ghanem et al. (2004); Ghanem (2007) provide Petri net fragments for constructing three logical operations. These are illustrated in Figure 3.6.

![Petri net fragments for logical operations](image)

**Figure 3.6:** Illustration of the logical relations $x \land y$, $x \lor y$ and $\neg x$ using Petri nets (adapted from Ghanem et al., 2004). Transitions with thick bars represent conditional transitions whilst transitions with thin bars represent regular transitions.

The first fragment is $x \land y$, which is represented by a regular transition whose input consists of the output from each of the operands. The second fragment is $x \lor y$, which is represented by a transition whose single input is the output from each of the operands. The third fragment is $\neg x$, which is represented with a transition whose input is the output from the operand however, the edge is an inhibitor which instead can be activated when the operand is not true, i.e. the transition can be activated when there are no tokens in the input place.

Temporal relations can also be represented using Petri nets. For the rather simple relations before and after, transitions are simply put in sequence. Besides before and after, it is however also possible to represent all of Allen’s (1983) temporal interval relations (depicted in Table 3.1) using Petri nets. Both Ghanem et al. (2004) and Lavee et al. (2007) illustrate Petri net fragments for each of the temporal relations presented in the temporal interval algebra (Allen, 1983). These are however not presented here.
3.3 Genetic algorithms

As discussed in Chapter 1, an important aspect of a solution to the situation recognition problem is to allow for manually constructed situation templates to be adapted. This thesis investigates the viability of Petri net based situation recognition, and this means that it is also important to be able to learn Petri net situation templates, or to adapt existing templates using available data. Learning and adaptation can also be cast in terms of an optimisation problem, where a space of possible Petri nets is searched for some Petri net that maximises some objective function. Genetic algorithms (GAs) have in past work been used for successfully evolving various types of Petri nets for tasks such as epistasis modelling, process model mining, metabolic pathway prediction, project scheduling and wafer fabrication. For details, see Mayo and Beretta (2010); Alves De Medeiros and Weijters (2004); Nummela and Julstrom (2005); Prashant Reddy et al. (2001); Chen et al. (2001). Moreover, Chen et al. (2001) argue that genetic algorithms are able to randomly, but efficiently, sample large search spaces in polynomial time without the same memory requirements as more traditional search algorithms. GAs should thus constitute a suitable mechanism for learning Petri nets for recognition too.

Genetic algorithms (Holland, 1975) is a search/optimisation method that is inspired by the Darwinian principles of evolution and survival of the fittest. In nature, all living organisms share a common ancestor: the first single-celled organism. Throughout evolution, this one single ancestor has evolved into many species through the application of genetic operators such as mutation and genetic crossover. These alter DNA from generation to generation. More specifically, genetic algorithms make use of the concept of a population of competing individuals, where the best fit individuals in each generation has a higher chance of survival. Luger (2002) argues that learning in genetic algorithms is viewed as “as a competition among a population of evolving candidate problem solutions”. In machine learning we are interested in finding a good hypothesis $h$ from a hypothesis space $H$ of candidate solutions (Mitchell, 1997). In this view, genetic algorithms simply serve as a methodology inspired by Mother Nature, for performing that search. A key advantage of GAs is however their ability of searching at many places of the hypothesis space simultaneously (Luger, 2002).

In GAs a population of candidate solutions is formed, where each individual is encoded as a set of genes that together form a genome (or chromosome). In the traditional view, genes are encoded as single bits, and a genome is thus a bit string which can be translated into a solution. More complex gene alphabets such as natural numbers can of course also be used. The population of candidate solutions is evolved over a number of generations. In each generation, each individual is evaluated on the problem solving task in order to establish its fitness on the task, i.e. how good is the solution at the task. Fitness values are generally calculated and normalised to be in the range $[0..1]$. The fitness values are in turn used as basis in a selection and reproduction phase, where
individuals in the present population are selected to form the basis of the next
c consecutive generation of candidate solutions. This procedure is generally re-
peated until a population of individuals have been evolved that performs fairly
well at the learning problem. Algorithm 3.1 shows a general form of a genetic
algorithm, adapted from Luger (2002) and Mitchell (1997).

Algorithm 3.1 General genetic algorithm for a population of \( N \) individuals and
with a mutation rate of \( m \).

1:  \textbf{procedure} Genetic Algorithm(\( N \), \( m \))
2:     initialise a population \( P \) with \( N \) individuals
3:     \textbf{while} termination condition is not met \textbf{do}
4:         evaluate fitness of each member of the population \( P \)
5:     \textbf{end while}
6:     \textbf{while} number of individuals in \( P_S \) < \( N \) \textbf{do}
7:         select a pair of members from the population \( P \) based on fitness
8:         create two offspring using crossover on the selected pair
9:         add offspring to \( P_S \) (assert that \( |P_S| \leq N \))
10: \textbf{end while}
11: choose \( m \) percent of \( P_S \) and for each invert a randomly selected gene
12: \( P \leftarrow P_S \)
13: \textbf{end while}
14: \textbf{end procedure}

3.3.1 Fitness and reproduction

The evolutionary step of a genetic algorithm consists of generating a new pop-
ulation of individuals based on the individuals in the present population. This is
usually carried out in an iterative selection, combination and mutation scheme,
until the new population meets a target size \( N \). In the selection step, two indi-
viduals are typically selected from the present population to be used together
with a recombination operator, which results in one or more offspring to be put
in the new population of individuals. Mitchell (1997) describes three popular
selection mechanisms:

- \textit{Roulette wheel selection}. An individual is probabilistically selected based
  on its contribution to the summed fitness of the whole population. Each
  individual \( j \) thus has a \( f(j)/\sum_{i=0}^{N} f(i) \) probability of being selected, where
  \( f(k) \) denotes the fitness of individual \( k \).

- \textit{Tournament selection}. Two individuals are selected randomly from the pop-
 ulation. Out of these two, there is a \( p \) chance of selecting the better fit indi-
 vidual, and a \( 1 - p \) chance of selecting the less fit individual, given some
  pre defined probability \( p \). Larger tournaments can also be kept. Mitchell
  (1997) argues that tournament selection usually provides more diversion to
  the population, compared with roulette wheel selection.
• **Rank selection.** Individuals in the population are first sorted by their fitness, which results in a rank for each individual. Using this information, an individual is selected probabilistically based on rank, i.e. after sorting each individual \( j \) has a \( j / \sum_{i=0}^{N} i \) chance of being selected.

After having selected two individuals from the present population, these are typically combined with each other using a genetic crossover operator, to result in two offspring. For a new individual, some genes are inherited from the first parent, whilst the remaining genes are inherited from the other parent. The scheme for determining which genes to copy from which parent, and to which offspring, is called a crossover mask (Mitchell, 1997). The crossover mask is a string of bits equalling the size of a genome. Locations in the crossover mask containing a 1 indicates that the corresponding genes are copied from the first parent to the first offspring and from the second parent to the second offspring. Locations containing a 0 indicate that the corresponding genes are copied from the first parent to the second offspring and from the second parent to the first offspring. Using this construct, any arbitrary crossover mask can be used. Three often used schemes are presented by Mitchell (1997):

• **Single point crossover.** A single split point is selected at position \( n \) in the mask. Bits up to \( n \) contain 1’s and bits after \( n \) contains 0’s. The result is offspring where the first \( n \) genes are contributed from one parent and where the remaining genes are contributed from the other parent.

• **Two point crossover.** Two split points are used and the mask contains 0’s up to the first point, 1’s between the two points and 0’s after the second point. This results in offspring that have the same genes as one of the parents, but with a segment in the middle contributed by the other parent.

• **Uniform crossover.** Bits in the mask are sampled from a uniform distribution. Naturally, the result is offspring with genes sampled uniformly from the two parents.

Another custom genetic operator is genetic mutation. Genetic mutation may be applied to change the values of some of the genes for selected individuals of a new population, and it can be carried out in a number of ways. For example, each individual can have a probability of \( m \) having a randomly selected gene mutated. Alternatively, each gene in each individual may have a probability of \( m \) of being mutated. Holland (1992) also discusses the use of mutation as two genes being exchanged.

Lastly, another interesting selection mechanism is called **elitism selection.** In elitism selection, the best performing individuals of the present population are retained as they are, without modification. The purpose of this is to assure that good solutions are preserved during the evolutionary process. Elitism selection is thus not typically used for selecting individuals that are to be combined with
each other, but rather, it resembles cloning. Besides the genetic operators of crossover and mutation Holland (1992) discusses other genetic operators such as: inversion, dominance, modification, translocation and deletion. For more information, the interested reader is encouraged to read Holland (1992).

### 3.3.2 Properties of genetic algorithms

One advantage of genetic algorithms is that they are less prone to get stuck in local optima in the hypothesis space. The reason for this is that the hypothesis space can be searched at multiple places in parallel. Mitchell (1997) discusses a few other attractive features of genetic algorithms: (1) evolution is known to be a robust and successful mechanism in nature, (2) they can be parallelised and thus take advantage of more recent processing architecture and (3) they can be used to search complex hypothesis spaces consisting many interactions that are otherwise hard to model and to assess.

A disadvantage of GAs however, is that they generally require extensive amounts of computational time during evolution. This problem gets even more severe in cases where the evaluation step consist of assessing the performance of solutions that address problems that are of temporal nature, e.g. if the task is to evolve agents that try to learn how to behave in some environment, then each individual in each generation possibly needs to be allowed to interact with the environment for some amount of time. Furthermore, it can on occasion be problematic to design proper fitness functions, since it can be hard to determine exactly what will be promoted.

### 3.4 Chapter summary

In this chapter we have looked at two different kinds of techniques for recognising patterns of temporal nature. The Rete algorithm is typically used when constructing rule based systems. Moreover, Walzer (2009) introduces extensions to the Rete algorithm which allow for the explicit use of Allen’s (1983) temporal interval relations in rules. Through the use of these extensions a rule based paradigm can be used for recognising situations. Petri nets are a generalisation of FSA and their main strength is their capability of representing concurrency. This can be of importance, since many interesting situations can play out over time and consist of activities of multiple objects occurring in parallel. Two approaches for using Petri nets for recognition of complex patterns have been discussed, of which object Petri nets seems to be more suited for the task of recognising situations. Chapter 5 presents an extended Petri net based approach that addresses some limitation of object Petri nets at the task of recognising situations. This extended Petri net based technique will in Chapter 10 be compared with a rule based technique based on the Rete algorithm with the temporal extensions suggested by Walzer (2009).
The chapter has also briefly looked at genetic algorithms, a search/optimisation/learning method that possibly can be used for constructing Petri net situation templates. This is important since it is of utmost important to have good definitions of what to recognise. In genetic algorithms, a space of Petri nets is searched by evolving a population of candidate Petri nets over a number of generations. In each generation, all individuals are evaluated on the task and the best individuals are combined with a set of genetic operators to hopefully achieve offspring that are of better quality.

There are of course many other learning schemes: artificial neural networks (ANNs), decision trees, Bayesian learning, genetic programming, instance based learning and inductive logic programming, to name a few. All approaches have their respective advantages and disadvantages. It all boils down to selecting some scheme that fits the properties of the learning problem at hand. Genetic algorithms should however be suited for the task of learning Petri nets, since they can be used to search in complex hypothesis spaces that can be difficult to model and for which guidance can not explicitly be formulated. The use of genetic algorithms for learning Petri nets is discussed in Chapter 6 and empirically investigated in Chapter 11.

This chapter concludes the background part of this thesis. In the next part, theoretical results are presented.
Part II

Theoretical results
Chapter 4
Defining situation recognition

This chapter addresses the first research objective in the thesis, and it is mainly based on one publication: Dahlbom et al. (2009c). The main contributions of the chapter are:

- A conceptualisation of the situation recognition problem.
- A definition of the situation recognition problem.
- An analysis of requirements that can be put on solutions to the situation recognition problem.
- An extended perspective on situation recognition.

4.1 Towards a conceptualisation of the problem

A situation assessment is within the fusion community often seen as a freeze frame picture, or state, of the system at a given point in time (McMichael et al., 2004; Jakobson et al., 2007). Many interesting real-world situations can however not be identified from a single picture of the world. They evolve over time and need to be identified by distinct events, or similar concepts, that can occur in series or in parallel. Moreover, parts may also be synchronised at various points in time. Components in a pattern for an interesting situation type can thus be partially temporally ordered.

For example, a smuggling situation may consist of temporally synchronised events such as a smuggling boat being deployed from a mother ship, after which two boats meet of the coast, after which a boat returns to shore and another boat returns to its mother ship. Moreover, subsituations may occur in parallel. For example, the mother ship deploys a second boat which also heads to shore, but which moves in a somewhat unnatural way. The second boat thus tries to draw the attention from any possible surveillance analysts. The three depicted subsituations may occur simultaneously: one boat heading for shore to a rendezvous, a second boat heading away from the coast to a rendezvous and a
third boat heading to shore and acting anomalously. Hence, the interest is not only in finding situations that can be detected in a single picture\(^1\) of the world, but also, in being able to recognise situations that can be of partial temporal nature.

McMichael et al. (2004) argue that a freeze frame picture does not model past states, it does not model the process that has generated the current state and it does not allow for extrapolation into the future. It can thus not be considered to be enough for inferring what is going on in the world. A freeze frame picture does however model our interpretation of the current state of some abstract process that we are observing. Nonetheless, in order to recognise behaviour that stretches over time we possibly also need to model the past explicitly. It is important here to acknowledge the fact that it is our interpretation of some occurrence in the real world that is being observed, i.e. it is not necessarily some universal truth that we are observing.

Although many more or less precise notions of situations have been promoted, there is no formal definition of the situation recognition problem that can be used for describing situations that are of partial temporal nature. For example, the definition of situation recognition provided by Steinberg (2009), Equation 2.1, depicts the type of an interesting situation as consisting of a number of entities for which specific relations have been inferred. It does, however, not allow for partial temporal constraints to be put amongst these relations. Moreover, in order to construct feasible solutions to the situation recognition problem, i.e. solutions that can be put into use in real world systems, then it is also important to understand the complexity of the problem as well as understanding requirements that can be put on solutions to it. In order to understand the complexity and to derive a set of requirements, it is a necessity to properly understand the problem. This chapter asks two important questions for understanding the problem:

1. What are situations instances with respect to that which we are able to observe, and how can they be represented?

2. How can we represent interesting situation types with respect to what we are able to observe?

A conceptualisation of the problem is needed to investigate these questions.

### 4.2 An abstract view of the world

We continuously aim at estimating the state of some part of the real-world with our sensors. This typically boils down to estimating a set of properties of distinct objects exerting some behaviour in the environment. The state of the world we are trying to observe is however highly dynamic, or to quote\(^1\) a pictorial representation.
the Greek philosopher Heraclitus (535 - 474 B.C.), “the only thing constant is change”\(^2\). There is a plethora of continuous processes that constantly change the state of the world. Natural processes are governed by naturally occurring phenomena, and man made processes are results of humans interacting with the world. Both are however different types of continuous processes, being in different states at different points in time, and which we can have an interest in observing. As we will see in the following, situations evolving over time can be viewed as processes.

Processes can be thought of on many levels of abstraction. Ultimately they are, however, realised through physical and chemical processes affecting the real-world. As an example, picture the process of a car moving on a road. From one viewpoint the process continuously affects the state of the car, i.e. its position. This process, however, is an abstraction as there is an underlying process of fuel and oxygen combusting as a result of a driver pushing the accelerator, thereby turning the wheels, resulting in propulsion. At an ever lower level, carbon type molecules (the fuel) are forced to bond with oxygen molecules as a result of an increase in pressure from a raising piston, again resulting in even higher pressure, thus pushing the piston down, which results in the crank shaft revolving, in the end spinning the fly wheel. Yet, it is infeasible, if even possible, to always model and observe the world in terms of processes at the finest level of granularity, which we possibly don’t even know yet, and instead we need to use abstractions that must be chosen carefully with respect to the purpose of the system at hand.

As an outside observer we are typically interested in estimating the state of some of these continuous processes, at a suitable level of abstraction, with the help of discrete processes in machines. Discrete processes can according to Sowa (2000) be described in terms of time and change. Change occurs in discrete events that are separated with periods of inactivity termed states. An event thus imposes change to the state of a discrete process, and the evolution of a process can be tracked as the change in state over time.

From the viewpoint of a man made process, we are interested in controlling an abstract process to achieve some goal. The goal can be seen as a state that we aim at reaching by constructing a plan whose execution will bring us from the present state to the goal state. More formally, a plan is in the planning literature considered to be a sequence of actions (often expressed in terms of effects) affecting the state to bring it from an initial state to a goal state, through a series of partial states (Luger, 2002).

As an outside observer we often lack the ability of knowing the action and state space used in planning. Moreover, we do usually not have the capability of observing directly in the state space in which a planned man-made process is being carried out. Instead we must resort to our own interpretations of the world,

\(^2\)This is more correctly translated as: “You cannot step twice into the same river, for other waters and yet others go ever flowing on.” Harris (2003)
founded in terms we are able to infer from data, information and knowledge, when trying to estimate the state of the process being carried out by some external entity. It is changes in the state space we are able to observe, or derivatives thereof, that are termed events. Hence, actions and events are closely related as they denote change in the state of some process. However, the abstract state space which they operate in may be different.

To clarify the discussion, Figure 4.1 illustrates an example where a planned man-made process is being carried out through a series of actions in a state space. This affects the continuous real-world which we observe in a state space different from that in which the plan is being carried out.

![Figure 4.1: Illustration of a planned man-made process being carried out in a state space, affecting the continuous real-world, and then being observed in another state space. In the figure, \(g_i\) are partial states in a plan carried out through actions \(a_i\) to achieve a goal \(G\), \(z_i\) are states inferred from observations \(o_i\), and \(e_i\) denotes changes in the observed state space. To illustrate a natural process being observed, the man-made layer can simply be removed.](image)

To summarise, continuous processes can be estimated with discrete processes in machines, which can be described in terms of states and events at various levels of abstraction. An event denotes change in state and a series of states is called a history (Sowa, 2000). Formally, Sowa defines a state as a set of properties describing some discrete process \(P\) during an interval of time, and an event as a change in state at a specific point in time, some of the properties have changed, for some discrete process \(P\). Sowa’s definition could be adopted, however as we will see in the following, properties might not be the best candidate for describing complex situations.
4.3 Properties and relations

If we chose to describe processes in a state space consisting of observable properties that change over time, then we could for example describe the state at time $t$ of an abstract process involving a single object as:

$$\{x = 45; y = -115; speed = 20; heading = 25^\circ\},$$

where $x$ and $y$ denote the position of the object (in some suitable coordinate system), $speed$ is the speed of the object and $heading$ is the heading of the object. The state of an abstract process consisting of multiple objects could, similarly, be represented as a union of the properties for several objects, one after the other. For three objects this could for example be:

$$\{x_1 = 45; y_1 = -115; speed_1 = 20; heading_1 = 25^\circ;$$
$$x_2 = 30; y_2 = 25; speed_2 = 0; heading_2 = 90^\circ;$$
$$x_3 = 550; y_3 = 79; speed_3 = 24; heading_3 = 12^\circ\},$$

where the variables have the same meaning as before and where subscripts denote which object a property belongs to. Obviously, the abstract state is only a concatenation of the separate object level states of the ingoing objects. It does not convey any meaning at a higher level of abstraction. Lambert (1999) argues that there are at least two problems with this kind of representation:

1. It quickly grows unmanageable for problems including numerous objects.
2. It is not expressive enough for capturing complex concepts.

One way of addressing these problems is to use relations. Lambert (1999) states that “relations are conceptions over and above properties” and that the use of relations provides a greater expressive power compared to only using observable properties. A perhaps primitive approach for expressing relations, yet one that matches the basic way of describing objects as a set of properties, is to attach additional properties to objects. These properties describe if specific relations hold between an object in question and another specific object. As an example, to express a relation describing if an object approaches fifth_street, a new property could be attached to every object:

$$\{x; y; \ldots; approaching\_fifth\_street\}.$$ 

The property $approaching\_fifth\_street$ is simply a value that describes if the relation approach holds between the object in question and the specified object $fifth\_street$. In order to represent the approach relation between any tuple of objects, it is necessary to add a property that describes the value of the relation
to at least one of the objects that are a part of the tuple\(^3\). Although this kind of representation offers the ability of representing relations, it quickly grows unmanageable in dynamic environments with many objects and a varying number of relations. Furthermore, Lambert (1999) argues that this form of representation brings two distinct problems:

1. General relations such as \(x\) is related to \(y\) cannot be expressed as they cannot be anchored to any distinct object since \(x\) and \(y\) are variables.

2. It cannot be identified to which object a relation should be assigned.

It is also a rather static approach for representing relations. In case a new object is inserted into the system, then the representations of existing objects must be updated in such a way as to create a new property for the new object, for each relation and tuple which the new object can be a part of. Clearly, this is not the best approach for representing relations in dynamic environments containing multiple objects. Instead, it is preferable to be able to explicitly conceptualise relations in a representation of the world. This can be achieved through the use of the syntax in for example first order predicate calculus.

First order predicate calculus is according to Russel and Norvig (1995) “by far the most studied and best understood scheme yet devised” for representing and reasoning with knowledge. In first order predicate calculus, or first order logic for short, two main ontological commitments are made: (1) the world consists of objects with properties and unique identities and (2) there exist various relations that may hold between objects in the world (Russel and Norvig, 1995). Besides representing objects, it also allows us to explicitly express relations in our model of the world on a more general level of abstraction. To clarify the discussion, the world in first order logic is not considered to be the complete world, but rather, a limited part of the world that we would like to express some knowledge about. This is also known as the universe of interest or the domain of discourse \(D\). The explicit use of relations allows us to express complex concepts, such as situations, in a more general way. For an introduction to first order predicate logic, c.f. Russel and Norvig (1995), Luger (2002), or any other introductory book on the topic.

### 4.3.1 States and events expressed in first order syntax

The syntax of first order predicate calculus allows for relations to be expressed at a more general level. Thus, instead of representing the state of some observed domain using properties, predicates could be used. This would result in a more general description of the state of some process including multiple objects. As an example, a piloting situation in progress could in first order predicate calculus be described with a complex sentence as follows:

\(^3\)The property does not necessarily need to be tied to a specific object, but it necessarily needs to be represented somewhere for every tuple.
4.3. PROPERTIES AND RELATIONS

\[ 
PilotingZone(\text{Area}_1) \land InZone(\text{Vessel}_1, \text{Area}_1) \land 
Approaching(\text{Vessel}_2, \text{Area}_1) \land PilotBoat(\text{Vessel}_2). 
\]

The piloting situation includes two vessels \text{Vessel}_1 and \text{Vessel}_2, and a piloting area \text{Area}_1. These are terms referring to objects in the domain. Naturally, this knowledge needs to have been derived (procedurally or by other means) and inserted into a knowledge base in order to be used. Nonetheless, complex situations can be represented at a higher level of abstraction. Without the use of relations it would be necessary to capture all ingoing parts of a piloting situation in terms of properties of single objects that are concatenated. As an example, the piloting boat situation could look as follows when using properties (and rectangular areas in two dimensions):

\[
\{ \text{Area}_1.\text{type} = \text{PilotingZone}; \\
\text{Vessel}_1.\text{type} = \text{Unknown}; \\
\text{Vessel}_1.\text{approaching}_\text{Area}_1 = \text{false}; \\
\text{Vessel}_1.\text{in}_\text{zone}_\text{Area}_1 = \text{true} \\
\text{Vessel}_2.\text{type} = \text{PilotBoat}; \\
\text{Vessel}_2.\text{approaching}_\text{Area}_1 = \text{true} \\
\text{Vessel}_2.\text{in}_\text{zone}_\text{Area}_1 = \text{false} \}. 
\]

This is not as attractive and as easily interpreted as the use of meaningful\(^4\) predicates. Furthermore, in case of there is a need to describe piloting situations in other parts of the environment, then it would be necessary to add more properties. As an example, to also describe piloting situations that occur in connection to \text{Area}_2, it would be necessary to add properties to vessels indicating if they are approaching \text{Area}_2. It would however also require that the values of these new properties are determined in some way.

Through the use of predicates in first order predicate calculus, we have the ability of expressing relations of any arity in a general, yet well defined, way. In accordance with Sowa (2000), but with the present discussion in mind, a state \(z\) and an event \(e\) can now be formally defined as follows:

**Definition 4.1.** A state \(z = \{a_1, \cdots, a_n\}\) is a set of atomic sentences, under interpretation \(I\) over domain \(D\), describing some discrete process \(P\) in domain \(D\) during an interval of time, where \(I\) necessarily assigns each term to exactly one object in \(D\).

**Definition 4.2.** Let \(a \in z\). Then an event \(e\) describes a change of interpretation \(I\) of \(a\) at a specific point in time.

\(^4\)The degree of meaningfulness of symbols in first order predicate calculus depends on how they have been defined and on the expertise of the reader.
4.3.2 The abstract observable universe

Recall our discussion concerning abstract processes (Section 4.2). In that view the world is understood in terms of continuous processes that are represented by discrete processes at suitable levels of abstraction\(^5\). These processes are at different points in time in different states, based on what we have been able to infer from observations and knowledge. If we choose to include all our knowledge about the observable universe, then this would represent the state of an unlabelled abstract process of our observable universe. This abstract process would indeed serve as the basis of our understanding of the world, and could be defined as:

**Definition 4.3.** The state of an abstract process of our observable universe \(z_U\), is a state \(z\) consisting of all atomic sentences in the domain \(D\), where \(D\) contains all information inferred from our observable universe.

To reduce the complexity, a closed world assumption is assumed for the observable universe, i.e. what is not known to be true is considered false. To exemplify the use of states, events and the observable universe, consider the following scenario consisting of three objects \(o_1\), \(o_2\) and \(o_3\). At time \(t\) objects \(o_1\) and \(o_2\) are close to each other. Object \(o_3\) is however at time \(t\) not close to neither \(o_1\) nor \(o_2\). The state \(z_U\) of an abstract process of our observable universe with the three objects could at time \(t\) be described as the set:

\[
z_U = \{\text{Close}(o_1, o_2)\},
\]

where the relation close is defined as appropriate. Object \(o_1\) is however moving and at time \(t+3\) the relation \(\text{Close}(o_1, o_2)\) does not hold any more and an event \(e_1 : \text{Close}(o_1, o_2) = \text{false}\) is generated. The event thus changes the interpretation of our observable universe, and the state \(z_U\) of an abstract process of our observable universe can after time \(t+3\) be described as the set:

\[
z_U = \emptyset,
\]

Object \(o_1\) continues to move and at time \(t+12\) it has moved very close to object \(o_3\). This results in an event \(e_2 : \text{Close}(o_1, o_3) = \text{true}\) being generated, again changing the interpretation. Hence, after time \(t+12\), the state \(z_U\) of an abstract process of our observable universe can be described as the set

\[
z_U = \{\text{Close}(o_1, o_3)\}.
\]

\(^5\)Lambert (2001) views the world as a monistic metaphysics of processes. There clearly is some resemblance here, although the present conception is not nearly as formally defined as Lambert’s.
As demonstrated, we have now arrived at a representation for describing processes in terms of states consisting of atomic sentences under interpretation. In this form of representation we do have the expressive power for capturing relations. However, although we also have the ability of capturing change in states; it is still only a freeze frame view of the world we are observing in a specific interval of time, or events of change at specific points in time. We do not have any means of capturing complex situations that develop over time.

4.4 Situations

The term situation has within the fusion community come to mean aggregates of many objects that are represented as networks of relations (Steinberg et al., 1999). These relations can for example be expressed using first order predicate calculus, or through the use of some other suitable formalism. Lambert (2003b) argues that the significance of relations often is underestimated, as it has changed the Aristotelian view of a world of objects with form and matter, which has endured for nearly 2000 years. Lambert traces history and follows the progress of views from Aristotle’s world of objects, and Ludwig von Wittgenstein’s world of facts (Wittgenstein, 1922), to Barwise and Perry (1983), who proposed that situations are the fundamental building blocks when assessing the world.

Situations are in situation theory (see Section 2.3.2) considered to be many things: events, episodes, scenes, changes and facts (Barwise and Perry, 1980). Furthermore, the number of situations is practically considered to be infinite and they can be combined logically to form compound situations. They are however also considered to be recursively combinable. Although this agrees nicely with the notion of processes at many different levels of detail, it is not ideal when trying to quantify the nature of situations. In order to properly define the situation recognition problem, and to determine its complexity, it is necessary to more precisely specify what situations are with respect to what we are able to observe. Lambert (2003b) argues that situations essentially are collections of spatio-temporal facts that are related to each other, where facts are expressed in some formal language and consist of relations between objects having properties. Although time clearly is part of the equation, temporal aspects of situations need to be highlighted more clearly.

In the original situation calculus (see Section 2.3.1) a situation is considered to be the complete state of the universe (McCarthy and Hayes, 1969). They are however never completely described since the universe is too large, and instead, partial information about situations is stated. Still, a situation in a domain of discourse can be completely described since the universe of interest is restricted. Moreover, a situation is considered to be a state. Recall Definition 4.3, which describes the state of our observable universe. This would indeed correspond
to the situation in a restricted universe\(^6\). Still, situations are only considered to be snapshots of the world. Lambert (1999, 2001) argues that we often wish to consider situations that exceed individual snapshots of the world.

In the extended situation calculus (Levesque et al., 1998) a situation is instead considered to be a sequence of actions. A situation is thus not per se considered to be the complete state of the universe at a point in time, but is instead considered to be a sequence of actions that has lead to the present state (situation). As previously argued however, in surveillance applications we rarely have access to the action space (if such a space exist) of the abstract process that we are observing. Instead, we must try to define evolving situations in terms of what we are able to observe, e.g. the abstract observable universe.

### 4.4.1 Situations of temporal nature

It is important to precisely define the notion of situations of temporal nature with respect to what we are able to observe. Let us therefore return to the notion of understanding the world in terms of processes. The aim is in this view to estimate the state of some abstract process with the help of machines. Previously, the state of such an abstract process was defined. However, single states are not enough for describing situations since there is a need for being able to capture evolving situations. Instead we must consider sequences of multiple states of processes. Sowa (2000) defines a process as “an evolving sequence of states and events” and a history as “a record of the sequence of states and events that existed in the evolution of some process”. Naturally, the concept of histories could be used to denote situations in abstract processes. In accordance with this, we extend our definitions to also consider sequences of historical states, histories, which we term state sequences.

**Definition 4.4.** A state sequence \(Q_P\) is a vector of states \(Q_P = <z_1, \ldots, z_w>\), \(w \geq 0\), describing the evolution of some discrete process \(P\) in domain \(D\).

Given Definition 4.4, we have enough expressive power to capture relations, the state at any particular time, as well as changes in state over time. Through the use of Definition 4.1, Definition 4.3 and Definition 4.4, we can now define an abstract process for our observable universe as:

**Definition 4.5.** An abstract process of our observable universe \(Q_{PU}\) is a state sequence \(Q_P\), where all states \(z_i\) consists of all atomic sentences, i.e. states of the observable universe \(z_U\), in the domain \(D\), where \(D\) contains all information inferred from our observable universe.

The world is thus understood in terms of atomic sentences describing information inferred from observed data. This is very similar to what is suggested in

\(^6\)I certainly agree with McCarthy and Hayes (1969), describing the universe is not feasible for reasoning. However, for conceptualising the problem this may be needed.
Lambert’s STDF model, in which the world is composed of a number of states including the present state, as well as all past states (Lambert, 2006b). But what is a situation? In the STDF model, Lambert (2009) views a situation at time $k$ as a set of transitioning situation instances, where each transitioning situation instance at time $k$ is a set of states of affairs: $\Sigma(k) = \{ \Sigma(t) \mid t \in Time \land t \leq k \}$. States of affairs $\Sigma(k)$ in this case being similar to a state $z$ as defined in Definition 4.1. Similarly, a transitioning situation instance is similar to a state sequence as defined in Definition 4.4. A situation can thus be defined for the abstract process of our observable universe, and this actually corresponds to Definition 4.5.

Is there only one situation then? Recall our initial discussion regarding processes in Section 4.2. Processes can be thought of on many levels of abstraction. Furthermore, we can think of a number of different subprocesses (at different levels of abstraction) contained within the abstract process covering our observable universe. Likewise, there are numerous subsituations within the evolving situation of our observable universe. This is also highlighted by Lambert (2009), who considers multiple transitioning situation instances, where each may consist of different aspects of interest.

To exemplify the notion of multiple situations in the abstract observable process, consider a scenario where a man walks into a store. He picks up some bread in the bread section. He walks to the dairy section, picks up some milk and then walks to the register. He stands in queue and finally he pays for his groceries and leaves the store. We could describe this with the help of a few relations that are true in specific periods of time. This is illustrated in Table 4.1, which depicts the evolution of an abstract process covering our observable universe.

<table>
<thead>
<tr>
<th>Time</th>
<th>State $z_{U_i}$</th>
<th>$i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>${ \text{Inside}(\text{Man, Store}) }$</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>${ \text{Inside}(\text{Man, Store}); \text{WalkTo}(\text{Man, DairySection}) }$</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>${ \text{Inside}(\text{Man, Store}); \text{AtLocation}(\text{Man, DairySection}) }$</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>${ \text{Inside}(\text{Man, Store}); \text{Have}(\text{Man, Milk}); \text{WalkTo}(\text{Man, Register}) }$</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>${ \text{Inside}(\text{Man, Store}); \text{Have}(\text{Man, Milk}); \text{AtLocation}(\text{Man, Register}) }$</td>
<td>6</td>
</tr>
<tr>
<td>15</td>
<td>${ \text{Inside}(\text{Man, Store}); \text{Have}(\text{Man, Milk}) }$</td>
<td>7</td>
</tr>
<tr>
<td>17</td>
<td>${ \text{Have}(\text{Man, Milk}) }$</td>
<td>8</td>
</tr>
</tbody>
</table>

As can be seen in the table, there are four relations ($\text{Inside}$, $\text{AtLocation}$, $\text{WalkTo}$ and $\text{Have}$) and five objects ($\text{Man}$, $\text{Store}$, $\text{DairySection}$, $\text{Milk}$ and $\text{Register}$). From this state sequence we can extract a number of different situations that could be interesting. For example, one interesting situation could be that a man is in the store. This situation would cover specific parts of the state sequence, namely when the relation $\text{Inside}(\text{man, store})$ is true. Another inter-
existing situation could be that the man has milk when he is inside the store. This would cover another (overlapping) part of the state sequence which in logical notation could be expressed as \( \text{Insided}(\text{Man, Store}) \land \text{Have}(\text{Man, Milk}) \). In fact, there is possibly an interesting situation for each possible combination of predicates in a state sequence. In light of this, let us define a situation as follows.

**Definition 4.6.** Let \( Q_P = \langle z_1, \ldots, z_w \rangle \) for some process \( P \). Then \( s \) is a situation in \( Q_P \) iff:

1. \( \exists z_k \in Q_P : z_i' \in s \rightarrow z_i' \subseteq z_k \).
2. \( \forall z_i', z_j' \in s : (\exists z_k, z_l \in Q_P : k < l \rightarrow i < j) \).

Every situation has a potential of being interesting depending on the context. Definition 4.6 allows us to model every potential situation in a state sequence. Given a large number of relations, a large number of objects and a quickly developing state sequence, the explicit use of all predicates is however less than optimal. Luckily, a situation as defined in Definition 4.6 can also be described using events in case events change the state of an abstract process of our observable universe in a sequential fashion. Recall, an event denotes a change of interpretation for an atomic sentence of a state (Definition 4.2). There is thus dependence between a state and the sequence of events that has been inferred. Given an initial state for the abstract process, \( z_0 \), every situation following \( z_0 \) can be described as a sequence of events. This actually corresponds very well to the notion of a situation in the extended situation calculus (Levesque et al., 1998). Here a situation is defined as a sequence of actions. Moreover, as argued in Section 4.2, actions and events are very similar as they denote changes to some abstract process, although in different state spaces. A specific sequence of events does however not denote unique situations, but rather sets of situations. Moreover, a specific situations \( s \) can be described by many different sequences of events. Still, a more compact description can be derived using events. Let us therefore define a set of situations in terms of events:

**Definition 4.7.** A set of situations \( S \) in a state sequence \( Q_P \) of some process \( P \) can at time \( k \) be described as \( S = (z_0, E_k) \), where \( z_0 \) is the initial state of \( P \) and where \( E_k = \langle e_0, \ldots, e_k \rangle \) is a sequence of events, where each \( e_i \) describes a change of interpretation \( I \) of an atomic sentence being part of the state \( z_i \) of the process \( P \).

Definition 4.7 provides a compact representation of situations, under the assumption that events are atomic and inferred in a sequential fashion.

---

\(^7\)In this chapter, the previously used vector notation and the logical notation, is considered to have the same meaning, namely as a conjunction.
4.5 The situation recognition problem

The state sequence $Q_{P_U}$ of our observable universe contains a very large number of situations (according to Definition 4.6). However, all of these are not interesting. Instead, the interest lies in finding a subset of these situations, corresponding to specific types of situations that have been defined as interesting. Let us define the domain of situations $S_D$ with respect to our observable universe $Q_{P_U}$, as consisting of all situations in $Q_{P_U}$. The situation recognition problem can then be formulated as finding a subset $S_I$ of the situations defined in $S_D$. The set of interesting situations $S_I$ depends on the purpose and goals of the system and it can be problematic to define exactly what is interesting. Instead, it can be necessary to resort to more general patterns of activity of interesting behaviours. Such general patterns can be termed templates, and the situation recognition problem can be denoted as being template based.

4.5.1 Situation templates

The purpose of a situation template $T$ is to impose constraints on the domain of situations $S_D$ to result in a subset of interesting situations $S_I$. Situations essentially consist of temporally ordered states, where states consist of $n$-ary relations and terms, and where terms refer to objects in the domain. Similarly, constraints can here be thought of at three levels:

- **Constraints with respect to terms.** A situation essentially consists of a number of predicates where the terms refer to objects in the domain. However, every situation will refer to a subset $O$ of the objects in the domain $D$. An instance of an interesting situation thus consists of a fixed number of objects. Now, a template refers to a situation type and when instantiated it should also refer to a fixed number of objects. Uninstantiated, a template simply refers to a set of variables for objects $X = \{x_1, \ldots, x_m\}$, where $m = |O|$, and where each $x_i$ will be bound to an object in $D$ for each situation instantiated by $T$. Terms can thus be constrained in two ways: (1) a priori assigning an object in the domain to $x_i$ and (2) a priori bounding the number of objects in an interesting situation, i.e. restricting the size of $X$.

- **Constraints with respect to $n$-ary relations.** A state has been defined as a set of predicates under interpretation (see Definition 4.1). A predicate $P^n(t_1, \ldots, t_n)$ consist of a name and arity $P^n$ and a tuple of terms $t_1, \ldots, t_n$, where terms may denote objects in the domain or variables for objects in the domain. Constraints can be put on the actual predicates that are part of a situation so as to be of specific type. A constraint $c_i = P_i^n(y_1, \ldots, y_q)$ is thus also a predicate. However, it necessarily has variables as terms where

---

8Predicates.
9Functions are not considered interesting since they according to Russel and Norvig (1995) can be seen as complex names for objects.
each variable \( y_i \in X \). Moreover, a template can consist of a set of constraints \( C_s \) and only situations consistent with \( C_s \) unified over \( X \) are considered interesting situations.

- **Constraints with respect to the temporal order.** States are temporally ordered, meaning that state \( i \) is true before state \( i + 1 \) is true. States consist of predicates and to enforce temporal constraints on states, one could enforce temporal constraints with respect to individual predicates, such that for example \( P_1(x, y) \) should be true before \( P_2(x, y) \) is true. A set of temporal relations is thus considered: \( TR = \{ TR_1, \ldots, TR_q \} \), where each temporal relation \( TR_i \) is defined over a binary tuple of constraints and where a constraint denotes a predicate. Temporal relations could for example include: before and after, and could be based on for example Allen’s (1983) temporal interval algebra. Another set of constraints can thus be introduced to also enforce partial temporal order amongst constraints on the contents of states: \( C_t = \{ TR_i(c_j, c_k) | TR_i \in TR \land c_j, c_k \in C_s \} \).

Given \( X \), \( C_s \) and \( C_t \), a template can now be defined as \( T = (X, C) \), where \( C = C_s \cup C_t \). In situation theory, situations are instances of situation types. As argued, a template defines a situation type. The situation recognition problem thus involves determining if a situation \( s \) can be considered to be an instance of the situation type described in a template \( T \).

### 4.5.2 Situation recognition defined

The situation recognition problem could now be considered to be defined, since it consist in constraining a domain of situation \( S_D \) to a set of interesting situations \( S_I \), such that \( S_I \) contains all situations \( s \) that can be considered instances of a template \( T \)? However, one important question remains: how do we algorithmically solve this problem? In other words, how exactly do we constrain \( S_D \) to \( S_I \) using a template \( T \). Two important classes of algorithms can be identified: (1) exact matching techniques and (2) approximate matching techniques.

In exact techniques, all aspects of a template must be fulfilled by a situation found in the input stream. In approximate techniques, the matching does not need to be exact. Instead, the aim is to determine some degree to which the input stream matches the target template. Approximate techniques require that we establish some form of similarity measure for comparing possible solutions with the target. It may not be all that wise to use exact techniques. As already argued, it can be hard to specify every ingredient in those situations that we are interested in finding. For example, when shopping milk, one might go past the vegetable section on the way to the dairy section, or in one store, perhaps the milk is located in the beverages section. Furthermore, in surveillance systems we do not often have exact and perfect information. It is rather the opposite, and much of the information is uncertain and seldom do we have perfect coverage. Hence, there is a need to focus on approximate techniques. Furthermore,
it would also be beneficial if a ranked list of potential instantiations of interesting situations could be provided, thus allowing a decision maker to easily sift through the most prominent matches of situations. This can be very important in situations where time pressure is an issue to acknowledge. Let us therefore define the situation recognition problem as follows.

Definition 4.8. An approximate solution to a template based situation recognition problem consist of a ranked set of situations $S_I$ such that $s \in S_D \land f_a(T, s) > \epsilon \rightarrow s \in S_I$, where $S_D$ is the domain of situations formed from a state sequence $Q_P$, $T$ is a template, $f_a$ is a similarity function and where $\epsilon$ is a similarity cut-off value.

### 4.5.3 Problem complexity

Although we have now defined the problem we have not yet determined its complexity. This section provides a brief discussion around the complexity of the problem space coupled to the situation recognition problem. In computational complexity theory, we are often interested in finding the computational complexity in time and space for solving a specific problem. This section will however only look at the computational complexity with respect to time. Nevertheless, it is of utmost importance to also look at the computational complexity in space. This has, however, been left for future work.

A naïve approach for solving the situation recognition problem is to extract all situations from a state sequence and for each situation, determine if that particular situation is a part of the solution. We have already defined a function $f_a$ for determining the similarity between a situation and a template. This function could be redefined as $f_a(T, s, \epsilon) = \{yes, no\}$, where yes means that $S$, is part of the solution, and where no means that it is not. The naïve approach would then consist of finding all situations $s$ and for each use $f_a$ for determining if that situation is part of the solution or not. In this form, the computational complexity in time becomes directly dependant on the number of unique situations that can be extracted. The similarity function can also be quite complex and contribute to the total computational complexity in time, however, let us begin with assuming that it has a constant complexity in time.

A state sequence was previously defined as a vector of states

$$Q_P = <z_1, \ldots, z_w>,$$

where a state was defined as a set of predicates, in which a predicate consists of a number of terms referring to objects. Given the closed world assumption, the exact number of predicates in a specific state $z_i$ cannot be determined exactly. However, let us denote it with $|z_i|$. The total number of predicates in a state sequence of length $w$ can then be calculated as:
4. DEFINING SITUATION RECOGNITION

\[ |Q_P| = \sum_{i=0}^{w} |z_i|. \]

For an empty state sequence, the number of situations is zero. For a state sequence with one state, i.e. \( w = 1 \), there are \( 2^{|z_1|} - 1 \) possible situations\(^{10}\). For \( w = 2 \), there are \( x = 2^{|z_1|} - 1 \) situations using only \( w_1 \), \( y = 2^{|z_2|} - 1 \) situations using only \( w_2 \) and \( z = (2^{|z_1|} - 1) \cdot (2^{|z_2|} - 1) - 1 \) situations in the combination. In total, there is thus \( x + y + z = 2^{|z_1|} - 1 + 2^{|z_2|} - 1 + (2^{|z_1|} - 1) \cdot (2^{|z_2|} - 1) - 1 \) situations.

This can in fact be calculated using the power set of the multi set consisting of all elements in \( Q_P \), i.e. \( x + y + z = 2^{|z_1|+|z_2|} \). The total number of situations in a state sequence of length \( w \) can thus be calculated as \( 2^{|Q_P|} \). There are thus many possible situations. As an example, consider \( |z_1| = 3 \), \( |z_2| = 4 \) and \( |z_3| = 5 \). This results in \( 2^{12} \) potential situations. Some of these may of course represent the same situation since predicates part of consecutive states do not yield unique combinations for every consecutive state they are part of. The naïve solution does however not consider this case, but is instead inclined to test every subsituation, resulting in a very complex problem. Still, a semi naïve solution would also be rather complex. Consider the following state sequence:

\[ Q_P = \langle \{A, B, C\}, \{A, B, C, D, E\} \rangle. \]

As can be seen, it is only the difference between two consecutive states that needs to be considered, i.e. \( \{D, E\} \). The unique number of situations can in this case be calculated as \( 2^3 - 1 + 2^2 - 1 + (2^3 - 1) \cdot (2^2 - 1) = 7 + 3 + 7 \cdot 3 \). In general, the number of unique situations can be calculated recursively as:

\[ |Q_{P_w}| = Q_{P_{w-1}} + 2^{|z_w| - |z_{w-1}|} - 1 + Q_{P_{w-1}} \cdot (2^{|z_w| - |z_{w-1}|} - 1), \]

which can be simplified to

\[ |Q_{P_w}| = 2^{|z_w| - |z_{w-1}|} \cdot (Q_{P_{w-1}} + 1) - 1. \]

At a minimum two consecutive states need to differ with respect to at least one predicate\(^{11}\). Assuming that \( |z_w| - |z_{w-1}| \) always is 1 (predicates are for example added or withdrawn in sequence) gives the following recursive formula:

\[ |Q_{P_w}| = 2 \cdot Q_{P_{w-1}} - 1. \]

In other words, the number of possible situations in a state sequence does at a minimum double for each new state in the state sequence, i.e. there is an exponential growth in the number of situations even in semi naïve solutions.

---

\(^{10}\)Indexing here starting at 1.

\(^{11}\)Otherwise the two states are equal and their interval can be joined to a single state.
Up to this point we have neglected the complexity of the function for determining if a situation is a part of the solution or not. This function can actually be quite complex in itself. For example, let us depict a potential situation consisting of \( r \) predicates. Also assume that the template that we are trying to match with consists of \( c_n \) constraints. Each predicate in the subset could potentially be matched with each of the constraints in the template. This leaves us with \( r^{c_n} - r \) possible assignments. That is, another exponential function, but with respect to the size of templates. Last but not least, consistent unification of terms also needs to be assured, again adding to the complexity. From the discussion so far, we can conclude that naïve approaches for solving the situation recognition problem are infeasible. We thus need to resort to other models of computation, models that require less computational resources and which also possibly make some assumptions that can be used to lower the complexity.

### 4.6 Situation recognition in a wider context

A conceptualisation and definition of the situation recognition problem has been suggested. Still, situation recognition is not something that exists in isolation. It is part of a larger system. Typically, such systems are in the information fusion domain discussed in light of the JDL model (see Chapter 2). In order to put situation recognition into a larger context, we will discuss it in light of the JDL model.

#### 4.6.1 System overview

The JDL model is not a prescriptive model (Steinberg and Bowman, 2004). It does however provide insights into the general information flow when working with surveillance systems. It recognises that there can be a large number of data sources, ranging from sensors to databases consisting of knowledge. Typically, the physical world is sensed and decluttered to establish measurable properties. These properties are assembled into unique objects or entities, which are tracked over time. The steps so far concern levels 0 and 1 of the JDL model, and thus address the issues of association, correlation and combination. The next step lay in inferring relations that may exist between objects in the world, ranging from grouping to more complex concepts. Relations are in general considered to be defined in some domain ontology or similar. The extraction of relations appears as a natural step before situation recognition, which is depicted to use relations as input. Recognised situations can again be used in for example impact assessment, or they can be analysed further by human operators. As argued, situation recognition does not exist in isolation. It depends on lower level fusion, and it serves higher level functions — it is depicted as being part of a system. This is illustrated in Figure 4.2.

Data and information from object fusion is used as input to a symbol grounding process in which specific relations are extracted. Naturally, exter-
nally created events and facts may also be used; however, symbols do not simply emerge in the system. Situation recognition maintains a set of partial matches (hypotheses) between the continuous flow of information and a set of templates of interesting situations. The set of partial matches can be used by external processes, such as impact fusion or human operators, and complete matches can be used for raising alarms. Also depicted in Figure 4.2 is a knowledge gathering process. This process serves the purpose of maintaining knowledge concerning templates, as well as symbol grounding functions, and it may concern both manual and automatic processing.

4.6.2 Solution requirements

There are a number of requirements that can be put on solutions to the situation recognition problem. A few of these requirements are discussed in the following. It should be noted that the requirements are not listed in order of importance. The importance of the requirements will depend on the purpose of specific solutions to specific problems in specific application domains.

- **Completeness.** Every instantiation of an interesting situation type should be found. Picture a case where the interesting situation is a double meeting separated in time. Let $T = (X, C)$ be a template with three variables, $X = \{x_1, x_2, x_3\}$, and three constraints $C = \{c_{s1}, c_{s2}, c_{t1}\}$, where $c_{s1} = \text{Meet}(x_1, x_2)$, $c_{s2} = \text{Meet}(x_1, x_3)$ and where $c_{t1} = \text{Before}(c_{s1}, c_{s2})$. In the input stream the following three events are found: $\text{Meet}(A, B)$, $\text{Meet}(B, C)$ and $\text{Meet}(B, C)$. Now, it is not only the case that we are interested in finding the binding $\{x_1 = A, x_2 = B, x_3 = C\}$, but also, bindings $\{x_1 = A, x_2 = B, x_3 = D\}$ and $\{x_1 = C, x_2 = B, x_3 = D\}$ may be equally interesting. A
complete solution finds all instantiations whereas an incomplete solution does not.

- **Recognition performance.** A solution to the situation recognition problem should in its essence find instances of interesting situations as defined by some model. It is however also of importance that uninteresting situations are not classified as interesting, i.e. the false alarm rate should be low. The recognition performance on the task should thus be high. Typically, performance can be determined using two measures: (1) we want the solution to recognise as many of the interesting situations as possible and (2) we do not want the solution to recognise situations that have not been declared as interesting. Formally, a solution should have high recall and high precision. The use of the precision and recall metrics are discussed in for example Lingard and Lambert (2008); Schrag et al. (2007)\(^\text{12}\). In the context of recognising situations, the topic of achieving high accuracy actually covers to different questions: (1) how well can the model be used for classification (recognition) and (2) how well does the model represent that which it is intended to model? In the machine learning community, it is generally the former notion that is the focus. When working with models constructed by humans, the second notion may be equally important since the conceptual and implemented models may differ.

- **Robustness.** In surveillance environments we do not always have a complete view of the world. Some pieces of information may be missing, and this must be taken into account. It can thus be of importance that a solution is able to recognise partially matching situations even though some pieces of information are missing. This will, however, often come at a cost in terms of recall and precision; they will likely be lower. Thus, robustness affects recognition performance. Still, robustness may be of high importance in domains with high degrees of incompleteness and uncertainty. The system should be operational although information may be missing or may be uncertain.

- **Understandability.** Situation recognition should support human decision making. It is thus of importance that the content of recognised situations can be understood by humans. Furthermore, humans, not machines, know what an interesting situation consists of in terms of system goals. Therefore, situation templates need to be definable by humans. The importance of explainable and understandable models is also highlighted by Jensen et al. (1995) and Bladon et al. (2002).

- **Time consumption.** In online systems for situation recognition there are also requirements coupled to processing time. We are interested in knowing now

12For an extensive view of the wider concept of accuracy, see for example the work of Mahoney et al. (2000) which discusses accuracy of situation assessment systems.
if something is occurring now; there are deadlines for when decisions need to be made. This puts requirements on a system to be sufficiently efficient with respect to time and to the amount of information that needs to be processed in a specified interval. The deadlines, and the amount of time available for processing, are highly dependant on the application domain, and sufficiently efficient algorithms are needed. In conclusion, we need to process information at least as fast as the average rate at which is made available during a specific interval. This leads us to measure the amount of time that is spent on processing information, put in relation to the pace at which information is made available, as well as to memory consumption.

4.7 Chapter summary

This chapter has addressed the first research objective in this thesis and has provided a conceptualisation and a definition of the situation recognition problem. The chapter started out with a broad understanding of a world consisting of processes. The scope was narrowed down to focus on the problem of defining and recognising situations in the abstraction of processes. As pointed out however, situation recognition is not something that exists in isolation since it is part of a larger system that ultimately should support decision makers in achieving enhanced situation awareness. The scope was therefore widened and situation recognition was put in the context of a larger system using the JDL model for information fusion. Lastly, a set of requirements that can be put on solutions to the situation recognition problem were discussed.

Although there may be many viable approaches for understanding the problem, the suggested conceptualisation has served the present thesis work very well. In the initial phase of the thesis work, much time was spent on discussing an unspecified problem together with our industrial research partner. The need from industry was expressed in terms of solutions to the more general situation assessment problem. In itself, this is a too large problem to solve as it includes many different aspects (refer to Chapter 2 for a short review). As the conceptualisation presented in this chapter took form, a more precise and delimited problem emerged. This was indeed a necessity for finding relevant literature and to eventually arrive at the Petri net based technique that is presented in the next consecutive chapter.

It is believed that the conceptualisation can provide a good starting point for further studies into the problem of recognising situations of temporal and concurrent nature. The chapter has also looked into the complexity of the situation recognition problem. Although the present complexity analysis only gives a feel for how complex the problem is, it clearly highlights that naive approaches are not the way forward. The complexity of recognising situations however needs to be investigated more thoroughly in future work.
Chapter 5
Petri net based situation recognition

In the previous chapter a conceptualisation for addressing the situation recognition problem was introduced. This chapter addresses specifically the second research objective and presents a specific solution to the problem. The chapter is based on two publications: Dahlbom et al. (2009b) and Dahlbom et al. (2010a). The contribution of this chapter is:

- A technique for situation recognition using Petri nets. The technique builds upon and extends previous work on recognition using Petri nets by Ghanem et al. (2004); Ghanem (2007); Lavee et al. (2007); Castel et al. (1996); Perše et al. (2008).

5.1 Iterative situation recognition

In its essence, a solution to the situation recognition problem aims at finding all situations of interest $S_I$ in the domain of situations $S_D$, which consists of all situations in a state chain $Q_P$. Situation recognition can be depicted as a black box. The input to the black box consists of a set of situations $S_D$ and, in the simple case, a single template $T$. The output of the black box is a set of situations $S_I$. Figure 5.1 illustrates.

![Figure 5.1](image)

Figure 5.1: Situation recognition as a black box with a template $T$ and the domain of situations $S_D$ as input, and with a subset of interesting situation $S_I$ as output.
Given $S_D$ and $T$, the naïve approach requires an exponential amount of time for deriving $S_I$. Let us instead look at the problem from the perspective of gathered information. $Q_P$ represents the evolution of an abstract process of our observable universe. Initially, $Q_P$ does not contain any situations since it does not contain any states. Thus, $S_D$ is initially empty. From previous definitions it is clear that it is only through events that the abstract observable process evolves, i.e. predicates of states are added or revoked. Thus, it is only when events are processed that $S_D$ grows. Per its definition, $S_I$ can only grow when $S_D$ grows and consequently it is only through the processing of events that the set of interesting situations may change. A slightly different model can thus be depicted, in which the input consists of events and the output consists of changes to the set of interesting situations $S_I$. This is illustrated in Figure 5.2.

![Figure 5.2: Illustration of situation recognition as an iteratively updated black box whose input is an event and whose output is the change in the set of interesting situations $\Delta S_I$. A template $T$ is of course also needed, but this is only provided once.](image)

Assume that $S_D(k)$ depicts the set of situations after the $k$th event $e_k$ has been processed. When $e_{k+1}$ is processed, it is only the set of situations $S_D(k + 1) - S_D(k)$ that needs to be tested for inclusion in $S_I(k+1)$. From this, an iterative matching procedure can be formulated. As stated, $S_D(k)$ contains a set of situations $S_I(k)$ that fully matches $T$. $S_D(k)$ however also contains another set $S_P(k)$ which partially matches $T$. Let us denote a subset of candidate situations $S_C(k+1)$ in $S_D(k+1)$ as the set of situations that has the potential of matching $T$ after $e_{k+1}$. $S_C(k+1)$ actually corresponds to $S_P(k)$ but with the inclusion of $e_{k+1}$ in each situation in $S_P(k)$. By keeping track of $S_P$ for each event that arrives, the complexity of the matching procedure can possibly be lowered. For each new event $e_k$, $S_P(k)$ can be derived from $S_P(k-1)$ by three steps: (1) inserting a new situation into $S_P(k)$ instantiated only from $e_k$, (2) inserting all situations from $S_P(k-1)$ that still match $T$ and (3) inserting all combinations of $e_k$ and $S_P(k-1)$ that partially match $T$. This in fact represents an iterative approach to situation recognition, where a set of partially matching situations is maintained and can be used for recognising instantiations of $T$ in $S_D$. For the iterative solution to work, however, it is only allowed to process one event at a time, i.e. events must be arriving in a sequence.
From the above discussion, we can conclude that solutions for addressing the situation recognition problem preferably should be iterative in nature, i.e. as data is processed, a space of partially matching situations should iteratively be built and maintained. This effectively limits the number of operations that are required for each new piece of information that is processed. The alternative is to carry out the complete matching procedure for each new piece of information that is added. Per its definition, the former approach represents a more efficient solution than the latter. It is however possible that it degrades to the latter approach in cases where every situation is considered an interesting situation. The Petri net based technique that is introduced in this chapter represents an instantiation of the former approach.

5.2 Analysis and problems

In Chapter 3, we looked at two approaches for using Petri nets to recognise complex patterns. The object Petri net technique described by Ghanem et al. (2004); Ghanem (2007) seems very suited for recognising complex situations that take place over time and which consist of many objects. Object Petri nets are considered preferable compared to plan Petri nets (Castel et al., 1996), since they allow multiple instances of situations to be represented in a single Petri net. There are however a few problems that need to be considered to allow for successful usage of the object Petri net technique for recognising situations.

- The complete matching space is not considered. As argued in Chapter 4 completeness is an important requirement that can be put on solutions to the situation recognition problem. To recapitulate, consider the interesting situation of a double meeting: x meets y before y meets z, and the event stream $\text{Meet}(A,B)$, $\text{Meet}(B,C)$ and $\text{Meet}(B,D)$. As argued, all tuples of the behaviour may be interesting: $<A,B,C>$, $<A,B,D>$ and $<C,B,A>$. The object Petri net technique does not allow for the recognition of the latter two tuples, since the recognition of the first tuple results in the remaining tuples not being recognisable. The reason is that tokens are consumed when they are used. Thus, when using $\text{Meet}(A,B)$ and $\text{Meet}(B,C)$ to results in tuple $<A,B,C>$, there are no partial matches that can be combined with the last event $\text{Meet}(B,D)$.

- Role assignment is not properly managed with respect to templates. Tokens in object Petri nets consist of a list of actors that are appended for each matched event. Furthermore, multiple tokens are inserted to cover for multiple possible roles of an actor. Still, it is not clear from the specification of tokens which actor that corresponds to which role in a template of an interesting situation. This becomes a problem from two perspectives. It may be a problem if role assignment is not specified in a clear and precise manner.

1And so does the rule based technique that is used for comparison in Chapter 10.
from an understandability perspective, since a human will have problems in intuitively inspecting tokens to understand which actor that corresponds to which role. Consider a pick pocketing situation that is recognised in action and for which there may be decision deadlines with respect to suitable actions. Is it person A or person B that is the victim — should we put resources on capturing A or B? It may however also be a problem from a unification perspective. If it is not known which actor corresponds to which role, then correctness in matching cannot be guaranteed. It is necessary for each new piece of information that is included, to know which variables that have already been assigned to actors.

- The matching procedure is not precisely defined. The matching procedure described by Ghanem et al. (2004); Ghanem (2007) consist of matching the actors of external events with actors in combinations of tokens from the input places of the transition handling the external event. Moreover, Ghanem et al. (2004) state that each possible combination of tokens from the input places is tested with the external event. Still, it is not defined what constitutes a valid combination of tokens and it is not precisely defined what constraints that are put on the external event for it to be matched with valid combinations. These are important properties in order to assure that the matching procedure is complete and correct. Furthermore, in the case of situation recognition, a template has been defined to consist of a number of constraints. A Petri net representation needs to clearly define what constitute each constraint and there needs to be a clear mapping between constraints and transitions.

Ghanem et al. (2004) also describe the use of a list of enabled transitions. These are transitions that have valid combinations of tokens at their input places. Furthermore, it is only enabled transitions that are invoked when external events are processed, and Ghanem et al. (2004) argue that this constitutes an efficient solution since the number of enabled transitions will be small. Intuitively this may seem beneficial however, depending on the complexity coupled to the task of deriving valid combinations, this may consume more memory and time compared to investigating each transition at event activation. This depends on the input stream of events, as well as on the type of situation that is being recognised. Furthermore, the number of enabled transitions is not per default small, rather, it is also dependent on the input, i.e. the events that have been processed.

- Incompleteness in incomplete matching. As described by Ghanem et al. (2004) object Petri nets does not allow for the recognition of partially matched patterns. As argued in Chapter 4, robustness and the ability of coping with missing information can be very important in many domains. The

---

2 It is possible that their technique actually solves these problems, however, it is not properly described and defined in literature.
extensions described by Perše et al. (2008) incorporates a facility for coping with missing information, by keeping a penalty score with each token. Still, it may be of importance to actually know which pieces of information that are missing, in order for decision makers to have as much information as possible at their disposal for analysing any potential matches further.

5.3 Representation

To address the identified issues, an extended technique is suggested. This technique incorporates the main ideas of Ghanem et al. (2004); Ghanem (2007), but ties the concepts to partial match space modelling between a sequence of events and a template. As in existing definitions, a Petri net is a quintuple

\[ PN = (P, T, F, W, M_0), \]

where \( P \) is a finite set of places, \( T \) is a finite set of transitions, \( F \) is a set of arcs, \( W \) is a weight function and where \( M_0 \) is the initial marking. For the purposes of matching, all weights on arcs in \( F \) are 1. The initial marking \( M_0 \) consist of one or more tokens at each input place, i.e. all places that are not used as output from any transitions have tokens to start with. The set of transitions \( T \) is defined as the union of two sets, \( T = T_E \cup T_R \), where \( T_R \) are regular transitions and where \( T_E \) are conditional transitions. Similar to Ghanem et al. (2004) conditional transitions in \( T_E \) are activated when invoked by external events and when their input place restrictions are fulfilled. Regular transitions are activated when their input place restrictions are fulfilled. In the following, each of the components of the extended Petri net based technique for recognition is presented in more detail, starting with tokens.

5.3.1 Tokens

First and most importantly, tokens represent partial matches between a situation template and the stream of events that has been observed. This follows the ideas of coloured Petri nets (Jensen, 1991), in which tokens are used as carriers of information. Recall, a template

\[ T = (X, C) \]

consists of a set of variables \( X \) and a set of constraints \( C \). A token represents a subset of a template, namely the variables \( X \) and a subset \( C' \) of \( C \) consisting of all non-temporal constraints in \( C \). Temporal constraints do not need to be modelled since they are modelled implicitly by the graphical structure of the Petri net. Constraints in a token can be bound to predicates of processed events and variables can be bound to real objects denoted by predicates. A token in which all variables have been bound to real objects, and in which all constraints
have been bound to events, is considered a perfect match of the modelled situation. A token in which nothing has been bound is an empty partial match. In the following, an unbound variable or constraint is denoted with $U$ and bound variables and constraints are denoted by a number from $\mathbb{N}$ and correspond to the sequence number of an event or to the id of an object.

Two tokens can be combined with each other if: (1) there are no variable bindings that stand in conflict with each other and (2) there are no constraint bindings that stand in conflict with each other. For example, if $x_1$ has been bound to object 3 in one token and to object 6 in another token, then these two tokens stand in conflict with each other and cannot be combined. Formally:

**Definition 5.1.** Two tokens $t_1 = (X_1, C_1)$ and $t_2 = (X_2, C_2)$ are combinable iff

1. $\forall x_i (x_i \in X_1 \land \exists x_j (x_j \in X_2 \land i = j \land (x_i = x_j \lor x_i = U \lor x_j = U)))$.
2. $\forall c_i (c_i \in C_1 \land \exists c_j (c_j \in C_2 \land i = j \land (c_i = c_j \lor c_i = U \lor c_j = U)))$.

Two combinable tokens are combined with each other by creating a new token which consist of the union of the variable- and constraint bindings of the two tokens. Formally, the union of two sets of bindings (either variable bindings or constraint bindings) is defined as:

**Definition 5.2.** Let $X$, $Y$ and $Z$ be sets of variable or constraint bindings in three tokens $t_1$, $t_2$ and $t_3$, respectively. Given that $t_2$ and $t_3$ are combinable, then the union of two sets of bindings $X = Y \cup Z$ is carried out by creating a new set $X$ of the same size as $Y$ and $Z$ and assigning to it as follows:

$$x_i = \begin{cases} y_i & \text{if } y_i \neq U \\ z_i & \text{if } z_i \neq U \\ U & \text{otherwise} \end{cases}$$

Three or more tokens can also be combined since token combination is commutative. This allows for tokens to be combined recursively in any order. Through token combination, we have the ability of recognising situations by iteratively refining the space of partial matches as new information is made available. There is however also a need for a scheme that identifies which tokens to combine, and when to combine them. This is taken care of by transitions.

### 5.3.2 Transitions

Two different kinds of transitions are used. The first is the basic Petri net transition, which is activated when a new set of combinable tokens exist in its input places. This type of transition will be referred to as an *unconditional transition* and will, when activated, combine all new sets of combinable tokens over its input places. This results in new tokens being produced at the output places of the transition. The second type of transition is a conditional transition, introduced
5.3. REPRESENTATION

by Ghanem et al. (2004), to which constraints can be assigned. For example, the constraint $\text{Approach}(x, y) = \text{true}$ can be assigned to a conditional transition, which in turn would be activated when an event of that type and value is processed in the system and if a combinable set of tokens exist in its input places. Conditional transitions are associated with non temporal constraints in a template, and for each non temporal constraint in a template, there exists one corresponding conditional transition. Moreover, since specific content of tokens is also associated with the non temporal constraints, each transition is associated with a specific part of tokens. More specifically, each transition is associated with a constraint $c_k$ in $C'$ and with a subset of variables $X_T$ in $X$.

The activation procedure of a conditional transition proceeds as follows. A new token representing a processed event is created when a conditional transition is activated. More specifically, $c_k$ is bound to the event sequence number and each $x_i$ in $X_T$ is bound to one of the objects in the tuple of the event. To clarify, if $X = \{x_1, x_2, x_3, x_4\}$ and the condition of the transition represents $\text{Close}(x_1, x_3)$, then $X_T = \{x_1, x_3\}$. Moreover, if the 15th event is $\text{Close}(3, 18)$, then the new token that is created to represent the event has the following bindings: $c_k \leftarrow 15$, $x_1 \leftarrow 3$ and $x_3 \leftarrow 18$. The relation $\text{Close}$ is however symmetrical, meaning that $\text{Close}(18, 3)$ is also true. A second token is therefore created, but now $x_1 \leftarrow 18$ and $x_3 \leftarrow 3$. This extra token is created for all symmetrical relations, but not for asymmetrical relations. Each new token is after creation combined with every set of combinable tokens in the input places of the transition. Valid combinations are then inserted to each output place of the transition. Conceptually, new token combinations represent partial matches in which the processed event is taken into account as additional information.

Recall the problem of completeness: all instantiations of an interesting situation need to be found. To address this, tokens are in contrast to regular Petri nets not consumed by default when transitions are activated. The reasoning behind this is that we wish to keep the complete matching space to be able to recognise all instantiations of a situation. The space of partial matches however grows rapidly when tokens are not consumed. To address this, it is here suggested that a global time bound is set for each situation template, i.e. the time between the first and last events should be less than some fixed amount of time.

5.3.3 Places

Places in the approach represent partial stages of the matching procedure between a modelled situation and the stream of events. A token at a specific place thus means that there is a partial match at that stage in the matching procedure. There are three types of places. First, input places are assigned an empty partial match and do not have any arcs leading to them. These serve as initiators for matching new occurrences of a situation, since any transition that only have
input places as input always can produce new tokens for every event type that is matched\(^3\). This effectively solves role assignment implicitly in the technique.

Secondly, match places denotes the existence of complete matches. As soon as a token is inserted to a match place, a situation has been recognised and an operator can be alerted to analyse the information further. Match places do not have any outbound arcs to transitions, and similar to FSA and FSMs match places are in graphical notation denoted with double circles.

The third type of place is termed a not match place. Although a template focuses on defining the contents of an interesting situation, it can also be of importance to be able to define what an interesting situation does not consist of. Besides using constraints that model something that should be the case, constraints can also be used to model something that should not be the case. To incorporate this, the third type of place is introduced as a means of removing partial matches that should not be considered for future matching. Tokens ending up in not match places are removed directly. However, not match place also have the capability of consuming tokens from their input places to incorporate functionality of something similar to an XOR gate.

Now that both transitions and places have been introduced, the basic layout of a Petri net representing a template of an interesting situation can be illustrated. Let \( T = (X, C) \) be a template describing an interesting pick pocket situation, where \( X = \{x_1, x_2, x_3\} \) and where \( C \) consist of five constraints with respect to predicates in states and five temporal constraints with respect to non temporal constraints. A Petri net\(^4\) for representing the interesting situation is illustrated in Figure 5.3.

\[\text{Approach}(x_1, x_2) \quad \text{Close}(x_1, x_3) \quad \text{Intercept}(x_1, x_3) \]

\[\text{Close}(x_1, x_3) \quad \text{Intercept}(x_3, x_1)\]

**Figure 5.3:** Illustration of a Petri net describing a pick pocket situation type.

### 5.3.4 Representing facts

Although Chapter 4 depicted situations to evolve with events, this may not always constitute the best conceptualisation. Some information is persistent and not directly related to the observable universe. Rather, it resides in databases and consists of, e.g. types of objects and regions of interest. It can be argued that

\(^3\)A special case where this is not true is introduced in Section 5.3.4.

\(^4\)We shall return to the use of this specific Petri net in Chapter 9, Chapter 10 and Chapter 11.
5.4 ALGORITHM

Persistent information too should be added to the system using events (events that for example are distributed at system start-up). However, this information would in the Petri net based technique be consumed when the events become older than the global time bound that has been put on a template of an interesting situation. This could be solved by marking such events as non removable. In case of a distributed system such a solution would however not suffice since persistent information only would be distributed at global system start-up. Whenever new components are added or in case components are rebooted or replaced, then persistent information would be lost. Persistent information generally reside in long term storages, and a more suitable solution would be to associate some aspect of a Petri net with facts in long term storages.

One way to achieve this is to associate some input places with persistent information. Already, input places are not subject to the sliding window removal of tokens. Persistent information would thus remain in the system if associated with tokens in input places. Instead of only using empty tokens in input places, tokens in which persistent information has been bound can be inserted too. Constraints can thus be put on input places to restrict the recognition process with respect to persistent information. To exemplify, consider a piloting situation where a vessel waits in a piloting zone until a pilot boat arrives. There are three objects in this situation: a vessel, a pilot boat and a pilot area. The pilot area represents persistent information that may exist in some database. A Petri net representing the situation is illustrated in Figure 5.4.

![Figure 5.4: A Petri net with persistent information modelled in an input place.](image)

As illustrated, the input place is now constrained by $PilotArea(x_1)$, which results in a set of tokens available in the input place, where each token has bound the variable $x_1$ to a specific piloting area in the region of interest. In the specific case of Figure 5.4, there are two piloting zones.

5.4 Algorithm

The algorithm for updating the marking of a Petri net follows three distinct steps that are taken for each new event that is processed: (1) partial match space pruning, (2) event-token derivation, and (3) token propagation. In the first step, too old information is removed. This serves the purpose of removing partial matches that break the global time bound of the situation being modelled. Each
5. PETRI NET BASED SITUATION RECOGNITION

place is therefore inspected to remove old tokens. This is realised through the use of a sliding window, where all tokens that have a constraint bound to an event outside of the sliding window are removed from the Petri net. Match places are excluded from the token pruning procedure since they only contain complete matches. Moreover, input places are also excluded since they only contain tokens that have not bound any events. The steps for updating the marking of a Petri net are illustrated in Algorithm 5.1.

Algorithm 5.1 Updating the marking of a Petri net $PN$ upon receiving a new event $e$.

1: function $UPDATE_{PETRI}NET(PN, e)$
2: for all places $p$ in $PN$ do
3:   for all partial matches $pm$ in $p$ do
4:     if $time(e) - time_{start}(pm) > sliding_{window}$ then
5:       REMOVE($p, pm$)
6:     end if
7:   end for
8: end for
9: output ← ∅
10: for all transitions $t$ in $PN$ do
11:   if $t$ is a conditional transition then
12:     if condition of $t$ is fulfilled by $e$ then
13:       $pm_list ← CREATE\_PARTIAL\_MATCHES(e)$
14:       $output[t] ← DERIVE\_OUTPUT(t, pm\_list, nil)$
15:     end if
16:   end if
17: end for
18: for all transitions $t$ in $PN$ do
19:   for all partial matches $pm$ in $output[t]$ do
20:     for all output places $p_{output}$ from $t$ do
21:       INSERT($p_{output}, pm$)
22:     end for
23:   end for
24: end for
25: end function

In the second step, the event is processed by each conditional transition to derive a set of new tokens that are to be inserted to each respective transition’s output places. In case the condition of a transition is matched, new tokens are constructed by trying to combine the new information with each valid combination of tokens at the input places of the transition. A valid combination consists of at least one token from each input place, in addition to the new information. Tokens are however not inserted immediately into their respective output places (sequentially ordered transitions are time constrained), but are instead kept in a list until all conditional transitions have processed the event. The algorithm for deriving output is illustrated in Algorithm 5.2.
Finally, in the third step of the update algorithm, all newly produced partial matches are inserted to their respective output places. Furthermore, tokens are also propagated further on in the Petri net, and for each conditional transition that a token passes in the propagation step, a missed event is bound instead of a real event. This allows us to consider partial matches where some events have been missed, as matching situations. This thus addresses the problem of robustness, which can be a very important aspect when working in domains where information can be incomplete (as in the case of surveillance applications). An initial measure for calculating the degree of instantiation of a situation with respect to a template is to calculate the ratio between the number of fulfilled constraints and the total number of constraints. In an enhanced measure, each constraint can also be weighted to allow for some constraints to be considered more important than others. This measure can be calculated as follows:

$$\frac{\sum_{i=0}^{\mid C \mid} (w_i \cdot f(c_i))}{\sum_{i=0}^{\mid C \mid} (w_i)}$$

where $C$ is the set of constraints in a token, $w_i \in W$ is a set of weights for constraints and where $f(c_i)$ is a function returning 0 in case constraint $c_i$ is unbound and 1 otherwise. Besides inserting partial matches and propagating partial matches to allow for missing information, each consecutive unconditional transitions of a place to which a partial match is inserted, is also invoked recursively in the insertion procedure. This effectively updates all affected unconditional transitions in the Petri net. The procedure for insertion of partial matches to places is illustrated in Algorithm 5.3.
Algorithm 5.3 Insertion of a partial match $pm$ into a place $p$, including derivation of output from consecutive non-conditional transitions.

1: function $I$nsert($p, pm$)  
2:     if $pm$ does not exist in $p$ then  
3:         if number of missed constraints in $pm < max\_misses$ then  
4:             add $pm$ to storage of $p$  
5:                 for all output transitions $t_{output}$ from $p$ do  
6:                     if $t_{output}$ is a non conditional transition then  
7:                        output $\leftarrow$ DERIVEOUTPUT($t_{output}, pm, p$)  
8:                         for all partial matches $pm'$ in output do  
9:                             for all output places $p'$ in $t_{output}$ do  
10:                               INSERT($p', pm'$)  
11:                         end for  
12:                     end if  
13:                 end for  
14:             end if  
15:         end if  
16:     end if  
17: end function  
18: end function

5.5 Illustrative example

This section illustrates the use of the Petri net based technique with a small example. In the example, incomplete matching is not considered, thus, partial matches are not propagated. The example is illustrated in Figure 5.5 - Figure 5.9, and it consists of a simple Petri net and changes to its marking as four events are processed. The Petri net consist of four places, $p_1, ..., p_4$, and three transitions, $t_1, t_2, t_3$. Tokens ($m$) consist of three variables for objects, $x_1, x_2, x_3$, and three variables for constraints, $c_1, c_2, c_3$, where $c_1 : close(x_1, x_2) = True$, $c_2 : approach(x_2, x_3) = False$ and where $c_3 : close(x_1, x_3) = True$. The initial marking of the Petri net is illustrated in Figure 5.5, and it consists of a single token in the input place, $p_1$, which represents an empty partial match.

![Initial marking of the Petri net](image)

**Figure 5.5:** Initial marking of the Petri net.
The first event to be processed is $\text{close}(54, 28) = \text{true}$. This event matches the relational constraint in the first and third transitions, which calls for further evaluation. The second transition does however not match the event, as it depicts an $\text{approach}$ constraint. In the first transition, a new token is created, in which $x_1 \leftarrow 54$, $x_2 \leftarrow 28$, and $c_1 \leftarrow 1$. $\text{Close}(x, y)$ however represents a symmetrical relation, and in order to allow for all possible matches, a second token is created in which $x_1 \leftarrow 28$, $x_2 \leftarrow 54$, and $c_1 \leftarrow 1$. The next step is to combine the newly created tokens with all tokens in the input place. The input place, $p_1$, however only consists of a single token in which nothing has been bound. Therefore, both tokens are put in the output list of $t_1$. Similarly, two new tokens are created in the third transition, but now, $x_2 \leftarrow 54$, $x_3 \leftarrow 28$, and $c_3 \leftarrow 1$, and vice versa. However, since there are no tokens available in neither $p_2$ nor in $p_3$, the newly created tokens cannot be combined with anything. Consequently, the two tokens are removed from further processing. Finally, the two tokens in the output list of $t_1$ are inserted into $p_2$. This completes the processing needed for updating the Petri net to account for the new information provided by the first event. The marking after the first event is illustrated in Figure 5.6.

![Diagram](image)

Figure 5.6: Marking of the Petri net after having processed one event.

The second event to be processed is $\text{approach}(54, 31) = \text{false}$. This event does not match the first and third transitions, whereby they are removed from further analysis. The event does however match the constraint of $t_2$. A new token is therefore created in $t_2$, in which $x_2 \leftarrow 54$, $x_3 \leftarrow 31$, and $c_2 \leftarrow 2$. $\text{Approach}$ represents an asymmetrical relation, and therefore, no more tokens are created. The newly created token is then combined with the empty token in $p_1$, after which the result is put on the output list of the second transition. This concludes the output derivation required for the second event, and the single newly created token is therefore inserted into $p_3$. The marking of the Petri net after processing the second event, is shown in Figure 5.7.

The third event to be processed by the Petri net is $\text{close}(54, 71) = \text{true}$. This time, two new tokens are created in the first transition (similar to before). These tokens are combined with the empty partial match, and the results are put on the output list of $t_1$. Again, the constraint of the second transition does not match, and no further processing is required. When the event is processed by
the third transition, two new tokens are created, in which \( x_1 \leftarrow 54 \), \( x_3 \leftarrow 71 \), and \( c_3 \leftarrow 3 \), and \( x_1 \leftarrow 71 \), \( x_3 \leftarrow 54 \), and \( c_3 \leftarrow 3 \). This time around, there are tokens available in both \( p_2 \) and \( p_3 \), whereby it must be investigated whether any of the two newly created tokens can be combined with any tuple from the Cartesian product of the content in \( p_2 \) and \( p_3 \). If we take a look at the single token in \( p_3 \), \( x_1 \) is unbound, \( x_2 \leftarrow 54 \), and \( x_3 \leftarrow 31 \). Neither of the newly created tokens can be combined with this token since \( x_3 \leftarrow 31 \), and not 54 or 71 (as in the new tokens). The tokens are thus removed, after which the marking of the Petri net is updated by inserting all new tokens into their respective places. The marking of the Petri net after the third event has been processed, is illustrated in Figure 5.8.

The fourth and final event to be processed is \( \text{close}(28,31) = \text{true} \). Yet again, the event matches the constraints of transitions \( t_1 \) and \( t_3 \). Similar to before, two new tokens representing the event are put on the output list in transition \( t_1 \). Similarly, two new tokens are created in transition \( t_3 \), where the first has \( x_1 \leftarrow 28 \), \( x_2 \leftarrow 31 \), and \( c_2 \leftarrow 4 \), and where the second has \( x_1 \leftarrow 31 \), \( x_2 \leftarrow 28 \), and \( c_3 \leftarrow 4 \). Again, all valid combinations from the Cartesian product of the input places of \( t_3 \) need to be tested for combination with the two tokens. There are a total of four combinations, of which only one is valid. In this token, \( x_1 \leftarrow 28 \), \( x_2 \leftarrow 54 \), \( x_3 \leftarrow 54 \), \( c_1 = 1 \), \( c_2 = 2 \), and \( c_3 \) is unbound. This can be combined with one of the two newly created tokens, and consequently, a new
5.6 TRADING MEMORY FOR SPEED

A partial match is placed in the output list of \( t_3 \). Finally, the content on the output lists of the transitions is inserted to their respective places. The marking of the Petri net after having processed the four events is illustrated in Figure 5.9. As can be observed, there is now a token in the match place \((p_4)\). Hence, a match has been found and an operator can be notified (or other actions of preference can be taken).

![Figure 5.9: Marking of the Petri net after having processed all four events. As can be observed, there is now a token in the match place, which indicates that we have recognised the target pattern.](image)

5.6 Trading memory for speed

An inspection of Algorithm 5.2 reveals that for each event that is matched, we need to try and combine all combinations of input tokens from each respective input place, with the newly created token. In fact, this turns out to be on the order of magnitude of the Cartesian product over the content in the input places, iteratively restricted by the set of matching subcombinations. This needs to be carried out for each event that matches the constraint of a conditional transition. For transitions with only one input place, the Cartesian product is relaxed to a linear comparison, however, in cases of two or more input places the complexity of the problem increases quickly. It thus becomes interesting to also investigate techniques that can be used for lowering the complexity. Preferably, such techniques should not degrade understandability, e.g. the graphical representation should not be cluttered.

It is often possible to trade computational time consumption at the expense of memory usage. One approach for doing so in the case of Petri nets is to have all valid combinations for each conditional transition precomputed. In other words, when the content of an input place changes, the changes can be propagated to all affected transitions. These can in turn update a list of valid combinations in their input places. This approach is in fact very similar to what is suggested by Ghanem et al. (2004), to use of a list of activated transitions. The task of maintaining such a list of activated transitions involves maintaining a list of valid combinations for each transition.
There are in the Petri net based technique two types of transitions: those with a condition and those without. The conditional transitions are only activated upon receiving external input. The non conditional transitions are however activated when their input change. The non conditional transitions can thus be used as a mechanism for implementing pre combination of valid combinations for consecutive conditional transitions. Another approach would be to extend the conditional transitions to have distinct memory and preprocessing units. These keep and maintain a set of valid combinations. It is argued that the second approach is favourable as we then avoid polluting the target concept with details for efficiency. Furthermore, it is not left for a template designer to think about issues coupled to efficiency.

The restricted Cartesian product still needs to be computed for both methods; however, instead of determining all valid combinations that each event can be combined with, this is done for each new token that is produced. It may be the case that neither of the two approaches is more efficient compared to the original specification. This mainly depends on the ratio between new events and new tokens. Ghanem et al. (2004) however claim that maintaining a list of valid combinations offers an efficient solution, and thus, it is interesting to investigate if this is the case when recognising situations too. Investigations that aim at determining any potential benefits of using pre combinations are carried out in Chapter 10.

5.7 Chapter summary

Existing Petri net based approaches have some limitations when it comes to the task of recognising situations: (1) the complete matching space is not considered, (2) role assignment is not properly managed with respect to templates, (3) the matching procedure is not precisely defined and (4) incompleteness in incomplete matching. An extended Petri net based technique for situation recognition has therefore been proposed. The suggested technique addresses the limitations and should thus constitute a good foundation for recognising situations. Moreover, an approach for possibly lowering the computational complexity of the Petri net based technique has also been suggested. This builds upon suggestions by Ghanem et al. (2004). Extensions such as this can be important since the space of partial matches grows quickly and may thus become unmanageable with respect to time.

Empirical investigations of the suggested Petri net based technique are carried out in Chapter 10. The Petri net based technique is compared with a rule based technique for recognition using the Rete algorithm and the temporal extensions suggested by Walzer (2009). The extension for possibly lowering the computational complexity is also investigated and compared with the plain Petri net based technique, i.e. the technique proposed in this chapter without the extension.
Chapter 6
Genetic algorithms for learning Petri nets

This chapter addresses the fifth research objective in this thesis. The content of the chapter is based on two publications: Dahlbom and Niklasson (2009) and Dahlbom et al. (2010b). The contributions of this chapter are considered to be:

- A method for using genetic algorithms to learn Petri net situation templates.
- Three genome representations for representing Petri net situation templates.

6.1 Learning with genetic algorithms

Genetic algorithms have since their introduction been used extensively on many problems in multiple domains (c.f. Mitchell (1997)). Mostly, however, genetic algorithms have been applied to problems in which the solutions are more or less directly transferable to vector format, e.g. consisting of vectors of bits, integers or real values. Nonetheless, work has been carried out on learning graphs and their structure or their content. Some examples include: learning Petri nets (Mayo and Beretta, 2010; Alves De Medeiros and Weijters, 2004; Nummela and Julstrom, 2005; Prashant Reddy et al., 2001; Chen et al., 2001), Bayesian networks (Larrañaga et al., 1996; Myers et al., 1999) and FSA (Bertelle et al., 2002; Niparnan and Chongstitvatana, 2002; Lobanov and Shalyto, 2007; Tu et al., 2000; Spichakova, 2007). The problem of using genetic algorithms for evolving Petri nets has also been discussed by for example Schwardy (2003) and Mauch (2003). The problem addressed in this thesis, however, involves both the task of learning a graph structure, as well as learning the content of nodes in the graph, where the content consist of predicates of varying arity. Genetic algorithms should be good candidates for this task, since they are especially suited for tasks where solutions are complex and when it is hard to estimate the value of individual aspects of a solution (Mitchell, 1997). Moreover, genetic algorithms carry out multiple searches in parallel. This too can be
important when having complex search landscapes, since partial solutions can be mixed without any specific hill climbing constraints.

An important aspect when using genetic algorithms on a learning problem is the issue of providing suitable sources of inductive bias. This includes two sometimes separate tasks. First, it is necessary to provide a source of preference bias, which in the case of genetic algorithms is given through fitness functions. Preference bias guides the evolutionary process towards solutions that are promising, with respect to criteria that are promoted with high fitness. Secondly, it is necessary to choose a suitable genome representation, as this provides a source of restrictive bias on the learning problem. For a detailed discussion about inductive bias, see for example Mitchell (1997).

6.2 Genetic procedure

A genetic algorithm is to be used for evolving Petri nets that can be used for recognising situations. The recognition task involves processing a sequence of events in order to recognise instances of interesting patterns. Similarly, the task of evaluating individuals in a population of candidate solutions also involves processing events. In the offline learning case, pairs of files are available to the evolutionary process, where each pair consist of a file containing events, and a second file containing all known instantiated behaviour patterns identified by a tuple of objects and an approximate interval of occurrence (unknown instances of the pattern may also exist). The genetic algorithm runs for a number of generations, and in each generation, each individual in the population is passed the events in order to recognise situations. After having parsed all events, each individual is evaluated on the task by establishing a fitness value describing its performance. Finally, a new population of individuals is created through selection, recombination, and mutation of genomes from the parent population, which are chosen based on their fitness values. The genetic algorithm that is suggested is presented in Algorithm 6.1, and it closely follows the general genetic algorithm presented by Luger (2002). The main difference however, is that the evaluation step consists of parsing events.

6.2.1 The initial population

The construction of an initial population is, as can be seen in Algorithm 6.1, the first step in the evolutionary procedure. This follows the traditional methodology, where a number of individuals are randomly generated until the target population size has been reached. The specifics of the initialisation procedure is shown in Algorithm 6.2, in which CREATEGENOME is a function that allocates a genome given size specifications, and where RANDOMISE is a function that assigns a random value to a gene from within its range.
6.2. Genetic Procedure

Algorithm 6.1 Genetic algorithm for evolving Petri nets.

1: function Evolution(event_list, situation_list)  
2:    population ← GENERATEPOPULATION(population_size)  
3:    while termination conditions is not met do  
4:        solutions ← CREATEPETRINETS(population)  
5:            for all events e in event_list do  
6:                for all petri nets PN in solutions do  
7:                    UPDATEPETRINET(PN, e)  
8:            end for  
9:        end for  
10:        for all petri nets PN in solutions do  
11:            CALCULATEFITNESS(PN, situation_list)  
12:        end for  
13:        population ← GENERATEOFFSPRING(solutions)  
14:    end while  
15: end function

Algorithm 6.2 Initial generation of a population of genomes describing Petri nets. In the algorithm, np denotes the number of places, nt the number of transitions and nv the number of global variables.

1: function GENERATEPOPULATION(num_individuals)  
2:    population ← ∅  
3:    while |population| < num_individuals do  
4:        genome ← CREATEGENOME(np, nt, nv)  
5:            for all genes g in genome do  
6:                RANDOMISE(g)  
7:            end for  
8:        add genome to population  
9:    end while  
10:    return population  
11: end function

6.2.2 Evolution

Evolution is carried out for a number of generations until a termination criterion has been met. In each generation, individual Petri nets are instantiated from the genomes in the evolving population of solutions. A Petri net is thus created for each genome. After this, all events in the event file are read, parsed and fed to each instantiated Petri net. This leads to an update of the marking of each Petri net for each parsed event. Situations that are found during this process are remembered indefinitely during the lifetime of each instantiated Petri net. The reason for this is to be able to correctly calculate fitness values after the parsing process has completed. Details on the mechanisms involved in updating Petri nets have already been presented in Chapter 5, and are not repeated here.
6. GENETIC ALGORITHMS FOR LEARNING PETRI NETS

6.2.3 Fitness and recombination

The calculation of fitness and the creation of a new population of individuals is the final step of the genetic procedure. Fitness values are established for each individual in the present population, and a new generation of individuals is created using selection, recombination and mutation, based on the fitness values. Details regarding fitness calculations are provided in the next consecutive section. The specifics of the suggested procedure for generating a new population of individuals are shown in Algorithm 6.3.

Algorithm 6.3 Algorithm for selection, recombination, and mutation, resulting in a new generation of genomes describing Petri nets for situation recognition.

1: function GENERATEOFFSPRING(population)
2:     population2 ← ∅
3:     while |population2| < 0.05 · |population| do
4:         genome ← ELITISMSELECTION(population, genome)
5:         add genome to population2
6:     end while
7:     while |population2| < |population| do
8:         parent1 ← ROULETTESELECTION(population)
9:         parent2 ← ROULETTESELECTION(population)
10:        child ← CROSSOVER(parent1, parent2)
11:        PROBABILISTICMUTATION(child)
12:        add child to population2
13:        if |population2| < |population| then
14:            child ← CROSSOVER(parent2, parent1)
15:            PROBABILISTICMUTATION(child)
16:            add child to population2
17:        end if
18:     end while
19:     return population2
20: end function

Elitism selection is suggested for selecting the best 5% of the individuals in the present population. These individuals should be retained as they are, and should thus not be subject to any mutation operators. The reason for using elitism selection and for not mutating these elitistically selected individuals is to keep the best, so far, evolved individuals. Any potentially promising solutions that have been discovered will thus never be lost during evolution.

The remaining 95% of a new population is suggested to be created through the use of roulette wheel selection, single point crossover and probabilistic mutation. These mechanisms are suggested to be used until the target size of the new population has been reached. To start with, two parent genomes are selected using roulette wheel selection\(^1\). Single point crossover is suggested to be

\(^1\)Details of the roulette wheel selection mechanism can be found in Chapter 3.
6.2. GENETIC PROCEDURE

used to create two new offspring from the two selected parents. Single point crossover offers a simple, yet efficient, approach for combining genomes. Thus, to keep it simple, single point crossover is suggested.

Finally, all offspring are also suggested to be subject to probabilistic mutation. For each gene in the genome of an offspring, a random number is thus drawn, and in case the number is less than the mutation rate, \( \alpha \), then the gene will be mutated. In case of binary values, ones are turned into zeros, and zeros are turned into ones. For other gene ranges, mutation simply randomises the value of the gene within the range of the specific gene type. Schwab (2004) claims that a mutation rate that is proportional to the number of genes has been found successful in many situations (Schwab, 2004), or rather, a mutation rate that is proportional to the inverse of the number of genes: \( 1/(\text{number of genes}) \).

Having a high mutation rate results in more genes (of more individuals) being mutated in each generation. Having a high mutation rate can be argued to introduce more diversity into the population and it can also be argued to widen the search of the evolutionary process. In other words, the genetic algorithm is turned into an exploration of new places of the search space. A low mutation rate on the other hand, aims at finding local optima with respect to the presently explored regions of the search space. Balancing between discovery and refinement is a known issue. Moreover, in different phases of the evolutionary process, different ratios of discovery might be preferable.

For example, in case the average fitness of the presently evolved population is too stable over a large number of generations, then it could be argued that the evolutionary process is either stuck in a local optimum, or it has found the global optimum. In case a local optimum has been found, then it can be argued that the mutation rate should be increased in order to widen the search. On the other hand, in case the average fitness is not stable, then it can be argued that the mutation rate should be lowered to allow the evolutionary process to more effectively search for optima. Similar reasoning can be found in Srinivas and Patnaik (1994). In conclusion, it is argued that the mutation rate should be allowed to vary with the average fitness over time. The mutation rate is suggested to be calculated as follows:

\[
\alpha = \frac{\delta}{\text{number of genes}},
\]  

where \( \delta \) is a value that is allowed to vary between \( \delta_{\text{min}} \) and \( \delta_{\text{max}} \). In case the standard deviation of the average fitness over the past \( n \) generations is lower than \( \sigma_{\text{min}} \), then \( \delta \) is increased. In case the standard deviation of the average fitness over the past \( n \) generations is higher than \( \sigma_{\text{max}} \), then \( \delta \) is decreased.

\(^2\)Naturally, a second child will not be created in case the target population size has been reached after the first offspring has been created.

\(^3\)And this can in most problems not be verified unless the fitness is 1.0, which indeed is very rare.
6. GENETIC ALGORITHMS FOR LEARNING PETRI NETS

6.2.4 Fitness calculations

As discussed in Chapter 4, the main objective when working with situation recognition is to achieve high performance on the recognition task. This means that as many true situations as possible should be recognised, whilst false situations should not be classified as true. In other words, the aim is to achieve high recall and high precision on the recognition task. It is thus argued that recall and precision should be the main contributors of fitness for any given individual. Recall and precision can be defined as follows:

\[
R = \frac{TP}{TP + FN},
\]

\[
P = \frac{TP}{TP + FP},
\]

where \(TP\) denotes true positives, \(FP\) denotes false positives and \(FN\) denotes false negatives on the recognition problem.

To initially guide the evolutionary search, and for introducing and keeping some degree of diversity in the population, it is however argued that a small fraction of base fitness also should be included in the fitness calculations. The importance of diversity is highlighted by Mitchell (1997). Through the use of recall \((R)\), precision \((P)\) and base fitness \((f_b)\), the fitness for an individual is suggested to be calculated as follows:

\[
f = 0.45P + 0.45R + f_b.
\]

The base fitness \(f_b\) gives a small initial fitness value and then awards additional fitness fractions if a solution has the following properties: \((v)\) valid network, \((s)\) solvable network (each global variable is referred to by a transition residing on a path from a specific match place to an input place), and \((nc)\) non-complex network (the Petri net can process events on the average rate at which they are produced). The base fitness \(f_b\) is suggested to be calculated as:

\[
f_b = 0.03v + 0.04s + 0.01nc + 0.02,
\]

in which \(v\) is 1 if the network is valid, \(s\) is 1 if the network is solvable, and \(nc\) is 1 if the network is not too complex. A Petri net is considered too complex in case the time spent on updating the marking of the Petri net when processing an event exceeds a fixed value \(nc_l\). In case a property is not fulfilled, then that respective value is set to 0. The weights in the base fitness calculation sum to 0.1, which when used in the fitness function yields a fitness range of \([0..1]\). It has in initial experimentation been found that the use of base fitness can be essential for finding promising solutions. Many experiments were initially carried out without any progress, i.e. not a single Petri net was found. One reason for this is that the search space is very large. The specific weighting factors used in the base fitness function have been determined through a trial and error process in the initial experimentation phase.
It is possible that it is not beneficial to award precision and recall equally, for achieving good Petri nets. It may be the case that it is beneficial to promote one of the two measures higher than the other. It is therefore important to also investigate variations of the fitness function defined in Equation 6.4. Two alternate fitness functions are proposed. The first awards precision higher than recall and is defined in Equation 6.6. The second alternate function promotes recall higher than precision and is defined in Equation 6.7.

\[
f = 0.75P + 0.15R + f_b \tag{6.6}
\]
\[
f = 0.15P + 0.75R + f_b \tag{6.7}
\]

\(P, R\) and \(f_b\) are defined as in Equation 6.4.

### 6.2.5 Initial seeding

Traditionally, individuals of the first generation in a population are constructed by generating genomes with randomly initialised genes. In the present case, a genome translates to a Petri net for recognition. In order to at all be able recognise something, a valid Petri net needs to have: (1) at least one input place, (2) at least one match place, (3) no places that are both input and match places and (4) at least one valid path from an input to a match place. Thus, instead of merely creating genomes randomly, it is argued that random genomes can be generated until the initial population only consist of valid Petri nets, e.g. Petri nets for which the above mentioned criteria are fulfilled. The initial generation of a population is thus suggested to be modified as illustrated in Algorithm 6.4.

**Algorithm 6.4** Generation of an initial population of genomes describing Petri nets, where the genomes are restricted to represent valid Petri nets. In the algorithm, \(np\) denotes the number of Petri net places, \(nt\) the number of transitions, and \(nv\) the number of global variables.

```plaintext
1: function GENERATEPOPULATION2(num_individuals)
2:   population ← \(\emptyset\)
3:   while \(|\text{population}| < \text{num\_individuals}\) do
4:     genome ← CREATEGENOME\((np, nt, nv)\)
5:     for all genes \(g\) in genome do
6:       RANDOMISE\(g\)
7:     end for
8:     \(PN \leftarrow \text{CREATEPETRINET}(genome)\)
9:     if \(PN\) is a valid Petri net then
10:        add genome to population
11:   end if
12: end while
13: return population
14: end function
```
The reason for using initial seeding is to possibly lower the number of generations required until finding promising solutions. A risk however, is that the quality of the resulting solutions also might become lower since the process has been more biased. This in turn may lead to a situation where fewer parts of the search space are explored.

6.2.6 Bootstrapping

Genetic and evolutionary methods are sometimes used as a bootstrapping procedure for finer search algorithms. In this work it is however suggested that the evolutionary process itself can be bootstrapped, by seeding the initial population with randomly mutated genomes based on manually defined Petri nets. The initialise function defined in Algorithm 6.2, of which a modified version for initial seeding was presented in Algorithm 6.4, can thus be modified to do this. The modified version is presented in Algorithm 6.5.

Algorithm 6.5 Initial generation of a population of genomes describing Petri nets, when bootstrapping the initialisation procedure with existing Petri nets. In the algorithm, \( np \) denotes the number of Petri net places, \( nt \) the number of transitions, and \( nv \) the number of global variables.

\begin{verbatim}
1: function GENERATEPOPPULATION3(num_individuals, PN)
2:     population ← ∅
3:     while \(|population| < num_individuals\) do
4:         genome ← CREATEGENOME(PN)
5:         for all genes \( g \) in genome do
6:             PROBABILISTICMUTATION(g)
7:         end for
8:         PN ← CREATEPETRINET(genome)
9:         if PN is a valid Petri net then
10:             add genome to population
11:         end if
12:     end while
13:     return population
14: end function
\end{verbatim}

The main difference between the bootstrapped initialisation algorithm (Algorithm 6.5), and previous initialisation algorithms (Algorithm 6.2 and Algorithm 6.4), is that instead of constructing genomes that are of a certain size (specified number of places, transitions, and global variables) with randomised genes, a genome is created from an a priori existing Petri net. The genome is after this the target of probabilistic genome mutation, described earlier, where each gene has a probability of \( 1/number\_of\_traits \) of being mutated. Thus, each genome in the initial population consists of manually defined Petri nets that have been mutated in order to introduce diversity.
Similar to the case of initial seeding, the purpose of bootstrapping the evolutionary process is to possibly shorten the number of generations required to find promising individuals. Another reason is to also exploit domain knowledge. Moreover, in cases where already existing Petri nets are to be adapted to changing circumstances, then bootstrapping becomes highly interesting. A risk is however that the search space is not properly explored since the evolutionary process is highly biased towards already fit individuals, as specified by the Petri nets used for bootstrapping.

6.3 Bit genome representation

Although the selection of a good genetic representation always is an important and difficult task, it becomes even more challenging when evolving complex structures such as Petri nets. A solid genome representation is therefore required, as it forms the basis of what can be evolved. The genome representation will for example affect:

- The performance that can be achieved on evolved solutions.
- How long it takes to evolve fit individuals.
- How often fit individuals can be evolved.

In the classical bit string representation, genomes consist of vectors of genes that can be 0 and 1. For Boolean valued problems it is straightforward to transform a genome to a problem solution. A Petri net for recognition is however symbolic in nature (as well as being a graph), which requires that two translation functions are defined, one for creating a Petri net from a genome and one for creating a genome from a Petri net. These functions must be able to translate every aspect of a Petri net to genes. Typically, however, genetic representations need to be of static size. To achieve this, the number of transitions, places and variables must be manually decided before the evolutionary process starts. Given a representation that allows for Petri nets consisting of a specified number of nodes to be described, individual substructures can be mapped to subvectors of the genome. One potential mapping is presented in the following.

In the suggested Petri net based technique for situation recognition (see Chapter 5), three special types of places are used in addition to regular places: input places, match places and not-match places. The genome representation thus needs to be able to distinguish between four place types. In the mapping that is suggested, one gene is assigned to each special place type, and when none of the genes are enabled (true), the place is considered to be a normal place. This will result in three genes that translate as: 000 regular place, 100 input place, 010 match place and 001 not-match place.

\footnote{It would have sufficed to use two genes since \(2^2 = 4\). This would also have lowered the evolutionary search space by the power of 2, however, three genes are suggested for clarity.}
Transitions are however a bit more complex, and for each transition, the following needs to be represented:

- One gene for determining if it is a conditional transition, 0 if it is a regular transition and 1 if it is a conditional transition.

- \( x \) genes for determining its optional conditional constraint, where \( x = \lceil \log_2(n_c) \rceil \), and where there are \( n_c \) different constraint types. Note that this leaves \( 2^x - n_c \) representable constraints that are unused. As an example, three genes are needed in case there are five different constraint types: 000 constraint 1, 001 constraint 2, 010 constraint 3, 011 constraint 4, 100 constraint 5, and where 101,110,111 are unused.

- One gene for describing the optional constraint value of the transition, 0 for false and 1 for true.

- \( 2z^5 \) genes for describing two constraint variables for conditional transitions, where \( z = \lceil \log_2(n_v) \rceil \), and where \( n_v \) is the total number of global variables. Again, this leaves \( 2z - n_v \) representable variables that are unused. As an example, 0011 would represent \( \text{Constraint}(x_1, x_4) \), 1001 would represent \( \text{Constraint}(x_3, x_2) \), and so forth.

Additionally, graph connectivity also needs to be represented, i.e. which places that are used as input, and which places that receives output, for each transition. An approach using connectivity matrices inspired by Larrañaga et al. (1996); Myers et al. (1999) is suggested here. For each transition, two bits are thus used for each place to determine which places that serve as input and output to each transition, respectively. This sums to \( 2n_p \) additional genes for each transition, where \( n_p \) is the number of places.

Finally, four genes are also needed to describe which variables that are used to match a target concept (2 bits for variables gives 4 variables in total, whereas a target concept perhaps only consist of 3 variables). The last four genes in the bit genome thus denote which variables that are used when matching with target concepts. The bit genome representation is illustrated in Figure 6.1.

\[ \text{Figure 6.1: Bit genome representation of a small Petri net.} \]

\footnote{Only unary and binary relations are thus allowed.}
6.3. BIT GENOME REPRESENTATION

In the figure, \( n_p \) denotes the number of places, \( n_t \) the number of transitions and \( n_v \) the number of variables. Also, \( size_t \) refers to the size of each transition in bytes, and depends on the maximum number of constraint types, the number of variables and the number of places. The size is calculated as follows:

\[
size_t = 1 + \lceil (\log_2(n_c)) \rceil + 1 + 2(\lceil (\log_2(n_v)) \rceil) + 2n_p. \tag{6.8}
\]

For clarification, Figure 6.2 shows a Petri net and its corresponding bit genome.

![Petri net and bit genome](image)

**Figure 6.2**: A bit genome representation of a small Petri net with four places, four transitions, and four global variables (of which one, \( x_2 \), is unused).

In the example mapping (Figure 6.2), a small Petri net consisting of 4 places and 4 transitions can be seen (one of the transitions is not used). This Petri net represents a double meeting behaviour where two people meet, a third person approaches one of the other two, after which the third person is close to the other of the two. As can be seen, 12 genes are required for representing places, \( 8 + 8 \) genes are required for each transition (8 for the transition in case there are at most are four global variables and four transition types and 8 for connectivity), and 4 genes four target concept matching. This sums to a total of 80 genes.

**Key observations**

The bit genome representation is very similar to the common form of representation, where individual genes are represented with bits or numbers. This may be an important aspect for successfully evolving fit individuals. However, it is a very static form of representation with at least three potential drawbacks when it comes to the task of representing complex Petri nets.

- All gene subsets representing numbers will have a capacity of \( 2^n \), where \( n \) is the number of bits required to represent the number. Assume there are five distinct event types when modelling situations. This requires three bits, resulting in a capacity of representing eight event types. A transition
referring to any of the three “non-existing” event types will however never be activated, resulting in 3/8 invalid combinations, for each transition.

- Petri nets contain distinct substructures: places and transitions. These structures may during crossover be split without any preference. It can however be beneficial to have distinct structural parts non-divisible.
- The size of the genome needs to be manually specified. This can be a very difficult task that possibly requires much elaboration.

6.4 Complex genome representation

To address the issues identified with the bit genome, a representation that makes use of a set of complex genes is suggested. It is also suggested that edge activation is disconnected from transitions, and instead is represented in a connectivity matrix located outside of transitions. This allows for the graphical structure to be kept intact, whilst content of places and transitions may change.

Three gene types have been identified: bit genes for single Boolean values, place genes for representing places and transition genes representing the content of transitions. The content of places and transitions can in this way be kept within allowed limits. Furthermore, it is also suggested that distinct structural parts are grouped into unique chromosomes\(^6\). In this way, there is one chromosome consisting of place genes, one chromosome consisting of transition genes, one chromosome consisting of bit genes (graph connectivity) and one chromosome consisting of bit genes (match concept variable activation). Chromosomes thus consist of genes, and the chromosomes are all part of the genome. Figure 6.3 illustrates the genome-chromosome-gene representation, where \(n_p\) is the number of place genes, \(n_t\) is the number of transition genes, \(n_e\) is the number of arc genes and \(n_v\) is the number of genes for variable activation.

\[\]

\(\text{Chromosome 1 (places)}\)  \(\text{Chromosome 2 (transitions)}\)  \(\text{Chromosome 3 (connectivity)}\)  \(\text{Chromosome 4 (variable activation)}\)

\[
\begin{array}{cccc}
\text{GENE} & \text{GENE} & \text{GENE} & \text{GENE} \\
1 & 2 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{GENE} & \text{GENE} & \text{GENE} & \text{GENE} \\
1 & 2 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{GENE} & \text{GENE} & \text{GENE} & \text{GENE} \\
1 & 2 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{GENE} & \text{GENE} & \text{GENE} & \text{GENE} \\
1 & 2 & 1 & 2 \\
\end{array}
\]

\[\]

\(n_p\)  \(n_t\)  \(n_e\)  \(n_v\)

Figure 6.3: The genome-chromosome-gene representation.

A genome-chromosome-gene representation however raises questions concerning how to perform genetic crossover. Should it be performed on a genome

\(^6\)Note that the word chromosome often in literature is used for denoting the whole genome.
basis or on a chromosome basis? For example, in case single point cross over is used, should each chromosome be a target of the crossover operation, or should the genome as a whole be the target of the operation. In line with the previous argumentation that distinct structural parts should be divided into different chromosomes, it is suggested that operations too should operate on a chromosome basis. In other words, it is proposed that crossover operations are conducted on a chromosome basis. The crossover operation thus results in a distinct crossover operation for each of the chromosomes in a genome, and each chromosome in an offspring thus contains parts from both parents.

The mutation rate needs to be calculated differently when using a genome-chromosome-gene representation. The reason for this is that the number of genes will be much lower, compared to when using a plain bit or numerical representation. For example, a transition gene consists of many different traits related to transitions, such as constraint type, variables and constraint value. All of these traits are represented using a single gene in the suggested representation. To take account for the effect that the decrease in genes has on mutation, it is suggested that the mutation rate should be proportional to the inverse of the number of traits, compared to the inverse of the number of genes. Each gene therefore needs to know how many traits it controls. Thus, Equation 6.1 is argued to be replaced with the following equation:

\[
\alpha = \frac{\delta}{\text{number of traits}},
\]

where \( \delta \), as before, denotes a variable that represents the stability of the population with respect to average fitness over time.

Key observations

The complex genome representation does not allow for any invalid solutions to be represented, and therefore possibly offers a better choice for quickly finding promising solutions. However, it still has some potential weaknesses.

- The number of places and transitions still need to be decided manually. Perhaps we decide on too few places and transitions, resulting in the target concept not being separable, or perhaps we decide on too many, significantly increasing the search time. This is a problem which possibly requires much elaboration and fine tuning, for each target concept that is being evolved.

- It may require excessively long evolutionary times\(^7\). The underlying reason for this is related to the use of connectivity matrices. Successful representations of target concepts will likely be sparse, since densely connected Petri nets are likely to be: (1) to complex with respect to time and (2) to unrestrictive and not provide enough constraints on the domain\(^8\). Solutions

\(^7\)Read many generations.

\(^8\)This can result in too many false alarms as well as memorisation compared to generalisation.
with many edges require many calculations. This will result in more time being spent on evaluation as well as Petri nets that are not efficient enough. However, most individuals in an initial population will, due to randomisation, have approximately half of their edges activated (a uniform distribution over 0 and 1 for each edge). This can result in many generations being needed for a population to converge on suitable edge activation for the problem at hand. It can be argued that this could be solved by not using a uniform distribution over 0 and 1, but to instead use a distribution in which it is less likely to draw a 1 than a 0. The problem may however persist since the part of the hypothesis space concerned with edge activation is still of the same size.

6.5 Dynamic complex genome representation

To address the problem of densely connected graphs, a fourth gene type is suggested for representing individual edges in the graph: arc genes. Thus, instead of having the third chromosome consisting of bit genes describing a connectivity matrix, it is suggested to consist of a number of arc genes that represent specific edges in a Petri net. Fewer edges can thus be modelled, resulting in more sparsely connected Petri nets, and possibly, fewer generations required for evolving promising individuals.

To address the problem of having genomes of static size, inspiration is drawn from Mayo and Beretta (2010)\(^9\), who use a dynamic number of edges when evolving Petri nets. Instead of having a fixed number of places, transitions and arcs, it is suggested that the evolutionary process is allowed to also evolve the size of the graph, in addition to its content and connectivity. To accommodate this, it is suggested that chromosome mutation can be used in addition to gene mutation. Chromosome mutation consists of two mutation operators: gene addition and gene removal, similar to Mayo and Beretta (2010). The gene addition operator is suggested to copy a randomly selected gene and put it at the end of the chromosome. The gene removal operator is suggested to remove a randomly selected gene from the chromosome. Figure 6.4 illustrates the dynamic complex genome representation.

In Figure 6.4, \(n_p\) denotes the number of place genes, \(n_t\) is the number of transition genes, \(n_e\) is the number of arc genes and \(n_v\) is the number of genes for match variable activation, before mutation. Three mutation operations are applied. First, a gene is removed in the first chromosome, resulting in \(n_p-1\) place genes. Secondly, gene 3 is duplicated in the second chromosome. This results in \(n_t+1\) transition genes. Thirdly, gene 1 is duplicated in the third chromosome, resulting in \(n_e+1\) arc genes.

A new problem may however arise when also evolving the size of genomes: genomes may grow too large. This can in turn lead to two distinct problems:

\(^9\)The use of variable sized genomes is also discussed in for example Mauch (2003).
6.5. DYNAMIC COMPLEX GENOME REPRESENTATION

1. The resulting Petri nets may become too complex with respect to time. It can be argued that this would solve itself through the promotion of less complex solutions over more complex solutions. This is already the case due to the variable $nc$ in the base fitness function. However, complex solutions may still negatively affect the time it takes to evolve fit individuals since much time may be needed for evaluating complex individuals during evolution.

2. Induction may fail, and instead of learning a general concept, instances are simply memorised, i.e. overfitting may occur (see Mitchell (1997) for a discussion).

In decision tree learning, a common assumption for guiding the learning process is to prefer shorter hypotheses over longer hypotheses (Mitchell, 1997). Similar reasoning can be applied here; shorter genomes could be promoted over longer genomes in order to avoid overfitting and complexity. The equation for calculating base fitness is therefore suggested to be modified as follows:

$$f_b = 0.03v + 0.04s + 0.01nc + 0.01g + 0.01,$$  \hspace{1cm} (6.10)

where $g$ is 1 for genomes of static size, and calculated according to Equation 6.11 for genomes of dynamic size, in which $gl$ is the length of the genome, and $gl_i$ is the initial length of genomes. As before, if a property is not fulfilled, then that value is set to 0. In cases with genomes of static size, the base fitness function reverts back to Equation 6.5, whilst it in the case of genomes of dynamic size will be calculated according to Equation 6.10.

Figure 6.4: The dynamic genome-chromosome-gene representation.
6. GENETIC ALGORITHMS FOR LEARNING PETRI NETS

\[
g = \begin{cases} 
1.0 & \text{if } gl \leq gl_i, \\
0.0 & \text{if } gl > 3gl_i, \\
1.0 - (gl/3gl_i) & \text{otherwise.}
\end{cases} \tag{6.11}
\]

Since genomes in the dynamic complex representation may be of different sizes, the genetic crossover needs to be modified. It is suggested that single point crossover is carried out as follows. Let \(l_1\) be the length of the first parent and let \(l_2\) be the length of the second parent. Moreover, let \(cut\) be a random number from \([0..1]\). The first child has genes \(<1, \ldots, cut \cdot l_1>\) from the first parent, genes \(<cut \cdot l_1 + 1, \ldots, min(l_1, l_2)>\) from the second parent and genes \(<l_2 + 1, \ldots, l_1>\) from the first parent. For the second child, the scenario is mirrored: \(<1, \ldots, cut \cdot l_2>, <cut \cdot l_2 + 1, \ldots, min(l_2, l_1)>\) and \(<l_1, \ldots, l_2>\), from the second, first and second parents, respectively. The genome of the first child is forced to be of the same size as that of the first parent, and the genome of the second child is forced to be of the same size as that of the second parent.

**Key observations**

The dynamic complex genome representation offer means of evolving the size of individual Petri nets, as well as their content. This can relieve experts from the, possibly, laborious task of manually deciding on maximum Petri net sizes. Moreover, different genome sizes may be suitable for different target concepts. When using genomes of static size, much elaboration might thus be needed for each target concept that is being evolved. Another important aspect is that the dynamic complex genome representation does not allow for any invalid solutions to be represented. Together, these two aspects may greatly decrease the number of generations that are required until promising solutions are found.

A potential disadvantage of the dynamic complex genome representation, as well as the complex genome representation, is however that they may impose too strong constraints on the learning problem in form of restrictive bias.

### 6.6 Chapter summary

This chapter has presented an approach for learning Petri nets for situation recognition through the use of genetic algorithms. Genetic algorithms should be well suited for this task since one of their main strengths is to search in complex spaces that may be hard to fully understand.

The choice of genetic representation is a challenge when using genetic algorithms for evolving complex structures, such as Petri nets. Three genetic representations have been discussed: bit genomes, complex genomes and dynamic complex genomes. They all have different properties, but it is believed that the latter may be used for more quickly finding promising solutions. We will return to this in Chapter 11, where the three representations are compared empirically.
Part III

Tools for evaluation
Chapter 7
A platform for working with situation recognition

This chapter addresses the first part of the second research objective, namely the issue of having tools for development and evaluation. The content of the chapter is largely based on one publication: Dahlbom et al. (accepted). The contributions of the chapter consist of:

- Design, architecture and implementation of a development environment for working with algorithms and representations for situation recognition (DESIRER\(^1\)). This development environment has also been released with a GNU General Lesser Public License version 3 and is publicly available at sourceforge\(^2\).

7.1 Motivation

As described in Chapter 4, situation recognition is concerned with recognising a priori defined situations of interest in a continuous flow of data and information. Moreover, interesting situations can be of both concurrent and temporal nature. To further delimit the problem investigated in the thesis, time constraints are also put on the problem. The thesis asks two research questions:

1. Can Petri nets be used for recognising situations as efficiently as rule based approaches using the Rete algorithm with extensions for explicitly modelling temporal constraints?

2. Can genetic algorithms be used to successfully learn Petri net based situation templates?

\(^1\)Development Environment for Situation REcognition Research
\(^2\)DESIRER is available for download at http://sourceforge.net/projects/desirer/
7.1.1 Towards answering research question 1

Although Petri nets already have been used for recognition, it was argued in Chapter 5 that existing approaches have some limitations. Hence, an extended technique addressing these limitations was suggested. Moreover, one extension to the technique was also suggested. The benefit of this extension is hypothesised to be potential gains in efficiency on the recognition procedure, at the expense of memory. Although worst case computational requirements can be derived theoretically, average costs are hard to estimate since they are highly dependent on the relation between templates of interesting situations and the stream of events that is processed. There is therefore a need for carrying out empirical investigations. It can be hypothesised that Petri nets are as efficient as rule based techniques and it can be hypothesised that the suggested extension does provide benefits. These hypotheses however need to be investigated using quantitative experiments and sufficient quantities of data. We shall return to the topic of data in Chapter 8 and Chapter 9. Having access to data alone is however not enough. There is also a need for having solid development and testing capabilities in which specific algorithms easily can be developed, evaluated and compared. This leads to the development of an environment that easily can be used to implement and investigate algorithms and representations for situation recognition.

7.1.2 Towards answering research question 2

It is well known that genetic algorithms can be used on a wide range of problems. Hence, it can be hypothesised that it can be used for learning Petri net situation templates too. This is, however, only a hypothesis, and as such, it requires testing. The task of learning Petri nets for situation recognition involves learning the graphical structure of Petri nets. It however also involves learning the contents of nodes consisting of constraints expressed with predicates. This is a complex learning task. Moreover, there are no exact training examples, but only limited amounts of feedback is available in existing data and the objective to optimise can be hard to specify. In Chapter 6 it was therefore argued that genetic algorithms should provide a suitable approach for learning Petri net situation templates. A potential problem with genetic algorithms is however that they can require much time for evolving promising individuals. Three different genome representations were therefore suggested. The suitability of these representations, with respect to the time that is required to evolve promising individuals and with respect to achievable performance of the resulting Petri nets, can only be hypothesised around. Thus, it is of interest to empirically compare the representations with each other at the task of recognising situations. Moreover, two approaches for possibly lowering the number of generations required to evolve promising individuals were also suggested. Similarly, in theory these approaches should lower the number of generations required to evolve fit in-
individuals. This however needs to be validated empirically. Lastly, a key issue when using genetic algorithms is the design of suitable fitness functions. Three potential fitness functions were discussed in Chapter 6. It needs to be determined empirically which of these that best suits the task of learning Petri nets for situation recognition. Hence, there is an interest in an environment that can be used for carrying out experiments.

7.1.3 Requirements and design goals

Both research questions leads to the construction of an environment that can be used for development of algorithms and representations for situation recognition, as well as for carrying out experiments that compare different solutions with each other. In Chapter 4, it was put forth that the performance of a solution to the situation recognition problem can be determined through the use of precision and recall (formally defined in Chapter 6). A requirement is therefore to be able to measure these properties. A main topic of concern is however also the computational complexity of solutions. It is therefore important to also inspect the time that is spent on recognition relative to the rate at which information is made available. Time consumption however also needs to be put in relation to the amount of memory that is required. After all, the amount of available memory is finite. For the topic of evaluating representations and approaches for using genetic algorithms, it is however also necessary to measure and compare fitness as well as evolutionary times. Being able to measure these properties is however not enough. Quantitative experiments are needed for providing reliable answers to questions and for reliably rejecting or accepting hypotheses with respect to algorithms, representations and input data.

In Chapter 4 it was argued that that situation recognition is not something that exists in isolation. It is part of a larger system for surveying the external world and for providing support to decision makers. The world is sensed and a number of objects are identified. Relations amongst these objects are inferred. These serve as the main input to situation recognition, which again provides output that can be used for other functions such as impact assessment or manual analysis. In order to work with situation recognition, and to develop and evaluate solutions, it can be important to also model parts of the surrounding components to some degree. This is the scope of DESIRER.

7.2 Desirer overview

DESIRER consists of two main components:

- A recognition platform library.
- A simple graphical user interface (GUI) for conducting experiments.
The GUI application simply serves as a host application which uses the library. It is thus largely dependent on the library. The library does however not have any dependencies towards the GUI, and can therefore be used as a stand alone library in other host applications. Figure 7.1 illustrates an abstract view of DESIRER, the library component and the GUI component, their dependencies, as well as an abstract categorical view of their subcomponents.

![Diagram of DESIRER components](image)

**Figure 7.1:** Overview illustration of DESIRER.

The GUI of DESIRER has been developed using Microsoft Visual C++ and its managed extensions for Windows. It thus makes use of the Microsoft .NET framework. The reason for using the .NET framework is that it allows for more rapid development of GUIs through the use of a wide range of already existing components. The library component has however been implemented in native C++ in order to minimise dependencies towards external components, and in order to explicitly manage the lifetime of data types related to recognition. The two components will now be discussed in more detail, starting with the library.

### 7.3 Framework library

The recognition platform library has been developed using native C++ to minimise dependencies and in order to explicitly manage the lifetime of objects. It consists of components that are interconnected through the use of a set of producer-consumer and publisher-subscriber patterns. In order to use the library, a host application needs to attach all necessary components to each other and then start feeding the framework with observations, events or facts. Before discussing the basic interfaces that can be implemented, a few fundamental data types that are useful when working with situation recognition will be discussed.

#### 7.3.1 Fundamental data types

The basic data types should serve as the core primitives for representing knowledge about the world, and can be used for recognising situations. Although situation recognition is depicted to operate on a symbolic level, using relations as primary concept, it is still intended to reside on top of systems based on classical
observation based data processing (tracks from radars and other sensing platforms, AIS data, etc.). This leads to the first two core data types: Observation and Entity. An Observation denotes properties (position, velocity) of an individual Entity (unique object, if uniqueness is an option).

As argued by Lambert (2003b), we want to be able to express more than measurable properties of objects when progressing through the data fusion process. We move into the domain of symbols and more specifically towards relations. A Relation can be described as a predicate in first order logic and it thus consist of a symbolic relation (name) together with a tuple of objects. Unary predicates denote properties of single objects, whilst binary and higher order predicates denote relations amongst objects. For asymmetrical relations the concept of tuple works nicely. Many interesting relations are however symmetrical (e.g. proximity between two objects). This could be solved by creating relations for every unique tuple of objects in a set of objects. Another option is to use a data type that can be used for expressing both symmetrical and asymmetrical relations without having to insert every combination of tuples. In DESIRER, this is modelled with an EntitySet, which can be both a set and a tuple, depending on a Boolean flag.

Recall, an event denotes the insertion or removal of a fact from the system. An event is in DESIRER termed an EventAtom\(^3\), and it consists of a relation, a Boolean value and a point in time when the event was created. A Fact is some piece of knowledge that can be used for reasoning. For example, all assertions in a rule based system are facts. Facts are usually seen as residing in some form of knowledge base and are often considered timeless due to their status as facts. When working with situation recognition, some facts are however only valid during a specific interval of time. This leads to the final concept: TimeFrame. A time frame depicts an interval of time and implements most of Allen’s temporal relations (Allen, 1983). A similar concept is for example used in (Matheus et al., 2003). Time frames are however also allowed to have both their start and end times open (unspecified). A fact that has its end time open is presently valid and a fact that has both its ends open is valid now, and always has been. Figure 7.2 shows an UML diagram of the basic data types in DESIRER.

![Figure 7.2: UML diagram illustrating the core data types in the library.](image)

\(^3\)The name event is often a reserved word in many programming languages.
7.3.2 Observation processing

Situation recognition is depicted as a function residing on top of observation based processing. Although higher level reasoning could be isolated from lower level processing when carrying out research and development, this would be rather counter productive since most data that is available consist of tracks, either produced by real world sensors, or produced by simulators simulating the real world to some extent. It can therefore be important to be able to also process observations in a system for working with situation recognition.

Observation processing has in DESIRER been implemented through three interfaces and two design patterns. First we have an ObservationReader (implemented by for example an ObservationFileReader). An observation reader is a producer of observations, which in turn should be consumed by an ObservationConsumer. This forms a producer-consumer pattern. Secondly, we have an ObservationDistributor (which implements the observation consumer interface). An observation distributor serves as a publisher of observations which can be subscribed to by other components. Any component interested in observations should thus implement the ObservationListener interface and subscribe to observations from a given distributor. This forms a publisher-subscriber pattern. The design of the observation processing subsystem is illustrated in Figure 7.3.

![UML diagram of the observation processing subsystem in DESIRER.](image)

7.3.3 Event processing

Now that we have the core data types and an observation processing system in place, we can start processing symbolic information. A system is depicted where specific symbolic relations between tuples of objects are grounded in observations provided by various sensing platforms. Naturally, there may also be other sources of symbolic information that can be used. A number of observation analysers are however depicted. These analysers implement the ObservationListener interface, and in turn produce events that should be
consumed by an event consumer. This again forms a producer-consumer pattern, but now at a higher level. The EventConsumer is implemented using an EventDistributor, which serves as a publisher of events from different sources. Components that are interested in receiving events should thus implement theEventListener interface and register its subscription to an event distributor. Again, this results in a publisher-subscriber pattern being used as foundation in the framework. Additional components that have been constructed in this subsystem includes a TickAnalyzer (used to distribute time information at certain intervals with respect to processed data and information), an EventReader (reads events from files) and an AnalyzerBase, which serves as an abstract base class for future observation analysers. Figure 7.4 illustrates the event processing subsystem.

As can be seen in the figure, there are currently three observation analysers: CloseAnalyzer, InterceptAnalyzer, and ApproachAnalyzer. These three classes implement the observation listener interface and the analyser base class. We shall return to their exact definition in Chapter 9.

### 7.3.4 Event usage

At this stage, observations and events are circulating in the system through the use of a number of producer-consumer and publisher-subscriber patterns. In order to provide a general basis for implementing situation recognisers, an abstract SituationRecognizer base class has been constructed. This class implements the event listener interface, and is in turn inherited by a PetriNetRecognizer and a RuleBasedRecognizer. Additionally, there is also an EventLogger class. This can be used for generating files consisting of events. Last but not least, a FactExtractor class has also been con-
structured. The FactExtractor implements the event listener interface and it serves the purpose of updating a KnowledgeBase consisting of facts. The knowledge base can in turn be used by prospective solutions to the situation recognition problem. At present, it is however not used by any of the existing situation recognisers. Figure 7.5 illustrates the components that are used for complex processing of events.

![Figure 7.5: UML diagram illustrating subsystems in DESIRER that are using events as their foundation.](image)

### 7.3.5 Situation recognition in DESIRER

The main purpose of this thesis is to carry out investigations with respect to Petri net based situation recognition. Hence, Chapter 5 presented a Petri net based technique for situation recognition, which addresses some limitations of previous approaches. The technique is an extension to work by (Ghanem et al., 2004; Lavee et al., 2007; Castel et al., 1996; Perše et al., 2008). In order to investigate this technique, it is necessary to implement it into the development environment.

**Petri net based situation recognition**

In the Petri net based technique, transitions are used as consumers of external events. Each transition is thus assigned a relation over a set of variables for objects. Places are simply used as containers for tokens, which in the technique represent partial matches between the flow of events and the modelled situation. A partial match is a template, and thus contains a set of variables for objects and a set of constraints. The Petri net based technique for situation recognition has been implemented into DESIRER and the components used for achieving this are illustrated in Figure 7.6.
A PetriNetRecognizer implements the abstract situation recogniser. It uses a PetriNetBuilder to load PetriNet objects from file. Multiple Petri nets may be loaded by a single recogniser. A Petri net consists of a set of places and transitions. Each Transition either operates directly on the content of places, or it operates using a PreCombiner in accordance with Section 5.6. Partial match combinations, PMCombination, are used by pre combiners and transitions, to iteratively construct valid combinations of partial matches.

The Rete algorithm with temporal extensions

The first research question in this thesis requires that the Petri net based technique is compared with a rule based technique using the Rete algorithm (Forgy, 1982) with extensions for explicitly using temporal constraints in rules (Walzer, 2009). Hence, there is also a need to implement a rule based technique into the environment. The implementation presented here differs slightly from that which is presented in Walzer (2009), in order to better fit into the development environment. The implementation of the rule based technique is presented in Figure 7.7.

A RuleBasedRecognizer has a ReteNetwork and a number of complex rules that are loaded from file using a RuleBuilder. A ComplexRule is represented as a parse tree which is used for compiling the ReteNetwork. Multiple complex rules may be compiled into the same network. Two basic node types are used: ReteAlphaNode and ReteBetaNode. Alpha nodes are used for constructing the pattern network (see Chapter 3) and the mode of operation for alpha nodes is to process events. Each alpha node has a single input. A ReteRootNode is used as the basis in the matching procedure. It
7. A PLATFORM FOR WORKING WITH SITUATION RECOGNITION

Figure 7.7: UML diagram illustrating the components of the rule based situation recogniser as implemented in DESIRER.

inherits from the alpha node and simply acts as a distributor of events to alpha constraint nodes. A ReteConstraintNode also inherits from the alpha node and it is associated with a constraint. Events that match the constraint are passed along to consecutive nodes. A ReteAdapterNode is located at the end of each path in the pattern network. An adapter node simply constructs working memory elements from events and stores them in a ReteOutputMemory. A ReteWME represents a partial match between a rule and the stream of events and its functionality is similar that of partial matches in the Petri net technique (it thus represents a template $T = (X, C')$ and has an associated time interval).

The join network is located after the pattern network, and it is constructed from beta nodes. The mode of operation for beta nodes is to process working memory elements. Each beta node is associated with an output memory for storing successfully joined partial matches. Beta nodes have two inputs where each input is associated with the output memory of a preceding node. Three types of beta nodes are used: join nodes, temporal nodes and terminal nodes. A ReteJoinNode joins two paths and follows the regular Rete procedure (Forgy, 1982). A ReteTemporalNode also joins two paths, but through the use of temporal relations between working memory elements. The mode of operation for temporal nodes follows the extensions suggested by Walzer (2009). Lastly, every path from a specific rule leads to a ReteTerminalNode. Working memory elements reaching a terminal node have been fully matched and can again be distributed back to the event subsystem as recognised complex events, or as termed in the present thesis, as recognised situations.
7.3.6 Genetic algorithms in DESIRER

The second research question in this thesis involves investigating if genetic algorithms can be used for learning Petri nets. Naturally, the development environment also has capabilities for using genetic algorithms. Figure 7.8 illustrates the components in DESIRER that are involved with genetic algorithms.

![UML diagram illustrating components in DESIRER that are related to genetic algorithms for evolving Petri nets.](image)

A PetriNetEvolver is used for controlling the evolution of Petri nets for situation recognition. It implements the event listener interface and distributes events to every Individual in a population of candidate solutions. The evolver also has a TargetConcept which describes the concept that is being evolved. Each individual in a population has a Genome from which a Petri net is constructed through the use a GenomeTranslator. In accordance with the discussion in Chapter 6, there are three genome representations. The abstract genome translator class is inherited by three subclasses, one for each genome representation. An instance of PetriNetBuilderBG is used for constructing bit genomes, an instance of PetriNetBuilderCG is used for constructing complex genomes and an instance of PetriNetBuilderDCG is used for constructing dynamic complex genomes. A suitable genome translator needs to be supplied by the invoker of the evolutionary process.

Each genome in a population of individuals consists of a number of chromosomes where each Chromosome consists of a number of genes. Genes are
implemented using an abstract Gene class, which is implemented as four sub-classes. An ArcGene represents an edge in a Petri net and is used in the dynamic complex representation. A TransitionGene represents a transition and is used by the complex and dynamic complex representations. A PlaceGene represents a place and is also used by the complex and dynamic complex representations. Finally, a BitGene represents a single Boolean value. This type of gene is used by all three types of representations. The PetriNetEvolver implements a number of selection, combination and mutation schemes, in agreement with the discussion in Chapter 6.

7.4 Host application

As previously discussed, the graphical user interface (GUI) has been implemented using Microsoft’s managed extensions to C++ together with the .NET framework. Besides a number of GUI components, the tool consists of a set of components for carrying out experiments.

7.4.1 Experimentation

The components for carrying out experiments act as proxies for native objects in the library, and are also responsible for connecting components of the framework, from observation input to recognition algorithms. Figure 7.9 illustrates the main components concerned with carrying out experiments.

There is an Experiment base class which runs as a background worker. The experiment class contains an InputParser and a Framework, of which the framework instantiates the consumer-producer and publisher-subscriber patterns. The input parser contains a number of ExperimentInput objects, where each is instantiated by either an ObservationInput object or by an EventInput object. These are used to read and parse observations and events, respectively. There are presently two experiment realisations: RecognitionExperiment and EvolutionExperiment. The evolutionary experiment is used for carrying out experiments using genetic algorithms, and it consist of an EvolutionarySettings object for setting up experiments, and a GenomeTranslator for instantiating genome types during evolution. The evolutionary experiment instantiates a Petri net evolver (see Section 7.3.6) and connects it to the framework, which feeds it with events. For more details about the evolutionary parts of DESIRER, see Chapter 6 and Section 7.3.6.

The recognition experiment contains a number of recognisers (contained in a RecognizerSet), where each implements the event listener interface. A set of recognisers can be compared with each other through the use of three experiment output components: RecognizerStats, which measures time and memory demands with respect to events, RecognitionPerformance, which determines precision and recall, and RecognizerTM, which measures time and memory on the whole task. In addition to these forms of output, there is also
7.5 Chapter summary

This chapter has addressed the first part of the third research question in this thesis. A development environment for situation recognition research has been
presented. Not only does DESIRER allow for promising algorithms to be de-
veloped, but it also allows for algorithms and representations to be compared
with each other. This is a key aspect for answering the research questions of this
thesis. The environment makes use of a number of design patterns that are con-
nected with each other to form a bond from observations to situation recogni-
tion. Observations are used as input. These are analysed to extract events with
respect to relations. Events are used for recognising situations. Situations are
complex events which when recognised can be reinserted into the framework
to increase the level of abstraction.

Two techniques for recognition have been implemented: a Petri net based
technique (presented in Chapter 5) and a rule based technique based on the Rete
algorithm with extensions for using explicit temporal constraints (based on
Forgy (1982) and Walzer (2009)). In addition, a foundation for evolving Petri
nets with genetic algorithms has also been implemented. In the next chapter we
shall look at a simulator that can be used for constructing data that can be used
as input to the recognition framework.
Chapter 8
A simulator for situation recognition research

In this chapter we shall address the second part of the third research objective of this thesis, namely, the construction of tools that can be used for creating data that can be used for evaluation. The content of the chapter is largely based on work presented in Dahlbom et al. (2009a), and the contribution of the chapter consists of:

- The design, architecture and implementation of a simulator for supporting research on situation recognition.

8.1 The need for data

As described in Chapter 4, situation recognition is concerned with recognising a priori defined situations of interest in a continuous flow of data and information. Interesting situations can, as argued, be of both concurrent and temporal nature. To further delimit the problem investigated in this thesis, real time constraints are also put on the problem. The interest is thus only related to real time situation recognition; in other words, an algorithm must be able to process information on at least the average rate at which is being made available.

In order to evaluate prospective approaches, and in order to compare different approaches with each other, there is a need for large amounts of data, as well as solid testing and evaluation capabilities. The availability of data can, however, be problematic within for example the military domain, as there is a lack of data, due to data being classified, kept secret, or for other reasons not being distributed throughout the research community. In case of real world data, there is yet another potential problem: data is often unlabelled (one does not know what it consist of).
In the *machine learning* community a central repository with data, the UCI repository\(^1\) (Asuncion and Newman, 2007), has been publicly available for many years. This has encouraged researchers to investigate new ideas for performing better than their competitors. As a result, many novel techniques have been conceived. Informally, it has been discussed that such a repository would be very beneficial to have in the information fusion community as well. Such a repository does however not exist yet, and instead the focus needs to be put on using simulated data. It has been recognised that modelling and simulation are efficient means for developing new technology (Nagi et al., 2006). Modelling and simulation thus seems to be a viable approach for constructing data that can be used to develop and compare algorithms and representations to the situation recognition problem.

There are many tools and simulators which can be used for building scenarios and simulating data in virtual worlds. Some examples include Stage\(^2\) and VR-Forces\(^3\). Some tools are publicly available, whilst others are commercially available for purchase on the market. Although it is not claimed that these simulators are flawed, they are very specific for certain purposes, of which conducting research on situation recognition is not one. Furthermore, many of these simulation environments often focus on resembling the real world as accurately as possible, which often results in very precise physical simulations. This is however not the main interest when investigating the situation recognition problem. It can often be assumed that the lower-level issues have already been taken care of (the implication of moving to a higher level of abstraction in the fusion model), and instead the focus needs to be on issues coupled to recognising complex patterns of activity. Modelling such activities often becomes a complex task in many of the available simulators, due to their very fine grained simulations, and this calls for the development of simulation environments that can be used for easily constructing data that can be used for working with the situation recognition problem.

This chapter elaborates on requirements that research on situation requirement puts on simulation tools. Moreover, these requirements have been used for constructing a component based simulation environment. In the development phase of this environment, the lead concepts have been that it should be flexible and easy to adapt, with a purpose to support experiments. Components are in the simulator used for adapting the environment to the needs of the research problem at hand.

---

\(^1\)The UCI repository is available at [http://archive.ics.uci.edu/ml/](http://archive.ics.uci.edu/ml/)

\(^2\)Stage is developed by Presagis, for more information visit [http://www.presagis.com/products_services/products/ms/simulation/stage/](http://www.presagis.com/products_services/products/ms/simulation/stage/)

\(^3\)VR-Forces is developed by MÄK Technologies, for more information visit [http://www.mak.com/products/vrforces.php](http://www.mak.com/products/vrforces.php)


8.2 Requirements analysis

The construction of simulators, for a research purposes or not, is a software engineering task. We need to carefully analyse what the purpose of the software is, what features it should have and which requirements that need to be fulfilled in order to achieve the features. When purpose, features, and requirements have been established, a design for the software needs to be constructed, which describes how it should be built and how all the pieces fit together. Pressman (2005) argues that the foundation of software engineering is the software engineering process, e.g. the process undertaken when constructing and maintaining software. The foundation of such a process can according to Pressman (2005) be captured in a generic process framework that includes the following activities: communication, planning, modelling, construction and deployment. For the work carried out in this thesis, however, not all aspects of software engineering are relevant. Instead, three important activities are addressed. These activities are concerned with the analysis and construction of a simulator for supporting research on situation recognition.

1. Requirements analysis. This falls into the framework activities of communication and modelling. The expected results from this activity are a clearly defined purpose of the simulator, a set of features and a set of requirements for achieving the features.

2. Architectural design. This falls into the framework activities of modelling and construction. The expected results from this activity are an architectural design and a set of designs of relevant modules.

3. Implementation. This mainly falls into the framework activity of construction. Modelling may however also be important since requirements change as a result of a changing world. The expected result from this activity is a first iteration of a piece of usable software.

8.2.1 Purpose

The expected results from an analysis of requirements is: (1) a clearly defined purpose of the software, (2) a well defined set of features that enables the purpose to be achieved and (3) a set of requirements that either achieves features, or which for other reasons can be of importance to include for achieving the purpose. In comparison with conventional software engineering however, there are not any external customers and stakeholders. The only customers and stakeholders involved are the participants of the specific research project. Nevertheless, there are many researchers in similar situations. Thus, the view of purpose, features and requirements provided in this thesis could potentially be interesting for improving, or easing, the work of other researchers in the future.
A divide and conquer strategy is employed to identify a set of top level requirements that are of importance. After this, a set of enabling technologies for fulfilling the requirements is presented. First however, the purpose of the software needs to be clearly defined.

**Purpose.** The purpose of the simulator is to provide data in support research on situation recognition and it should thus focus on relations between objects.

Situation recognition is concerned with identifying complex patterns of behaviour. Instead of focusing on properties of involved objects, relations serve as building blocks. For example, an interesting situation could consist of two objects that are close to each other, after which one of the objects is close to another object. This could at a high level be described as a double meeting. We have however not explicitly discussed properties of any of the involved objects. Rather, the focus is on relations between objects. It is however still a physical world of objects that is being simulated. It is thus desirable if the simulator covers a larger span, from relations to objects. Nevertheless, it can still be of importance to provide similar kinds of data as would be provided by real world system, i.e. track based data. The reason for this is that the algorithms and representations being investigated should be applicable in real world situations.

### 8.2.2 Requirements

In this section, the main requirements that can be put on a simulator for supporting research on situation recognition are derived. As previously mentioned, a divide and conquer strategy has been employed on the purpose of the simulator. Research often involves carrying out experiments. Experiments are typically used to accept or reject hypotheses (Berndtsson et al., 2002). The output of the simulator should thus be of such kind as to support experimentation on representations and algorithms for situation recognition, for quickly being able to discard or verify hypotheses that have been formulated. From this, a number of top level requirements can be formulated.

**In support of experimentation**

An important aspect of research on algorithms and representations is the topic of evaluation. This typically translates to conducting many experiments for establishing properties of the algorithms and representations that are investigated. Even though one, in the end, likely will be forced to assert that data which has been used for carrying out experiments is a representative sample of real world data, an initial requirement is that it at least should be possible to carry out evaluation using simulated data. The ability of constructing large amounts of data is thus required. It is however also a necessity that variation can be included in the data. As an example, if it is investigated if specific
types of situations can be detected, then it is not suitable to hard code situation instances into a simulation. After all, it is situation types that are being represented, and not situation instances. It is thus necessary to be able to implement variations in the situations that are being investigated. However, it is also of high importance to be able to repeat experiments that are carried out, either exactly or with variation. This leads to the first two requirements.

**Requirement 1.** It should be easy to construct large amounts of data supporting quantitative experiments, in which base line behaviours easily can be varied.

**Requirement 2.** It should be possible to repeat experiments, exactly or with some variation.

**In support of variation**

Situation recognition is a problem of importance within many application domains. For example, the interest could be in recognising specific situations within the stock market: are we on the verge of a bull or bear market. Another example could be to recognise situations in a retail setting: when are a set of customers exerting purchase patterns that can be exploited? In the case of this thesis however, the task is on recognising situation in the military and civilian surveillance domains, in which a number of sensors are used to detect the activities of a number of distinct objects. The purpose of situation recognition is in this setting to detect patterns of activity that include many different objects and their behaviours over time. Naturally, a simulator could be developed for the specific research cases to quickly make progress. However, as it so often turns out, specific cases may change. Instead, the ability of quickly changing focus is needed. Furthermore, in case promising results are achieved, it is also of great interest to investigate if these results are applicable in other application domains as well. This leads to the third requirement which reads.

**Requirement 3.** It should be possible to construct data for varying application domains.

As discussed in Chapter 4, it is not feasible to solve the situation recognition problem in a naïve manner. The focus thus needs to be put on algorithms and representations for intuitively managing the potentially large number of partial matches that may arise, or even, to make use of some form of heuristic. This often involves investigating the effects of different parameters in different situations. This in turn requires that the same simulation can be run many times, but with the possibility of easily varying specific parameters. From this, the fourth requirement is formulated as follows.

**Requirement 4.** It should be possible to vary specific parameters in a simulation.
8. A SIMULATOR FOR SITUATION RECOGNITION RESEARCH

In support of generating noise

Situation recognition involves finding specific patterns of activity within a flow of data and information. It can be interesting to see if it at all is possible to represent and find specific situations, but in the end, it necessary to investigate if it is possible to find interesting situations in large sets of data, where most data is irrelevant. This kind of data is often referred to as noise. In order to correctly carry out evaluation, the ability of modelling relevant noise is needed, e.g. noise that would typically arise in the kinds of situations that are being modelled. Relevant noise, in these settings, does not consist of random behaviours exerted by a set of objects. The objects that should represent noise need to exert relational behaviours that pose realistic problems. Noise is therefore needed at the relational level. The fifth requirement thus reads.

Requirement 5. It should be possible to easily construct varying types of relevant noise in simulations.

In support of rapid prototyping and extensibility

As already touched upon, variation is an important aspect in order to construct simulations that contain more or less realistic noise. This is true on a level higher than noise as well, at the scenario level. Although one initially might have a certain type of scenario in mind: two pedestrians meet after which one of them starts to run. This may turn out to not be exactly what was intended. It may be the case that the scenario needs to be changed to instead model that the runner, after having started to run, meets a third pedestrian. Being able to quickly construct scenarios in different ways is thus of importance, and this is formulated as the sixth requirement as follows.

Requirement 6. The simulator should support rapid prototyping of scenarios.

Some aspects might however not have been thought of in an initial phase of the development process. This can lead to a need for extending the simulator with new functionality. For example, assume that a good algorithm and representation has been developed for detecting potential pick pocket situations in a civilian surveillance setting. Moreover, assume that it in the initial experiments has been assumed that perfect sensor coverage exists. A single sensing model has thus been used which provides ground truth data. This is however not a realistic assumption, but it has served the development process well. Naturally, the next step would be to include more realistic sensing models in order to investigate how the suggested algorithms and representations behave in more realistic scenarios. From this, the seventh requirement is formulated.

Requirement 7. The simulator should be easily extensible with respect to new functionality.
8.2.3 Enabling technologies

This section discusses key features that are considered to be enablers for fulfilling the high level requirements that have been outlined.

Scripting

Scripting is an important feature in order to allow for rapid development of scenarios. It allows for development to be conducted in a fail safe manner. Moreover, flaws in scripts do not make software crash. Scripts are simply aborted when errors are encountered. Furthermore, the use of scripts also allows for behaviours and scenarios to be defined in a higher level language, with fewer syntactical constructs obscuring the view.

There are many scripting languages that could be used, or, a completely new scripting language could be constructed which perfectly fits the research needs. However, the benefits of using scripting technology that has been proven useful in simulation environments with high time constraints, outweighs possible benefits of constructing case specific languages. Lua⁴ is a light weight scripting language that has become very popular within for example the computer games industry in the past decade. Computer games share many aspects with the kind of simulator that is developed, and thus, it should be sufficient and suitable for aiding in achieving some of the requirements outlined in the previous section. Using a light weight scripting language can also be of high importance, since the intention is to simulate vast numbers of objects.

Scripting directly supports requirements 1, 4, 5, 6, and 7.

Component based architecture

At the highest level, it is suggested that an object oriented programming paradigm is exploited, since concepts developed within the object oriented paradigm over the years have proven to be beneficial. A potential problem when using an object oriented paradigm is the risk of getting caught in a classical hierarchy of objects. As an example, my Volvo is an instance of Volvo series V50, which is a car, which is a vehicle, which is an object, which is an entity in the universe. This simply entails too many problems. Properties also tend to move through the hierarchies as interpretations and designs change — “Ah, all cars have doors, let’s put that property in the car object. Wait, all vehicles have doors. Let’s put doors there” — to in the end result in a scenario where some classes tend to contain very much, whilst other classes simply define a name. Furthermore, too deep inheritance hierarchies often slow software down. Moreover, if the intention is to create a new type of object, for example, a car with a washing

---

⁴Lua is developed and maintained at PUC-Rio (the Pontifical Catholic University of Rio de Janeiro, Brazil) and is available for download at http://www.lua.org/
sink, then a new class would need to be created, even if both cars and washing sinks have already been defined.

Having a component based view on things can however alleviate for some of these problems\(^5\). For example, different types of components could be modelled: a component for engines, a component for doors and a component for wheels. Then simply by aggregating these components together, a specific type of object can be defined, which in turn can be instantiated. Thus, if a component for a washing sink have been developed, as well as all the other components of more traditional cars, then a washing sink can very easily be added to the car and voila.

Component based architecture is in direct support of requirements 1, 3, 5, 6, and 7.

**Message passing**

Message passing is traditionally one of the core concepts of the object oriented programming paradigm. Often it is however achieved through method call and return constructs. Situation recognition, as discussed here, often involves finding situations which preferably are described with relations. Objects, meet, they follow each other, they are transported by each other or they approach each other. In other words, many actions are coordinated amongst objects.

There is thus a need for objects to be able to easily communicate with each other. Communication could be achieved by letting objects invoke methods in each other, however, this would only allow for synchronous calling. This would also make objects become highly dependent on each other. Instead, it is argued that asynchronous invocation and explicit message passing is preferable. Moreover, on occasion, the invocation should perhaps be delayed by some amount of time, in order to better simulate delays that are present in the real world.

Explicit message passing is in direct support of requirements 1, 3, 5, 6, and 7.

**Time management**

Time is perhaps the most important aspect of simulation. Discrete steps are used to simulate a flow that in the real world would be continuous. It is thus of importance to have access to good time functionality with a high granularity. Furthermore, in the case of surveillance applications, the purpose is on simulating a physical world with objects that obey to the laws of physics (not too realistic to start with). It is thus of importance to be able to represent time deterministically. The computers that are used to run simulations are in theory deterministic. However, this is not the case in practise since many applications

\(^5\)It is not claimed that they are the solution to everything though.
can be running simultaneously, as well as services in the operating system. These external processes may cause clock cycles to be consumed differently from time to time, and this may in turn affect the simulation. It is therefore not enough to rely on the real-time clock present in the computer.

Having a well defined approach for time management is in direct support of requirements 1, 2, and 4.

**Random number generation**

Evaluation is an essential aspect of conducting research. It should be possible to run simulations that have a varied behaviour. However, it is also necessary that experiments are repeatable. On some occasions, only specific parameters needs to be varied, whilst other parameters need to be kept fixed. Thus, there is a need for having many different random number generators that can be used for controlling what is varied and what is not. A scenario developer should have the possibility of choosing from which generator to pick numbers. Furthermore, settings for each available generator should be easily controllable.

Having multiple controllable random number generators is in direct support of requirements 1, 2, 4, and 5.

## 8.3 Architectural design

The architecture that has been developed for the simulation environment basically consists of two main parts. First we have the simulator itself. This is the most important component. However, to enable scenarios to be easily developed, there is also a need for a scenario editor. The environment thus consists of an editor and a simulator that are integrated into a common framework. The editor is used to create scenarios which can be stored to disk. These can later be loaded by the simulator, which simulates the scenarios and provides output in form of data. An implication of this structure is that contents of a simulation cannot be changed while it is running. The main argument behind this is however in support of requirement two: it should be possible to repeat experiments. In case a user, or some external process, is allowed to manipulate aspects of a simulation while it is running, then it can be problematic to assure that the data that is being produced can be reproduced exactly.

### 8.3.1 Simulator design

The simulator component of the simulation environment basically follows a traditional object oriented paradigm. However, the core components involved in representing and manipulating objects have been implemented in a component based fashion, in agreement with the discussion in Section 8.2.3. The architecture of the simulator is illustrated in Figure 8.1.
As can be seen in Figure 8.1 the top level class is the WorldSimulator. This class consists of a SimView object and a SimEngine. The purpose of the view is to allow for a simulation to be visualised as it runs. SimEngine consists of a Timebase and a Simulation, where the time base is used for rendering purposes. The purpose of the Simulation is to represent everything in a simulation, and the purpose of the simulation engine is to control the simulation. A simulation has an associated World, which consists of a number of instances of WorldNode. The simulation also has access to a Timebase. This instance of the time base keeps track of time in the simulation. Furthermore, a simulation also contains an EventManager, which keeps track of telegrams that are passed around in a simulation. Message passing is thus implemented through the EventManager and through the use of instances of Telegram (a generic message container). Last but not least, the simulation also contains a number of random number generators (Randomizer). It can be explicitly chosen which random number generator to use in a scenario.

Lastly, there is a WorldNode class and a NodeComponent class. These constitute the core of the component based design. A node does not contain anything (almost, we shall return to this). It is simply an aggregate of components. A certain combination instead makes a specific node to what it is. Specific components can be constructed as subclasses of the NodeComponent class. The interface of nodes and components are similar. They contain an update method, a handle message method, a method for initialisation from XML and a method for generating XML. When any of these methods are invoked on a node, it invokes the corresponding method in its components.

Now, a component based utopia has not been achieved. The WorldNode class actually contains something on its own. In the specific research that is the focus of this thesis, military and civilian surveillance applications, most scenarios have a set of common properties. The aim is to simulate more or less realistic scenarios consisting of objects moving around in some world of inter-
est. Everything that is simulated thus has a position and representation in some world. Furthermore, as we shall see later, certain functionality will be called very often. A choice has therefore been made to put properties such as position, velocity and bound (not shown in the figure) directly in the WorldNode class. During the development of the simulator it was discovered that having these things as components simply slows simulations down too much.

8.3.2 Editor design

The purpose of the editor component of the simulation environment is to allow for easy construction of scenarios. Furthermore, it should also allow a scenario designer to easily manipulate specific parameters. The design of the editor revolves around a classical .NET form construct. Details of .NET will however not be described here\(^6\). Instead, the focus will be put on how the editor interfaces with objects in the simulator.

The editor is simply a GUI that is put into place for easily constructing scenarios. This requires that the objects and components used in a simulation can be managed using interfaces in the GUI. Allowing the GUI to know all aspects of the simulator is however not a good idea. This would lead to very high coupling and very low cohesion, thus violating the famous mantra in software engineering — low coupling and high cohesion. Coupling refers to the degree to which objects depend on each other and cohesion is concerned with how focused objects are. Neither is it an option to include GUI related aspects in the simulator objects. This too would result in unfocused objects implemented with unfocused code. Clean code leads to better productivity and reusable software, and an aspect of clean code is that it is focused (Martin, 2009). Focused objects are therefore preferable, meaning that the GUI should not have precise knowledge of objects, and objects should not have knowledge of the GUI.

A viable approach, however, is to use some form of design pattern (Gamma et al., 1995) that allows for access to the objects, without polluting code related to the objects, nor to code related to the GUI. In this particular case, it has been decided that adapters\(^7\) should be used to allow access to objects in the simulator environment. The adapters implement a few standard interfaces for variable retrieval and update. These interfaces can be used in a standardised fashion by the GUI components in the editor. An adapter object is thus created in the editor for each corresponding object in the simulator. The task for each adapter object then becomes to determine the parameters that are accessible in its associated simulator object. Selected parameters are exposed using the standard interfaces, which are implemented by the adapters. Adapters are thus used to access scenario elements in a standardised way, and the resulting architecture of the editor is illustrated in Figure 8.2.

\(^6\)More information about the Microsoft .NET framework is available at \url{http://www.microsoft.com/net/}

\(^7\)Thus using the adapter pattern (Gamma et al., 1995).
To the right in Figure 8.2, some of the components of the simulator are shown. To the left, key components for interfacing with the simulator are shown. There is a world editor which contains a world, which, as before, contains a number of world nodes that have components. The adapter for a world is called IWorld, the adapter for a world node is called an IWorldNode and the adapter for each respective component that exist in the simulator and which needs to be reachable from the GUI, would be called IComponentName. Two interface classes, IComponentInterface and IVariableInterface, have been designed to support a coherent way of accessing variables in the world, nodes and components. These interfaces are thus selectively implemented by the adapter classes.

In addition to these classes there is also an IComponentFactory, which dynamically constructs object adapters from objects (world, world nodes, or components). The IComponentInterface allows access to variable interfaces, which in turn allows access to single parameters (that have been exposed). Access to individual parameters is achieved through INodeVariable, which have links directly into the GUI (as an entry in an edit box or similar). IWorld and IWorldNode implement both interfaces, whereas IComponent only implements the variable interface.
8.4 Implementation

The simulation environment has been implemented using Visual Studio 2005. In order to allow for resources to be used and managed efficiently, native C++ has been used in the simulator. For the editor however, Microsoft’s managed .NET framework has been the choice, since it allows for easier development of GUIs and interfaces. It is very easy to build extensions to existing objects such as text boxes, list boxes and other Windows Forms components, which clearly is an advantage as time does not need to be spent on such aspects.

8.4.1 Scripting

As discussed in Section 8.2.3, Lua has been used as a basis for scripting. Figure 8.3 illustrates how the simulator has been extended to support scripting.

![Diagram](image.png)

**Figure 8.3:** Illustration of how the simulator has been extended with capabilities for scripting. New classes are highlighted.

As can be seen in Figure 8.3, four classes have been added. Scripting is a component that can be added to any node. The scripting component has a LuaState, which is owned by a LuaEngine. Each Scripting component thus has its own runtime environment with stack and heap. The LuaEngine initialises instances of LuaState, keeps track of their existence and makes sure they are properly removed. The final class is LuaExposed. The main purpose of including scripting is to allow for easily manipulating a simulation, but in a controlled environment. To achieve this, LuaExposed exposes relevant functionality of the simulator. It registers a number of C++ functions that can be called from Lua. These functions do in turn have access to other
components. As illustrated in Figure 8.3, select features of the WorldNode class, EventManager class and a class named ComponentX, has been exposed. ComponentX, in this case, simply represents that functionality has been exposed from many existing components. In case new components are built, then these would in turn need to register exposable functions to LuaExposed, in order to be accessible from the scripting environment. The purpose of exposing the EventManager class is to allow for scripted objects to communicate with components, either in their owning world node, or of other world nodes in a simulation.

8.4.2 Nodes, components, resources and allocation

An object consists of a node and a number of components in the component based view. Nonetheless, it can be important to be able to define a set of object types for easy creation and use. Unique classes could be constructed for each object type, which in turn would define which components the type should consist of. However, this would violate the extensibility theme, since new classes needs to be created for each new object type that is needed. Instead, object type definitions are preferably kept external to the simulator. It has been chosen that object types should be defined using XML. Templates for object types can thus be constructed using XML, and these templates can at runtime be loaded and used for instance creation. As an example, a template for an object type (in this case a vessel) could be defined as illustrated in Listing 8.1.

Listing 8.1: Example of a template for a node representing a vessel, defined in XML.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<nodetemplate target="Dynamic" name="Vessel1">
  <node name="Vessel1" bound="circle">
    <component type="shape" owner="true">
      <icon2d>
        <texture>
          <![CDATA[data\icons\vessel1.bmp]]>
        </texture>
      </icon2d>
    </component>
    <component type="momentum" mass="75" max_speed="13" owner="true" />
    <component type="steering" owner="true" />
    <component type="scripting" owner="true">
      <script update_method="update" update_time="5">
        <file>
          <![CDATA[vessel1.lua]]>
        </file>
      </script>
    </component>
    <component type="cargoholder" owner="true" />
  </node>
</nodetemplate>
```
Object instances can be stored and loaded in a similar fashion, but with more information being put in the XML file, such as for example position and velocity. This however requires interpretation. There is thus a need for a facility to convert XML into actual object instances, and to save object instances into XML. Such a facility requires dynamic allocation of nodes and components. Furthermore, in the initial work on the simulator, it was discovered that it is of high importance to properly manage the lifetime and access to individual nodes and their components. Typically, object allocation and deletion can be tricky, but manageable. The inclusion of scripting for easy construction of scenarios has however made it even trickier. In a scenario with a couple of hundred objects being updated 50 times or more per second, some objects are bound to be removed, whilst other objects try to communicate with them. To overcome these problems and to allow for easy creation of objects and components from XML, the basic architecture has been extended with three classes for management and creation of nodes and components. This is illustrated in Figure 8.4.

The world does no longer take ownership of individual nodes. Instead, a NodeManager owns every node and takes care of their creation and destruction. Moreover, nodes should not be accessed without assuring that they are still valid. This is carried out by retrieving instances from the node manager, using unique ids. Furthermore, a NodeFactory (not the factory pattern) has been created to constructs nodes from XML. Similarly, a ComponentFactory is used to construct components from XML. With these classes, management and construction of objects is easily extensible. It has however been decided that each component is responsible for it’s interpretation to and from XML, e.g. every node and component is required to implement its own conversion functionality to and from XML. The purpose of the factories is thus not to interpret XML, but rather, to manage dynamic allocation of classes.

---

8Nodes and components are not required to parse XML in text format, but in tree format.
8.4.3 Basic agent behaviours and collision detection

As already mentioned, the research cases that this thesis is focused on is concerned with situation recognition in the military and civilian surveillance domain. An environment is thus to be simulated, which should resemble objects moving around in a world. Furthermore, in addition to having objects that are able to move, they should move with a purpose. The aim is to close the gap to the relational description level. Moreover, many interesting situations consist of objects being close to each other, being at certain places, approaching each other, following each other, and other similar relations. To achieve this, the simulator has been extended with components that allow objects to actually move, move to other objects, steer, check for collisions and be notified when certain triggering conditions are true. Figure 8.5 illustrates the extended architecture.

![Diagram of extended simulator architecture](image)

**Figure 8.5:** Illustration of how the simulator design has been extended with components for physics, basic steering behaviours, collision detection through bounding volumes and a hierarchical scheme of spatial.

A Momentum component has been created which implements Newtonian physics for objects that should be able to move (the level of detail in the calculations is however quite coarse). In addition to this, a Steering component has also been constructed. This component implements a few basic steering behaviours, such as seeking, arriving, following, path following, fleeing, wandering and avoidance. The specific implementation details of these behaviours are based on the book on artificial intelligence for computer games by Buck-
8.4. IMPLEMENTATION

Moreover, a Trajectory component has also been created. This component simply contains a number of points that can be iterated and moved to in a cyclic fashion. The Trajectory component can however also consist of objects in the environment, thus movement along a dynamic pattern can be achieved. The final new component is a Trigger, which can receive notification when objects enter its associated area.

In order to test for collisions (an essential aspect), an abstract Bound class has been constructed. The Bound class is implemented using three basic types of bounding volumes (of which only one actually is a bound). BAll represents a bound that is considered to intersect with everything that it is tested against (we shall shortly see why). BEmpty represents a bound that is considered to not intersect with anything (some objects do simply not “want” to be tested for intersection). Finally, a bounding sphere, BSphere, has been constructed for carrying out simple and efficient intersection tests.

Performing collision detection efficiently is a problem in its own right. In this simulator the choice has however been made to construct a simple bounding volumes hierarchy. Dynamic management of this hierarchy is carried out using a CellSpacePartition object, which has a number of associated SpatialNode objects. The CellSpacePartition and all SpatialNode objects are owned, constructed and destroyed by the World object. Every spatial node has an associated Bound, which in turn belongs to an object. The reason for having an instance of Bound in each node object is performance related. An option would be to have the bound as a component in its own right; however, this would mean that dynamic type checking is required for carrying out collision detection. This is simply not feasible in a simulator that is intended to simulate many objects, efficiently. However, components may also have bounds and spatial representations. As an example, the Trigger component can have its own spatial representation in order to created extended bounds that are different from the actual bound of the node that it belongs to. This is useful when for example modelling an object that has an inner bound (something bumps into it), as well as an outer bound (an object is able to sense everything within a 500km radius). When carrying out collision detection, BAll is used as the universe (it thus contains everything per its definition).

The final addition in the architecture is a PhysicsEngine. The purpose of this class is to coordinate updates of nodes and components, to carry out collision detection and triggering, and to produce notifications for such events. Lastly, the Steering component has access to the CellSpacePartition object, and the reason for this is that objects should have access to collision avoidance behaviours.

9For more complex crowding behaviours, see for example (Qiu and Hu, 2010).
10In this specific case a small quad tree representation is used for achieving some efficiency.
8.5 Example usage: smuggling scenario

In this section an example maritime smuggling scenario is implemented in the simulator for demonstrative purposes. In order to implement the scenario, however, new functionality is required. Consequently, the core of the simulator is extended.

8.5.1 Scenario description

As argued by the European Security Research Advisory Board (2006), smuggling of drugs and weapons, and trafficking, pose large problems within Europe. Furthermore, there are many km’s of coastlines in Europe which needs to be surveyed to find some of these situations, e.g. smuggling and trafficking situations. Numerous vessels constantly move along the coastlines, of which freighters, tankers, sailing boats and ferries are some examples. Automatic Identification Systems (AIS), radars and other surveillance equipment continuously supply tracks of individual objects that are detected. The problem thus becomes to detect objects that possibly are carrying out smuggling operations. This can possibly be achieved by analysing the tracks of individual objects to extract relational information, which in turn can be used for detecting complex situations with multiple participating objects. One type of smuggling situation that could be interesting to recognise is depicted in Figure 8.6.

![Illustration of an example smuggling scenario.](image)

The scenario depicted in Figure 8.6 proceeds as follows: (a) a larger vessel moves along the coast, (b) a smaller speed boat which heads towards the coast is deployed and another speed boat leaves the shores and heads out to sea, (c) the two speed boats meet and exchange goods and money and (d) the two boats return to where they came from, one boat returns to the larger vessel, and the other heads back to shore. To end the situation, the smaller speed boat is picked up by the larger vessel.

---

11Of course, smuggling situations may take on many forms, and this is simply an illustrative example depicting one type of situation that could be interesting to recognise.
8.5.2 Component construction

The simulator can however not be directly used for constructing the scenario. The reason for this is that the scenario contains aspects that require functionality that has not been thought of in advance. Luckily, the simulator can be extended to include components necessary for modelling the depicted scenario.

There are a few key functions that need to be constructed. First, the core of the simulator does not have any support for creating data. Preferably, data should be produced in such a way as to resemble real world data, e.g. tracks of objects. Some form of sensing capability thus needs to be included. Secondly, it can be important to anchor the scenario to some form of background, may it be a realistic charts database, or simply some pictorial representation of a stretch of coast. Thirdly, it is necessary to be able to dynamically model vessels in various areas, and to be able to make them move along some set of predefined routes. Finally, the interesting situation depicts a smaller boat being deployed and picked up by a larger vessel. The simulator has been extended to address these issues. The extensions are illustrated in Figure 8.7.

![Diagram](image)

**Figure 8.7:** Illustration of how the simulator design has been extended with components for modelling a maritime smuggling scenario.

A *SimpleRadar* component has been constructed to simulate some form of sensor. Its purpose is to store tracks of all active objects within its coverage area at a fixed frequency. At present, the component provides ground truth data for all objects within a circular area surrounding it. In order to anchor a scenario to some relevant background, a *Geometry* object has been created. This is a generic container for 3D surface consisting of any number of joined trian-

---

12As it has been shown, luck did not have anything to do with it.
gles. It is possible to assign textures, images and other visual properties to a surface object. A FlatTerrain component has been constructed to represent a simple patch of flat of terrain with an associated texture. It thus has a geometry object, and this enables a scenario to be anchored to some background. The geometry can also be used for carrying out intersection tests, thus keeping objects on the ground. Next, a Shape object has been constructed. This is an abstract base class for representing 2D shapes. Subclasses include Circle, Rectangle and Icon. Circles and rectangles can be used for representing areas of interests. The purpose of the Icon object is to allow for object types to be distinguishable when designing and inspecting simulations. Also constructed is a Point object. These can be used as static way points. A ShapeComponent has also been constructed to contain a set of shapes.

Two things can be noted: (1) geometry and shapes are not constructed as components, but as objects and (2) geometry and shapes are owned by various components, but they are also shared with the owning node which has direct access to them. Both of these choices are based on performance considerations.

The last two components that have been constructed are CargoHolder and Cargo. Cargo holders can contain nodes that have a cargo component, and nodes that have a cargo component can be loaded onto objects that have a cargo holder component. When cargo is put into a cargo holder, the node acting as cargo is inactivated. Similarly, when a cargo component is unloaded, the node acting as cargo will again be activated. Inactivated objects are not tested for intersection, they are not rendered and they cannot be sensed. They are however still updated and they are allowed to communicate.

8.5.3 Scenario construction

The scenario is depicted to take place along the eastern coast of Sweden and more specifically in a small area to the south. A conceptual design has been constructed and it consists of seven circular areas off the coast. These areas are to be used for constructing varying behaviours. Areas 1 and 5 have an associated script which is called once every five seconds. When invoked, the script generates a random number from a uniform distribution. In 30% of the cases a new vessel will be created. The initial position of the vessel is chosen randomly from within the area that it was created in. A trajectory is then created for the vessel, consisting of random points from within areas 2, 3 and 4 (4, 3 and 2 if it was created in area 5), to finally have a random destination selected at the opposite side. The speed of vessels is randomised between 5 and 15 knots.

The interesting situation is modelled by using a smuggling centre object, which has an associated script that is invoked every 6 seconds. When invoked, the script first checks so that a smuggling situation is not already in progress. If not, it generates a random number, and in 30% of the cases a smuggling vessel and a speed boat are created. The speed boat has a Cargo component
and is loaded onto the vessel which has a \texttt{CargoHolder} component. The vessel has its initial position set randomly within areas 1 or 5, after which the same procedure as for normal vessels is used for constructing the behaviour of the vessel. However, a script associated with the smuggling vessel requests to receive triggering messages when entering area 3. Once the trigger message is received, the speed boat is deployed. Once deployed, the speed boat is set to intercept its close to shore counter part, which in turn, is set to meet the offshore speed boat. When they have intercepted each other, they return to where they came from, i.e. the smuggler speed boat returns to the mother ship, and the close to shore speed boat resumes its previous behaviour. The speed of both speed boats is initially set to 25 knots.

Initially, the close to shore boat has a very simple behaviour. It is statically created within area 6 and it has an associated script that is invoked by a trigger that is associated to each of the areas 6 and 7. When invoked, it sets a new destination in the opposite area. This behaviour continues until it is ordered to meet the smuggler. Since the purpose of this scenario has been to demonstrate the simulator, data generation using the \texttt{SimpleRadar} has not been used. This could however be achieved very easily by constructing an object that has a simple radar component, which in turn has a sensing radius, a target output file and a sensing frequency. Figure 8.8 illustrates the scenario being edited, and the resulting smuggling scenario is illustrated in action, in Figure 8.9.

![Figure 8.8: Illustration of the example maritime smuggling scenario when being edited.](image-url)
8.6 Chapter summary

In this chapter, the second part of the third research objective of this thesis has been addressed. The design, architecture and implementation of a simulator for supporting research on situation recognition have been presented. This simulator should allow for quick and easy construction of a wide range of scenarios that can be of relevance. The simulator is extensible, thus allowing for new scenarios and functionality to be incorporated easily. For example, if it is interesting to have access to 3D terrain information, which can be used for realistic terrain-following, then three components would be needed: a terrain interface component, a terrain geometry object with intersection test capabilities, and thirdly, a navigational component. Besides an operational simulator, a contribution of this chapter consists of an analysis and reasoning with respect to the design and architecture of the simulator. This may be of value to other researchers, or developers, who face similar problems.

In the next chapter, two case scenarios for carrying out empirical investigations are presented. The first scenario is an artificial pick pocket scenario, and consequently, it has been implemented in the simulator described in this chapter. The second scenario is a maritime pilot boat scenario. This scenario plays out in reality, and the data that is used thus come from real world sources.
Chapter 9
Scenarios for evaluation

This chapter addresses the third research objective. The content is based on material appearing in all publications, and the contributions consist of:

- Design, implementation and usage of a simulated pick pocket scenario.
- Usage of real AIS data for detecting situations in maritime traffic.

9.1 Pick-pocket scenario

Pick pocketing is a problematic crime relevant in most parts of the world. It affects many people and the number of resolved crimes is very low. For example, in Sweden there were 432 reported pick pocket crimes per 100,000 citizens in 2009, or around 40,000 in total. Of these, 5% were dismissed for various reasons, and only in around 1.4% of the cases, a suspect could be associated with the crime\(^1\). Crimes such as stealing from a bag might also have been included in the statistics for pick pocket thefts (Swedish National Council for Crime Prevention (Brå), 2010). Nonetheless, some of these pick pocket thefts have occurred in largely crowded areas such as airports, train stations and shopping malls, where large numbers of people are located.

The use of closed circuit television (CCTV) cameras in public areas has become increasingly popular in the past decade (Welsh and Farrington, 2007). Such systems may for example be used to prevent crime, to identify culprits or as evidence in criminal prosecutions. These systems do not usually have capabilities for automatically identifying interesting behaviours, but are instead used by humans for detection and analysis. Even rarer, is the use of such equipment for the task of real time recognition of complex patterns. Much research has however been carried out in the past decades on tracking objects in streams of video data. Naturally, a next step would be to also use these tracks of objects in higher level systems for detecting complex patterns of interest.

\(^1\)These number have been extracted from statistics collected by the Swedish National Council for Crime Prevention (Brå) (2010)
9. SCENARIOS FOR EVALUATION

9.1.1 Scenario description

A pick pocket scenario has been implemented in the simulator described in Chapter 8. The scenario is located in a shopping zone (could be a street, a mall or something similar), where many pedestrians move around on their daily business: approaching each other, meeting each other, doing some shopping or just seemingly moving around at random (although most will probably have some purpose). Interesting situations may occur in this flow of information about pedestrians, and these need to be recognised. More specifically, the objective is to recognise pick pocket situations. Consider two thieves, a primary thief and an accomplice. The primary thief scouts for a suitable victim, moves close to the victim and then attempts to perform the theft (stealing a wallet, watch, or similar). After this, the primary thief and the accomplice intercept each other and meet to hand the stolen goods over. Figure 9.1 illustrates the pick-pocket situation.

Figure 9.1: Illustration of the pick pocket scenario. In figure (a) the thieves are passively moving around in the environment and the primary thief is searching for a potential victim. In figure (b) the primary thief has located a victim and has started moving towards it. Figure (c) depicts how the thief bumps into the victim, thereby taking the goods. In figure (d), the two thieves move towards each other in order to hand the stolen goods over. Figure (e) illustrates how the two thieves meet each other, thus completing the hand over. Finally, figure (f) illustrates how the two thieves move away from each other, possibly leaving the scenario or carrying out some other activity of interest.

Now, assume that there are a number of cameras located throughout the environment. Also assume that these cameras can provide accurate tracks of how people move. It is on top of these tracks that an abstract state space can be defined. The abstract state space can in turn be used for defining high level interesting activities; templates of interesting situations. During runtime, the task becomes to extract relational information from tracks, push this information

\footnote{Someone who exerts behaviour that thieves can use to classify the victim as a suitable victim.}
to high level recognition capabilities, which in turn can use the information for identifying if any instances of defined situation types possibly are occurring.

### 9.1.2 Scenario construction

Two spawn/despawn zones are modelled to achieve a dynamic flow of pedestrians. These are used for creating and destroying pedestrians. Each zone is modelled using a script that is executed every five seconds. Upon execution, a script creates new pedestrians according to a uniform distribution. It is however not enough to have pedestrians that move straight through the environment since their behaviours needs to be of varying nature. Thus, four shops are modelled in the scenario. The shops are also modelled using scripts that are executed every five seconds, however, these script distribute advertisements with a purpose of attracting pedestrians. Upon execution, advertisement messages are sent to every pedestrian in the scenario. Upon receipt of an advertisement message, a pedestrian picks a random number, termed the attraction value, which is added to previous attraction values for the sender of the message. Each pedestrian thus keeps a table with attraction values for each shop in the scenario. When the attraction value for a specific shop becomes greater than some predefined limit, a pedestrian may stop its current activity and instead head towards the shop. This however depends on the type of pedestrian that is modelled.

Three types of pedestrians are modelled: normal shoppers, determined shoppers and purposeful pedestrians. Normal shoppers start in one of the spawn zones and head for the other. While moving they have the ability of visiting multiple stores. Purposeful shoppers start in one of the zones and head for exactly one shop, after which they return to the zone in which they started. Purposeful pedestrians simply move through the environment from one zone to the other (randomly chosen). When a pedestrian is created, there is a 10% chance that it will be a determined shopper. The determined shoppers wait in their spawn zones until attracted by a shop. For the remaining 90% of the created pedestrians a random value, termed will to shop, is drawn from a uniform distribution. As long as the will to shop for a pedestrian is above 65, the pedestrian may be attracted by shops. Initially there is thus a 35% chance that a normal shopper is at all willing to shop. If not in these 35%, the shopper is a purposeful pedestrian. When the will to shop of a pedestrian drops below 65, the pedestrian heads for its exit zone (randomly chosen). A pedestrian that is attracted by a shop heads straight for the shop. Once shopping is complete, the attraction value for the particular shop is reset and the will to shop is decreased by a random number in the range 0-50. Lastly, every pedestrian has a 60% chance of being a suitable victim of a pick pocket theft. The speed of suitable victims is set randomly from 2.5 and 3.5 m/s, and the speed for unsuitable victims is set randomly between 1.5 and 3.5 m/s. At points where pedestrians change behaviour (heads to a shop, done shopping), their speeds are randomly reset. All pedestrians also actively try to avoid colliding with each other. All
variables have been tuned in collaboration with experts from our industrial partner Saab AB, to create a somewhat realistically looking scenario.

On top of this model of normal behaviour, a number of pick pocket situations are instantiated in each simulation. The initiation of pick pocket situations is handled by a script that is executed every five seconds. In case there presently is no pick pocket situation in progress, then there is upon execution of the script a 30% chance that one is created. Upon creation, two thieves are allocated randomly in the two spawn zones. One of the thieves is selected to be primary thief and the other is set to be the accomplice. Both thieves head towards the centre of the scenario, and after 15 seconds, the primary thief starts to scout for a victim. When a victim has been found, the thief moves straight towards the victim in order to carry out the theft. If the bounding volumes of the thief and the victim intercept, then the theft is considered complete. After this, the two thieves intercept each other, after which they leave the scenario. The speeds of thieves are chosen randomly to be between 2 and 3 m/s. In pick pocket mode, the primary thief will have its speed set to 3.5 m/s, and after completion, the speed is reset to a random value between 2 and 3 m/s. Thieves also tries to avoid colliding with each other and with pedestrians, except when in intercept mode. The layout of the pick pocket scenario is illustrated in Figure 9.2.

![Image of the simulator](image.png)

**Figure 9.2:** Illustration of the pick pocket scenario in the simulator.
9.1.3 Extraction of events

With the existence of object level behaviours, the next step lay in constructing an abstract state space that can be used for defining interesting situations. The modelled interesting situation is executed in five distinct steps which consist of: (1) scouting for victim, (2) approaching victim, (3) proximity with victim, (4) thieves intercepting each other and (5) proximity between thieves. Three distinct spatio-temporal relations can be extracted from this definition: \textit{Approach}(x, y), \textit{Close}(x, y) and \textit{Intercept}(x, y). For each of these relations, a relation extraction procedure that makes use of an internal knowledge base is used. Initially, the knowledge base for each type of relation is empty. As observations are processed, relational information is added to and removed from each respective knowledge base. When relations are added or removed from a knowledge base, corresponding events are also distributed externally (e.g. to situation recognition components). An addition of a specific relation results in an event containing the relation, the value \textit{true} and a time point for the event. Similarly, a removal of a specific relation results in an event with the relation, the value \textit{false} and the time point of the event. In the following, the relations are also denoted by \textit{Cl}(x, y), \textit{App}(x, y) and \textit{Int}(x, y), respectively.

To start with, spatial proximity, \textit{Close}(x, y), can be implemented rather easily by considering the Euclidean distance between two objects. Two objects can be said to be close to each other when the distance between them is less than a cut off distance $\varphi$. During experimentation has however been discovered that this results in very unstable results with very many events being extracted due to small fluctuations (close true, close false, close true, ...). Thus, the close relation between two objects has instead been derived through the use of an activation distance $\varphi_i$ and a deactivation distance $\varphi_o$. Formally, the relation \textit{Close}(x, y) is at discrete time step $t$ extracted as follows:

$$
KB_{Cl} \leftarrow \text{add } Cl(x, y) \mid d(p(x)_t, p(y)_t) < \varphi_i \wedge \neg Cl(x, y)
$$
$$
KB_{Cl} \leftarrow \text{remove } Cl(x, y) \mid d(p(x)_t, p(y)_t) > \varphi_o \wedge Cl(x, y),
$$

where $p(x)_t$ denotes the position of $x$ at $t$ and where $d(p_1, p_2)$ denotes the Euclidean distance between two positions. The close relation is a symmetrical relation, in other words \textit{Close}(x, y)$ \rightarrow$ \textit{Close}(y, x).

Next we have the \textit{Approach}(x, y) relation. The aim of this relation is to find out if an object $x$ seems to be purposefully approaching another object $y$. It is therefore not enough to simply look for a decreasing distance between the two objects, but also, we need to look at the direction of the two objects, $x$ needs to be headed towards $y$. Similar to the \textit{Close}(x, y) relation, it has during experimentation been found that the use of an activation angle $\omega_i$ and a deactivation angle $\omega_o$, provides stable relation extraction. Events for the relation \textit{Approach}(x, y) are therefore formally defined as:
Through experimentation, the following definitions have been found suitable: unary relations should denote objects that significantly change their speeds. These changes in speed can either be speed-ups or slow-downs. From this, two additional spatio-temporal relations can be extracted:

Lastly, we have the relation \( \text{Intercept}(x,y) \). The aim of this relation is to find out if an object \( x \) tries to intercept an object \( y \). To determine if the intercept relation is true, the closest point of approach (CPA) can be used. This metric denotes the future distance between two objects when they are at their closest, given that they continue with the same speed on their present course. Moreover, since the aim is to determine if \( x \) intercepts \( y \), then the relation can also be conditioned on a decreasing distance between \( x \) and \( y \) for two consecutive updates of \( x \). Similar to before, it has been found out during experimentation that the use of an activation limit \( \tau_i \) and a deactivation limit \( \tau_o \) provides stable relation extraction. These limits are applied to the closest point of approach metric. Formally, events for the \( \text{Intercept}(x,y) \) relation are defined as follows:

\[
KB_{\text{Int}} \leftarrow \text{add } \text{Int}(x,y) \mid (d(p(x)_{t-1},p(y)_{t-1}) - d(p(x)_t,p(y)_t) > 0 \land \nonumber \\
\text{cpa}(x,y) < \tau_i \land \neg\text{Int}(x,y) \nonumber \\
KB_{\text{Int}} \leftarrow \text{remove } \text{Int}(x,y) \mid (d(p(x)_{t-1},p(y)_{t-1}) - d(p(x)_t,p(y)_t) < 0 \lor \nonumber \\
\text{cpa}(x,y) > \tau_o \land \text{Int}(x,y),
\]

where \( \text{cpa}(x,y) \) denotes the CPA metric between objects \( x \) and \( y \). Noting is that \( \text{Approach} \) and \( \text{Intercept} \) are asymmetrical relations, in other words \( \text{Approach}(x,y) \not\rightarrow \text{Approach}(y,x) \) and \( \text{Intercept}(x,y) \not\rightarrow \text{Intercept}(y,x) \).

In addition to the defined binary relations, pedestrians and thieves also change their speeds at various points in time. These changes in speed can either be speed-ups or slow-downs. From this, two additional spatio-temporal relations can be extracted: \( \text{SpeedingUp}(x) \) and \( \text{SlowingDown}(x) \). These two unary relations should denote objects that significantly change their speeds. Through experimentation, the following definitions have been found suitable:

\[
KB_{\text{SpU}} \leftarrow \text{add } \text{SpU}(x) \mid \Delta\text{avg}_s(x_{t-1}) > 0 \land \Delta\text{avg}_s(x_t) > \text{avg}_s(x_{t-1}) \land \nonumber \\
\text{abs}\left(\Delta\text{avg}_s(x_{t-1})\right) > 0.05\text{avg}_s(x_t) \land \neg\text{SpU}(x) \nonumber \\
KB_{\text{SpU}} \leftarrow \text{remove } \text{SpU}(x) \mid \text{abs}\left(\Delta\text{avg}_s(x_t)\right) < \text{abs}(s(x_t) - 0.05\text{avg}_s(x_t)) \land \nonumber \\
\text{SpU}(x),
\]
KB_{SLD} \leftarrow \text{add } SlD(x) \mid \Delta \text{avg}_s(x_{t-1}) < 0 \land \Delta \text{avg}_s(x_t) < \Delta \text{avg}_s(x_{t-1}) \land \text{abs}(\Delta \text{avg}_s(x_{t-1})) > 0.05 \text{avg}_s(x_t) \land \neg SlD(x)
KB_{SLD} \leftarrow \text{remove } SlD(x) \mid \text{abs}(\Delta \text{avg}_s(x_t)) < \text{abs}(s(x_t) - 0.05 \text{avg}_s(x_t)) \land SlD(x),

where \( SpU \) is short for \( \text{SpeedingUp} \), \( SlD \) is short for \( \text{SlowingDown} \), \( s(x_t) \) denotes the speed of object \( x \) at \( t \), \( \text{avg}_s(x_t) \) denotes a weighted average speed of \( x \) at \( t \) calculated as \( \text{avg}_s(x_t) = 0.25 s(x_t) + 0.75 \text{avg}_s(x_{t-1}) \), and where \( \text{abs}(v) \) denotes the absolute value of \( v \). In their essence, these relations are true when the delta of the weighted average speed of an object is more than 5% of the weighted average speed of the object, with the other conditions being put in place for stability.

The following parameter settings have been used in all pick pocket simulations: \( \varphi_i = 2 \), \( \varphi_o = 3 \), \( \omega_i = 3^\circ \), \( \omega_o = 5^\circ \), \( \tau_i = 1 \) and \( \tau_o = 3 \). Moreover, approach and intercept have in order to avoid irrelevant relations only been extracted for pairs of objects \( x \) and \( y \) that are within 30 m of each other.

### 9.1.4 Interesting situations

Using the defined relations, a template for the interesting situation, \( T = (X, C) \), can be constructed using three variables for objects, \( X = \{x_1, x_2, x_3\} \), five object level constraints \( C_s \) and five temporal constraints \( C_t \), where \( C = C_s \cup C_t \). \( C_s \) and \( C_t \) of the template are defined as follows:

\[
\begin{align*}
C_{s_1} &= \text{Approach}(x_1, x_2) & C_{t_1} &= \text{before}(C_{s_1}, C_{s_2}) \\
C_{s_2} &= \text{Close}(x_1, x_2) & C_{t_2} &= \text{before}(C_{s_2}, C_{s_3}) \\
C_{s_3} &= \text{Intercept}(x_1, x_3) & C_{t_3} &= \text{before}(C_{s_2}, C_{s_4}) \\
C_{s_4} &= \text{Intercept}(x_3, x_1) & C_{t_4} &= \text{before}(C_{s_3}, C_{s_5}) \\
C_{s_5} &= \text{Close}(x_1, x_3) & C_{t_5} &= \text{before}(C_{s_4}, C_{s_5})
\end{align*}
\]

### 9.2 Piloting situations in real data

The ability of being able to detect interesting vessel activity is becoming an increasingly important goal within the maritime surveillance domain. Numerous vessels travel along the coasts, and some of these may be interesting to identify. For example, trafficking of drugs, weapons and illicit substances such as chemical agents, remains a threat from criminal- and terrorist groups. The borders of Europe represent interception choke points for such goods, however, only limited control is possible. Moreover, the European Union has expanded
and today consists of around 85,000 km of coastlines (European Security Research Advisory Board, 2006). This even further highlights the importance of introducing enhanced surveillance capabilities. Numerous radar stations are located along the coastlines to survey traffic. These can provide quite accurate tracks of individual vessels. Moreover, most large size vessels are equipped with AIS systems that transmit the location and additional information to nearby receiving stations. Both of these sources can be used for identifying complex behaviours of interest, such as for example smuggling of drugs or other goods. Situation recognition could be an interesting capability to include here, for detecting such interesting situations.

In order to demonstrate the applicability of Petri net based situation recognition in a real world scenario, it therefore becomes interesting to investigate its use in a maritime surveillance scenario. A problem however, is that there is a lack of identified situations of interest that can be used for verification. Instead, it is argued that the applicability can be demonstrated using piloting situations. Such situations are naturally occurring on a daily basis, and they can be considered suitable since they consist of activities that include behaviours of multiple objects over both space and time.

### 9.2.1 Scenario

The use of a Bayesian network approach for recognising piloting situations is investigated in Fooladvandi et al. (2009). It is suggested that a similar scenario is used here. Larger vessels arriving to a port of call are often assisted by harbour pilots in the docking procedure. The reason, simply put, is to minimise the number of accidents through the exploitation of localised knowledge about the harbour and approach area. There are many different types of piloting situation. Fooladvandi et al. for example use a scenario where pilot boats are actively engaged in the docking procedure, e.g. similar to towing. Another variant, is that a pilot boats meets an approaching vessel in a designated area, a harbour pilot boards the vessel, and then in persona aids in the approach and docking procedure. This is the scenario considered in this thesis. The scenario is illustrated in Figure 9.3 and the activities consist of:

1. A pilot boat leaves its docking position.
2. A vessel enters a predesignated meeting zone.
3. The pilot boat meets the vessel in the meeting zone and a harbour pilot boards the vessel.
4. The vessel moves towards the harbour (assisted by the pilot).
5. The vessel enters a harbour area and docks (thereby not moving any more).
6. The pilot boat returns to its docking position after possibly having picked up the pilot (the pilot can of course also use other means of transportation or remain for a departing vessel).
9.2. PILOTING SITUATIONS IN REAL DATA

Figure 9.3: Activities in the piloting scenario (adapted from Fooladvandi et al., 2009).

Existing AIS data can be used for recognising this type of situation through the use of an abstract state space put on top of the object level data.

9.2.2 Extraction of events

Fooladvandi et al. (2009) describe the use of five higher level concepts for modelling the interesting pilot boat situation: (1) vessel waiting at area, (2) vessel meets pilot boat, (3) vessel escorted by pilot boat, (4) vessel reaches harbour and (4) pilot boat returns to its starting area. From the previous scenario description and from these higher level concepts, four interesting relations are defined: $\text{Waiting}(x)$, $\text{InArea}(x,y)$, $\text{Close}(x,y)$ and $\text{InHarbour}(x,y)$.

Events for the $\text{Close}$ relation can be derived as in the pick pocket scenario however, the limits for extraction needs to be increased due to vessels being of much larger sizes than pedestrians and also due to the scenario playing out in a less confined space. The limits are thus set to: $\varphi_i = 100$ and $\varphi_o = 150^3$. The unary relation $\text{Waiting}(x)$ simply denotes an object being located in roughly the same position over some period of time. It can thus be defined by looking at the change in position over time, speed, and this change should be close to zero. Similarly to the pick pocket scenario, two limits, $\kappa_i$ and $\kappa_o$, are used to achieve stable relation extraction. Events for the $\text{Waiting}(x)$ relation, also denoted $Wa(x)$, are formally defined as:

$$KB_{Wa} \leftarrow \text{add } Wa(x) \mid \text{avg}_s(x_t) < \kappa_i \wedge \neg Wa(x)$$
$$KB_{Wa} \leftarrow \text{remove } Wa(x) \mid \text{avg}_s(x_t) > \kappa_o \wedge Wa(x),$$

where $\text{avg}_s(x_t)$ denotes the average speed of $x$ at $t$, calculated as $\text{avg}_s(x_t) = 0.25s(x_t) + 0.75\text{avg}_s(x_{t-1})$, where $s(x_t)$ denotes the speed of object $x$ at $t$, calculated as the change in displacement between $t - 1$ and $t$.

\footnote{Freighters and tankers can easily be a coupled of hundred meters long, which means that the selected limits at occasion even may be too small.}
9. SCENARIOS FOR EVALUATION

The \( \text{InArea}(x, y) \) relation concerns an object \( x \) begin located within an area \( y \). For simplicity, all areas are in this thesis estimated using either circular or rectangular shapes. Moreover, vessels are approximated using circular bounding volumes with a radius of 10 m. With these assumptions, the \( \text{InArea}(x, y) \) relation can be estimated using an intersection test between the two area shapes. Again however, to achieve stability a limit \( \psi \) is used. Events for the \( \text{InArea}(x, y) \) relation, also denoted with \( \text{InA}(x, y) \) are formally defined as:

\[
\text{KB}_{\text{InA}} \leftarrow \text{add InA}(x, y) \mid \text{overlap}(x, y) > \psi/2 \land \neg \text{InA}(x, y)
\]
\[
\text{KB}_{\text{InA}} \leftarrow \text{remove InA}(x, y) \mid \text{overlap}(x, y) < -\psi/2 \land \text{InA}(x, y),
\]

where \( \text{overlap}(x, y) \) denotes the number of meters of overlap in 2D Euclidean space between the shapes of objects \( x \) and \( y \). The last relation to be defined is \( \text{InHarbour}(x, y) \), which should depict an object \( x \) being located in a harbour area \( y \). The relation can be extracted in the same fashion as the \( \text{InArea} \) relation; however, also assuring that the area is actually a harbour area. The parameters for all relations have been set as follows: \( \kappa_i = 0.5 \), \( \kappa_o = 1.5 \) and \( \psi = 10 \).

9.2.3 Interesting situations

Through the use of the defined relations, a template for the interesting situation, \( T = (X, C) \), can be constructed using five variables for objects, \( X = \{x_1, x_2, a_1, a_2, a_3\} \), six object level constraints \( C_s \) and six temporal constraints \( C_t \). The constraints in the template are defined as \( C = C_s \cup C_t \), where \( C_s \) and \( C_t \) are defined as follows:

\[
\begin{align*}
C_{s_1} &= \neg \text{InHarbour}(x_2, a_3) & C_{t_1} &= \text{before}(C_{s_1}, C_{s_2}) \\
C_{s_2} &= \text{InArea}(x_1, a_1) & C_{t_2} &= \text{before}(C_{s_2}, C_{s_3}) \\
C_{s_3} &= \text{Close}(x_1, x_2) & C_{t_3} &= \text{before}(C_{s_3}, C_{s_4}) \\
C_{s_4} &= \text{InHarbour}(x_1, a_2) & C_{t_4} &= \text{before}(C_{s_4}, C_{s_6}) \\
C_{s_5} &= \text{Waiting}(x_1) & C_{t_5} &= \text{before}(C_{s_4}, C_{s_5}) \\
C_{s_6} &= \text{InHarbour}(x_2, a_3) & C_{t_6} &= \text{before}(C_{s_6}, C_{s_7}) \\
C_{s_7} &= \text{Waiting}(x_2)
\end{align*}
\]

9.3 Chapter summary

Two scenarios have been introduced in this chapter. The first is a fictive pick pocket scenario that has been implemented in the simulator presented in Chapter 8. The second is a piloting boat scenario based on work presented in Fooladvandi et al. (2009), and it plays out in the real world. The two scenarios will be used in consecutive empirical investigations into recognition and learning.
Part IV

Empirical results
Chapter 10
Recognition results

This chapter addresses the fourth research objective of this thesis, namely, to empirically evaluate techniques for situation recognition. The content of the chapter is largely based on two accepted publications: Dahlbom et al. (2009b) and Dahlbom et al. (2010a), and one submitted paper. The contributions of the chapter consist of:

- Empirical results comparing a Petri net based technique for situation recognition with a rule based technique using the Rete algorithm with temporal extensions.
- Empirical results evaluating the effects of trading memory for speed in Petri net based situation recognition, through the use of precombinations.

10.1 Experimental setup

This chapter makes use of both scenarios presented in Chapter 9. The simulated pick pocket scenario consists of numerous pedestrians moving around on their daily business. On top of this model a number of “interesting” pick pocket situations have been dynamically implemented. The piloting scenario is based on real AIS data collected around the archipelago of Gothenburg, Sweden. In this data there are a number of naturally occurring piloting situations and the task is to recognise these, without the aid of labelled data.

10.1.1 Notation

The notation that is used in this chapter is presented in Table 10.1.

10.1.2 Pick pocket scenario

The pick pocket scenario has been thoroughly described in Chapter 9, and thus, only relevant parameter settings are presented here. Five different setups ($S_1$ -
Table 10.1: The notation used when presenting recognition results.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RB</td>
<td>The rule based recogniser has been used.</td>
</tr>
<tr>
<td>PN</td>
<td>The Petri net based recogniser has been used.</td>
</tr>
<tr>
<td>+PC</td>
<td>Pre combination as described in Section 5.6 has been used.</td>
</tr>
<tr>
<td>TP</td>
<td>True positives</td>
</tr>
<tr>
<td>FP</td>
<td>False positives</td>
</tr>
<tr>
<td>FN</td>
<td>False negatives</td>
</tr>
<tr>
<td>μ</td>
<td>Mean</td>
</tr>
<tr>
<td>σ</td>
<td>Sample standard deviation.</td>
</tr>
<tr>
<td>P</td>
<td>Precision (calculated according to Equation 6.3).</td>
</tr>
<tr>
<td>R</td>
<td>Recall (calculated according to Equation 6.2).</td>
</tr>
</tbody>
</table>

The results of the scenario setups are a varying number of average pedestrians moving around in the environment. On top of these models of normalcy, a number of pick pocket situations have been instantiated. Pick pocket situations are constructed using a “thief factory” object in the simulator. This object is invoked every five seconds, and at each invocation there is a 30% chance of instantiating a pick pocket situation in case one is not already in progress. Since the pick pocket situations are dynamic, the result is a number of differently allocated situations in each simulation. Simulations have also been carried out for each scenario setup without creating any pick pocket situations. These setups are denoted $S'_i$ to $S'_5$. The reason for having access to these simulations is to be able to investigate the default false alarm rate. Object statistics for the different scenario setups have been collected and are presented in Table 10.3.

As can be seen in Table 10.3 the number of created pedestrians increases in the varying setups. Moreover, the number of generated pedestrians in setups $S'_i$ and $S_i$ are similar. It can also be noted that the number of instantiated pick pocket situations remains roughly the same, as intended. As also can be seen, the average number of active pedestrians (average number of pedestrians at
10.1. EXPERIMENTAL SETUP

Table 10.3: Object statistics for the generated pick pocket scenarios.

<table>
<thead>
<tr>
<th></th>
<th>Pedestrians</th>
<th></th>
<th>Pick pockets</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total created</td>
<td>Average active (0-600 sec)</td>
<td>Average active (60-600 sec)</td>
<td>Instantiated</td>
</tr>
<tr>
<td></td>
<td>( \mu )</td>
<td>( \sigma )</td>
<td>( \mu )</td>
<td>( \sigma )</td>
</tr>
<tr>
<td>( S_1 )</td>
<td>130.3</td>
<td>11.53</td>
<td>9.77</td>
<td>1.804</td>
</tr>
<tr>
<td>( S_2 )</td>
<td>227.1</td>
<td>17.70</td>
<td>16.54</td>
<td>3.156</td>
</tr>
<tr>
<td>( S_3 )</td>
<td>289.0</td>
<td>11.93</td>
<td>20.97</td>
<td>4.156</td>
</tr>
<tr>
<td>( S_4 )</td>
<td>346.5</td>
<td>17.53</td>
<td>25.10</td>
<td>4.857</td>
</tr>
<tr>
<td>( S_5 )</td>
<td>405.3</td>
<td>11.59</td>
<td>29.65</td>
<td>5.763</td>
</tr>
<tr>
<td>( S'_1 )</td>
<td>119.7</td>
<td>10.32</td>
<td>7.69</td>
<td>0.589</td>
</tr>
<tr>
<td>( S'_2 )</td>
<td>195.3</td>
<td>17.29</td>
<td>14.26</td>
<td>0.718</td>
</tr>
<tr>
<td>( S'_3 )</td>
<td>269.2</td>
<td>17.83</td>
<td>18.24</td>
<td>0.951</td>
</tr>
<tr>
<td>( S'_4 )</td>
<td>332.4</td>
<td>12.98</td>
<td>22.98</td>
<td>0.896</td>
</tr>
<tr>
<td>( S'_5 )</td>
<td>388.6</td>
<td>16.90</td>
<td>27.27</td>
<td>1.010</td>
</tr>
</tbody>
</table>

Each point in time) stabilises as time passes. This can be observed as a decrease in standard deviations when measuring from time 60 - 600 compared to 0 - 600. The reason for this is that there initially are no active pedestrians.

The data generated from each simulation consist of the ground truth sampled at a frequency of 4Hz. Each sample consists of observations of all active objects in the environment during the sample duration. These observations are analysed to extract relational information, and more specifically, Close, Approach and Intercept relations. For more details about the extraction procedures, please refer to Chapter 9. Statistics about the resulting extracted events has been gathered and are presented in Table 10.4.

Table 10.4: Event statistics for the generated pick pocket scenarios.

<table>
<thead>
<tr>
<th></th>
<th>Close</th>
<th></th>
<th>Approach</th>
<th></th>
<th>Intercept</th>
<th></th>
<th>( \Sigma )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \mu )</td>
<td>( \sigma )</td>
<td>( \mu )</td>
<td>( \sigma )</td>
<td>( \mu )</td>
<td>( \sigma )</td>
<td>( \mu )</td>
</tr>
<tr>
<td>( S_1 )</td>
<td>60.4</td>
<td>33.3</td>
<td>575.4</td>
<td>148.5</td>
<td>378.4</td>
<td>157.3</td>
<td>1014.2</td>
</tr>
<tr>
<td>( S_2 )</td>
<td>368.0</td>
<td>538.5</td>
<td>1743.7</td>
<td>697.1</td>
<td>1156.7</td>
<td>505.2</td>
<td>3268.3</td>
</tr>
<tr>
<td>( S_3 )</td>
<td>713.2</td>
<td>949.1</td>
<td>2605.0</td>
<td>1406.3</td>
<td>2284.5</td>
<td>837.4</td>
<td>5602.6</td>
</tr>
<tr>
<td>( S_4 )</td>
<td>1453.6</td>
<td>1311.2</td>
<td>3583.3</td>
<td>2298.4</td>
<td>3549.5</td>
<td>1925.4</td>
<td>8586.4</td>
</tr>
<tr>
<td>( S_5 )</td>
<td>1585.7</td>
<td>2466.3</td>
<td>6246.6</td>
<td>2562.0</td>
<td>4901.8</td>
<td>2161.8</td>
<td>12734.1</td>
</tr>
<tr>
<td>( S'_1 )</td>
<td>50.6</td>
<td>74.4</td>
<td>238.4</td>
<td>110.8</td>
<td>308.5</td>
<td>134.3</td>
<td>597.4</td>
</tr>
<tr>
<td>( S'_2 )</td>
<td>192.2</td>
<td>245.5</td>
<td>887.3</td>
<td>373.9</td>
<td>1362.7</td>
<td>559.6</td>
<td>2442.2</td>
</tr>
<tr>
<td>( S'_3 )</td>
<td>663.4</td>
<td>860.5</td>
<td>1520.6</td>
<td>926.9</td>
<td>2101.5</td>
<td>933.8</td>
<td>4285.5</td>
</tr>
<tr>
<td>( S'_4 )</td>
<td>1291.5</td>
<td>1368.8</td>
<td>2648.5</td>
<td>1740.6</td>
<td>3469.8</td>
<td>1740.5</td>
<td>7409.8</td>
</tr>
<tr>
<td>( S'_5 )</td>
<td>1553.9</td>
<td>1679.1</td>
<td>4225.1</td>
<td>2366.7</td>
<td>5233.4</td>
<td>2656.6</td>
<td>11012.4</td>
</tr>
</tbody>
</table>

As can be observed in Table 10.4, the number of extracted events is not stable for any of the setups (the deviations are too large). This means that the extracted events are not normally distributed. This can be both positive and negative. It can be negative since results that compare various algorithms
over the different data sets for each setup, cannot directly be used for carrying out significance tests. Instead, comparisons need to be carried for each distinct data set. These results can then be counted and ranks can be established. The nature of the distribution can however also be viewed as positive, since it means that the data sets are of varying nature. This in turn implies that the results of algorithm comparisons have been carried out on a larger number of distinct cases. Results can thus be argued to be more general. Nonetheless, it can be observed in Table 10.4 that the number of events increases with scenario setup. This means that there is an increasing complexity over the scenarios. Moreover, the number of events increases in a similar way with and without pick pockets.

**The interesting situation**

A situation template for recognising pick pocket situations was presented in Chapter 9. To compare the two algorithms, the situation template has been converted into a Petri net as illustrated in Figure 10.1 and to a complex rule as illustrated in Listing 10.1. Variables $x_2$ and $x_3$ have been replaced with $y$ and $z$ in the complex rule. When loaded into the development platform, the complex rule is automatically translated into the Rete representation with temporal extensions.

```
rule "PickPocket"
IF ( (Approach(x,y) BEFORE Close(x,y)) AND
    (Close(x,y) BEFORE Intercept(x,z)) AND
    (Close(x,y) BEFORE Intercept(z,x)) AND
    (Intercept(x,z) BEFORE Close(x,z)) AND
    (Intercept(z,x) BEFORE Close(x,z))
)
THEN
    Send event PickPocket(x, y, z)
END
```

**Figure 10.1:** A Petri net representing a template describing a pick pocket situation.

**Listing 10.1:** Complex rule for a pick pocket situation template.
10.1.3 Maritime scenario

The maritime piloting scenario is based on real AIS data that has been collected around the archipelago of Gothenburg, Sweden. The data consists of AIS reports sent from individual vessels to receiving devices around the coast. The data has been gathered between February 15 and February 16, 2010. There is thus only one large data set that can be used. This is however a rather large data set and in order to restrict the universe of interest, a rectangular area around the archipelago and harbour of Gothenburg has been used as a first filter (North 57°34′- 57°87′, East 10°9′ - 12°1′). Statistics about the resulting data set are presented in Table 10.5.

<table>
<thead>
<tr>
<th></th>
<th>μ</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of observations</td>
<td>9,718,97</td>
<td>-</td>
</tr>
<tr>
<td>Total number of objects</td>
<td>290</td>
<td>-</td>
</tr>
<tr>
<td>Average number of active objects (per 30 seconds)</td>
<td>80.72</td>
<td>31.15</td>
</tr>
</tbody>
</table>

The individual reports from vessels have after the filtering procedure been analysed to extract relational information. More specifically, relations: Close, InArea, InHarbour and Waiting, are extracted. The procedures for extracting these relations are described in Chapter 9. Statistics about the extracted events are presented in Table 10.6.

<table>
<thead>
<tr>
<th></th>
<th>Close</th>
<th>InArea</th>
<th>InHarbour</th>
<th>Waiting</th>
<th>∑</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>35731</td>
<td>4686</td>
<td>14009</td>
<td>6005</td>
<td>60431</td>
</tr>
</tbody>
</table>

10.2 Recognition performance

Situation recognition is depicted as being part of a system where real time demands may be put on the algorithms and representations that are used. Observations are processed, relational information is derived, situations are recognised, and finally, situations are again inserted into the system for further usage. Time constraints may be put on the situation recognition component from at least two different perspectives: internal and external. From the internal view, data needs to be processed at least on the average frequency at which it is made available. If this is not the case, then unprocessed data will queue up to eventually reach system limits. From the external view, other components in the system may depend on situations being recognised in a timely fashion,
e.g. we would like to know now if something is occurring now, not tomorrow. Efficiency can be an important issue in both views.

Although a wide range of techniques may be used for addressing the situation recognition problem, this thesis focuses on two: rule based techniques and Petri nets. Already, rule based techniques using the Rete algorithm (Forgy, 1982) are known to be efficient. Furthermore, temporal extensions to the Rete algorithm have been suggested to allow for temporal constraints to be modelled (Walzer, 2009). In order to determine if the Petri net based technique also provides an efficient basis for recognition, it has been argued that its efficiency should be on par with that of the rule based technique. Thus, the first research question was formulated:

**Research question 1.** Can Petri nets be used for recognising situations as efficiently as rule based approaches using the Rete algorithm with extensions for explicitly modelling temporal constraints?

The main objective of any technique for situation recognition is to be able to recognise situations. Thus, in order for efficiency results to be interesting it needs to be known that the investigated techniques actually can be used for recognising situations. Although both techniques in theory should be able to do so given a suitable underlying symbolic language, this needs to be verified through empirical investigations. It could be the case that the input cannot at all be used for recognising situations. In order for efficiency results to be meaningful, it however also needs to be asserted that the two techniques actually have comparable performance. There could be differences however, given the same symbolic language and patterns that represent the same subspace of situations, then their performance should be similar. Three hypotheses can be formulated:

**Hypothesis R₁.** The Petri net based technique can be used for recognising situations.

**Hypothesis R₂.** The rule based technique can be used for recognising situations.

**Hypothesis R₃.** The Petri net based technique and the rule based technique have similar performance on the situation recognition task given that similar representations are used.

An extension to the Petri net based technique was suggested in Chapter 5. The use of this extension should in theory lower the time needed for recognising situations, at the expense of memory. The extension consists of using pre-computed combinations of tokens. A similar approach is also suggested by Ghanem et al. (2004), who claim that it is an efficient approach for implementing Petri nets for recognition. The extensions should not affect the recognition performance. However, practical use does not always align with theory. Unthought-of
aspects may exist. This is however not likely, since the extension does not carry out computations differently. It is only when they are carried out that differ. The following hypothesis can be formulated:

\[ \text{Hypothesis } \mathcal{R}_4. \text{ The suggested extension to the Petri net based technique does not affect recognition performance.} \]

### 10.2.1 Method

In order to investigate the formulated hypotheses, empirical investigations are needed to assert that the techniques can be used for recognising situations. Moreover, the performance for each of the techniques, as well as when using the suggested extension, needs to be compared. In order for hypothesis \( \mathcal{R}_1 \) to hold, it is necessary that the Petri net based technique can be used for recognising situations. In order for hypothesis \( \mathcal{R}_2 \) to hold, it is necessary that the rule based technique can be used for recognising situations. In order for hypothesis \( \mathcal{R}_3 \) to hold, the performance of the Petri net based technique and the rule based technique needs to be similar. Finally, in order for hypothesis \( \mathcal{R}_4 \) to hold, it is necessary that the performance of the Petri net based technique is the same when using the extensions, as when not using it.

It is suggested that the investigations are carried out using the different setups of the simulated pick pocket scenario, since these allow for means and deviations to be established. Furthermore, the complexity increases in various setups and this allows for comparisons to be carried out with respect to increasing complexity.

In Chapter 4, it was established that the performance of a solution to the situation recognition problem can be measured using precision and recall. Recall is a measure denoting how many of the existing interesting situations that were found and precision is a measure that denotes the number of positively classified situations that actually were interesting (it is the inverse of the number of false alarms). It can on occasion be problematic to separate two solutions when two measures are used. Picture a case where one solution has good recall but bad precision and another solution has good precision but bad recall. Which of the solutions should be considered the best? Lingard and Lambert (2008) discuss the use of the f-value for combining precision and recall into a single metric. The f-value is the geometric mean of precision and recall, and it can be used for measuring the quality of a solution. The f-value can be calculated as:

\[
F_v = \frac{2 \cdot R \cdot P}{R + P}
\]  

(10.1)

In addition to precision and recall, it is argued that the f-value also should be calculated. It can however be misleading to only compare the algorithms using a single measure, since important information may be neglected. In combination however, the three measures should allow us to compare various solutions.
10.2.2 Results

In order to determine if both rule based recognition and Petri net based recognition can be used for actually recognising situations, and in order to determine if there are any differences in their performance, each of the two techniques has been evaluated on each of the five setups of the pick pocket scenario. As discussed, each setup consists of 20 data sets, and thus, average precision, recall and f-value are based on the recognition performance achieved on each of the data sets. Table 10.7 shows the average recognition results for both techniques and for all setups.

Table 10.7: Precision, recall and f-value results for Petri net (PN) based and rule based (RB) recognition, in five different setups of the pick pocket scenario.

<table>
<thead>
<tr>
<th>S</th>
<th>Precision</th>
<th>Recall</th>
<th>F-value</th>
<th>FP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PN</td>
<td>RB</td>
<td>PN</td>
<td>RB</td>
</tr>
<tr>
<td>S₁</td>
<td>0.6274</td>
<td>0.6234</td>
<td>0.9783</td>
<td>0.9783</td>
</tr>
<tr>
<td></td>
<td>0.2235</td>
<td>0.2170</td>
<td>0.0535</td>
<td>0.0535</td>
</tr>
<tr>
<td>S₂</td>
<td>0.3146</td>
<td>0.3052</td>
<td>0.9028</td>
<td>0.8957</td>
</tr>
<tr>
<td></td>
<td>0.1011</td>
<td>0.0976</td>
<td>0.1093</td>
<td>0.1169</td>
</tr>
<tr>
<td>S₃</td>
<td>0.1908</td>
<td>0.1850</td>
<td>0.9045</td>
<td>0.8982</td>
</tr>
<tr>
<td></td>
<td>0.0564</td>
<td>0.0612</td>
<td>0.1120</td>
<td>0.1304</td>
</tr>
<tr>
<td>S₄</td>
<td>0.1262</td>
<td>0.1222</td>
<td>0.9062</td>
<td>0.9068</td>
</tr>
<tr>
<td></td>
<td>0.0277</td>
<td>0.0268</td>
<td>0.1092</td>
<td>0.1091</td>
</tr>
<tr>
<td>S₅</td>
<td>0.0745</td>
<td>0.0712</td>
<td>0.8734</td>
<td>0.8734</td>
</tr>
<tr>
<td></td>
<td>0.0233</td>
<td>0.0223</td>
<td>0.1516</td>
<td>0.1516</td>
</tr>
</tbody>
</table>

As can be seen in Table 10.7, the two techniques achieve similar results for every setting. This indicates that their performance is similar. There are, however, some differences, and this means that their definitions of the interesting pattern are not identical. It can also be seen that the performance degrades as the complexity of the scenario increases. For the least complex setup, both techniques are able to achieve acceptable recognition results with only around 5 false alarms on average. However, the precision and f-value results achieved in the denser scenarios are not satisfactory. In the most complex scenario, there are around 80 - 90 false alarms on average.

Two possible reasons for this drastic decay read as follows. Firstly, it could depend on the definitions of instantiated pick pocket situations. The only difference in the input data is the number of pedestrians that are moving around in the environment. As more pedestrians are instantiated however, the nature of instantiated pick pocket situation may change, since these are dynamically allocated and depend on the modelling of pedestrians. Secondly, it can depend on an increase in the number of unintentionally existing patterns. These are patterns that are identical to the intended pattern of a pick pocket situation, however, they are unintended such patterns.
The number of pedestrians affects the number of events about relations that are extracted. In the densest scenario there are for example on average 12734 extracted events (see Section 10.1.2). As the number of events increase, so does the risk of the pattern emerging by chance. Of these two reasons, it is believed that the latter is more likely, since a pick pocket situation still needs to go through a distinct set of steps to be instantiated. Moreover, the completion of these steps does not depend on the number of active pedestrians.

In order to investigate the number of unintentionally existing situations in the scenarios, both of the algorithms have been investigated using scenario setups $S'_1$ to $S'_5$. These are identical to setups $S_1$ to $S_5$, except that there are no intentionally instantiated pick pocket situations. Moreover, it is also possible to use the number of instantiated pick pocket situations from scenarios $S_1$ to $S_5$, in order to predict the precision of the two algorithms. Table 10.8 presents the number of identified situations for each of the two algorithms in scenario setups $S'_1$ to $S'_5$. The predicted precision is also included in the table.

Table 10.8: The number of identified pick pocket situations in scenario setups without any intentionally instantiated pick pocket situations for Petri net (PN) based and rule based (RB) recognition. $PP(S_i)$ denotes the number of intentionally instantiated pick pocket situations in scenario $S_i$ which is of the same complexity as $S'_i$, and $Pr$ denotes the predicted precision if pick pocket situations would have been included.

<table>
<thead>
<tr>
<th></th>
<th>PN</th>
<th></th>
<th>RB</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$PP(S_i)$</td>
<td>Identified</td>
<td>$Pr$</td>
<td>Identified</td>
</tr>
<tr>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
<td>$\mu$</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>$S'_1$</td>
<td>6.15</td>
<td>1.20</td>
<td>1.44</td>
<td>0.84</td>
</tr>
<tr>
<td>$S'_2$</td>
<td>7.20</td>
<td>8.30</td>
<td>6.42</td>
<td>0.46</td>
</tr>
<tr>
<td>$S'_3$</td>
<td>7.20</td>
<td>14.80</td>
<td>7.24</td>
<td>0.33</td>
</tr>
<tr>
<td>$S'_4$</td>
<td>7.75</td>
<td>33.00</td>
<td>12.79</td>
<td>0.19</td>
</tr>
<tr>
<td>$S'_5$</td>
<td>7.55</td>
<td>63.90</td>
<td>24.23</td>
<td>0.11</td>
</tr>
</tbody>
</table>

As can be observed in Table 10.8, the number of false positives increases with complexity. In the densest setup, there are roughly 64 – 67 unintended instances of the interesting pattern. Thus, the specific Petri net and complex rule are not restrictive enough in comparison with the normal model. Also observable in Table 10.8, is that the predicted precision of the two techniques decrease in a similar fashion to how it decreases in scenarios $S_1$ to $S_5$.

Last but not least, it needs to be investigated if the suggested extension to the Petri net based technique results in the same performance as the technique without the extension. In case the performance is not the same, then the extension interferes with the algorithm specification. This in turn means that any potential gains in efficiency are not reliable since the results are not the same. Precision, recall and f-value results for each of the setups, with and without the extension, are presented in Table 10.9.
Table 10.9: Precision, recall and f-value results of the Petri net based technique, without the use of precombinations (PN) and with the use of precombinations (PN+PC), on the five different setups of the pick pocket scenario.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F-value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>PN</td>
<td>0.627437839938</td>
<td>0.978273809524</td>
</tr>
<tr>
<td></td>
<td>PN+PC</td>
<td>0.627437839938</td>
<td>0.978273809524</td>
</tr>
<tr>
<td></td>
<td>PN</td>
<td>0.225836015953</td>
<td>0.053493921933</td>
</tr>
<tr>
<td></td>
<td>PN+PC</td>
<td>0.225836015953</td>
<td>0.053493921933</td>
</tr>
<tr>
<td>S2</td>
<td>PN</td>
<td>0.314648667464</td>
<td>0.902817460317</td>
</tr>
<tr>
<td></td>
<td>PN+PC</td>
<td>0.314648667464</td>
<td>0.902817460317</td>
</tr>
<tr>
<td></td>
<td>PN</td>
<td>0.101083041747</td>
<td>0.109334457593</td>
</tr>
<tr>
<td></td>
<td>PN+PC</td>
<td>0.101083041747</td>
<td>0.109334457593</td>
</tr>
<tr>
<td>S3</td>
<td>PN</td>
<td>0.190849640701</td>
<td>0.904464285714</td>
</tr>
<tr>
<td></td>
<td>PN+PC</td>
<td>0.190849640701</td>
<td>0.904464285714</td>
</tr>
<tr>
<td></td>
<td>PN</td>
<td>0.056355078525</td>
<td>0.112034582486</td>
</tr>
<tr>
<td></td>
<td>PN+PC</td>
<td>0.056355078525</td>
<td>0.112034582486</td>
</tr>
<tr>
<td>S4</td>
<td>PN</td>
<td>0.126175579708</td>
<td>0.906150793651</td>
</tr>
<tr>
<td></td>
<td>PN+PC</td>
<td>0.126175579708</td>
<td>0.906150793651</td>
</tr>
<tr>
<td></td>
<td>PN</td>
<td>0.027680895962</td>
<td>0.109249570397</td>
</tr>
<tr>
<td></td>
<td>PN+PC</td>
<td>0.027680895962</td>
<td>0.109249570397</td>
</tr>
<tr>
<td>S5</td>
<td>PN</td>
<td>0.074306355044</td>
<td>0.873432539683</td>
</tr>
<tr>
<td></td>
<td>PN+PC</td>
<td>0.074306355044</td>
<td>0.873432539683</td>
</tr>
<tr>
<td></td>
<td>PN</td>
<td>0.023287800953</td>
<td>0.151572346087</td>
</tr>
<tr>
<td></td>
<td>PN+PC</td>
<td>0.023287800953</td>
<td>0.151572346087</td>
</tr>
</tbody>
</table>

Unsurprisingly, the Petri net based technique behaves exactly the same with and without the extension. This was expected since the same template has been used on exactly the same data. Therefore, in case there are gains in efficiency when using the extension, then these results can be considered valid.

10.2.3 Conclusions

It has been shown that both the Petri net based technique and the rule based technique can be used for recognising situations. Hence, hypotheses $R_1$ and $R_2$ are considered valid. Moreover, the performance of the two techniques is similar. Hence, hypothesis $R_3$ is argued to hold. It has also been shown that the recognition performance of the Petri net based technique is the same with and without the extension. Hence, hypothesis $R_4$ is also considered valid.

The performance achieved on small scale setups is satisfying. However, as the complexity of the input data increases, the performance decreases. In the densest setup, the precision and f-value results are very low. This in turn means that the situation template is not able to separate interesting and uninteresting situations. This calls for an improvement of the templates that are used. However, the purpose of this chapter is not to assess the quality of individual situation templates, but instead, the purpose is to assess the quality of the Petri
The problem of combining multiple partial matches, which is addressed in both of the techniques, is of similar complexity. Given that the combination problem is of similar complexity, it should follow that the computational consumption should be similar too. However, the algorithms used for combining partial matches are different. In the Rete algorithm, only pairs of two paths are joined in each beta node. Moreover, each beta node only iteratively joins paths together, e.g. the output of a sequence of alpha nodes is joined with results from previous join operations. In the Petri net based technique, two or more paths may be joined in the same node at the same time as new information is also included. Still, the total number of join operations should roughly be equal, given that a rule and a Petri net represent the same type of situation. Hence, their computational demands should be similar. Moreover, the fact that it has been shown that Petri nets can be used as an efficient basis in rule based systems, strengthens the belief in Petri nets as an equally efficient mechanism for recognition. The following hypothesis is thus suggested:

**Hypothesis R_5.** The Petri net base technique can be used as efficiently as the rule based technique.

The problem of combining multiple partial matches can quickly grow unmanageable with respect to time. It therefore becomes interesting to investigate if extensions to the Petri net based technique can be exploited to further increase the efficiency, at the cost of more memory. Memory is, however, also a finite resource and this needs to be taken into account. The ratio between the loss in memory and gain in efficiency thus needs to be inspected.

In Section 5.6 an extension for possibly increasing efficiency of the Petri net based technique at the expense of memory, was suggested. The extension consists in precomputing and storing combinations of partial matches in the Petri net. This extension closely follows the suggestion of Ghanem et al. (2004), to use a list of active transitions for determining when transitions are to be activated. Ghanem et al. argue that this provides an efficient basis for computation. The maintenance of such a list of activated transitions requires that combinations are precomputed. Intuitively, the use of precombinations should lower
the computational demands, since an externally activated transition only needs to investigate a priori known valid combinations over the input places, compared to investigating the full, iteratively restricted, Cartesian product of the content in the input places. A closer inspection however reveals that the use of precombinations only moves the combination problem. Instead of deriving valid combinations when a transition is activated, the task is instead carried out when the input of the transition changes. Thus, if changes in the input places of a transition occur more frequently compared to the number of activations of the transition, then efficiency might actually decrease. Furthermore, the memory requirements coupled to keeping a set of valid combinations, for each transition, might become too large. However, in line with Ghanem et al. (2004), it is believed that the use of precombinations can result in increased efficiency. Thus, the following hypothesis is suggested:

**Hypothesis R₆.** It is beneficial to use precombinations of partial matches in transitions in the Petri net based technique.

### 10.3.1 Method

In order to verify the validity of the two hypotheses, the time needed for recognition needs to be compared for the two techniques as well as when using the suggested extension. It is however also necessary that the demands in memory are inspected since memory is a finite resource. In order for hypothesis R₅ to hold it is necessarily the case that the Petri net based technique at most may consume as much time and memory as the rule based technique. In order for hypothesis R₆ to hold, two criteria must be met: (1) the required time needs to be lower when using the extension and (2) the increase in memory consumption must not be too large when using the extension. It is straightforward to determine if the first criteria is met. However, the second criteria require some elaboration.

Although percentages in loss and gain of memory and efficiency can be established, it is hard to assess how much of a loss in memory that is acceptable for a certain gain in time. Loss and gain in consumption can be put in relation to the complexity of the input data. In order for hypothesis R₆ to be considered valid, it is argued that the loss in memory should be at most proportional to a fixed factor of the memory usage of the algorithm without the extension, with respect to the complexity of the input data. Moreover, there also needs to be a gain in efficiency with respect to time.

In order to investigate the hypotheses, it is suggested that the simulated pick pocket scenario should be used. Since there are many perturbations of each scenario setup, the simulated data allows for averages to be formed. Moreover, since the complexity of the setups is increasing, it also allows for the change in efficiency to be measured as a function over the complexity of the input data.
10.3. TIME AND MEMORY CONSUMPTION

10.3.2 Results

Experiment 1

In the first experiment time and memory consumption have been measured for the Petri net based recogniser and for the rule based recogniser. Each of the five setups of the pick pocket scenario has been used. The purpose has been to investigate hypothesis \( R_5 \).

Let us start by investigating the time consumption. Time consumption is influenced by other processes and on the load on the processors on the platform at which experiments are carried out. In order to account for fluctuations, it is suggested that multiple recognition passes are carried out on each input data set. More precisely, it is suggested that time consumption is derived by carrying out 30 runs on each input file for each scenario setup. In each run, the total time spent on recognition can be measured and an average can be formed for each data set. The average of the total time for each of the recognisers on each of the scenarios is presented in Table 10.10.

Table 10.10: Average of the total time consumption of the two algorithms for each scenario setup (in seconds).

<table>
<thead>
<tr>
<th></th>
<th>( S_1 )</th>
<th>( S_2 )</th>
<th>( S_3 )</th>
<th>( S_4 )</th>
<th>( S_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>PN</td>
<td>( \mu )</td>
<td>0.01048</td>
<td>0.05455</td>
<td>0.1159</td>
<td>0.2256</td>
</tr>
<tr>
<td></td>
<td>( \sigma )</td>
<td>0.00288</td>
<td>0.01391</td>
<td>0.0195</td>
<td>0.0537</td>
</tr>
<tr>
<td>RB</td>
<td>( \mu )</td>
<td>0.01094</td>
<td>0.07497</td>
<td>0.1935</td>
<td>0.4710</td>
</tr>
<tr>
<td></td>
<td>( \sigma )</td>
<td>0.00410</td>
<td>0.02589</td>
<td>0.0394</td>
<td>0.1596</td>
</tr>
</tbody>
</table>

As can be observed in Table 10.10, the average total time consumption is similar for the two algorithms on the least complex scenario setup. This indicates that the Petri net based recogniser and the rule based recogniser are equally efficient with respect to the time that is needed for recognition. As the scenario complexity increases, however, the Petri net based technique seems to be more efficient. However, the events of a scenario setup are not normally distributed across simulations, and thus, it is not meaningful to derive standard deviations for scenario setups. Instead, the two recognisers need to be compared on each data set. Each algorithm has been run 30 times for each input file. These results are normally distributed and can thus be used to form 95\% approximated confidence intervals for each algorithm and data set as:

\[
[\overline{x} - 1.96 \frac{s}{\sqrt{n}}, \overline{x} + 1.96 \frac{s}{\sqrt{n}}],
\]

where \( \overline{x} \) denotes the mean, \( s \) the sample standard deviation and \( n \) the size of the sample. For each file there are now three mutually exclusive outcomes: (1) the Petri net based technique consumes significantly less time, (2) the rule based technique consumes significantly less time or (3) there is no significant
difference between the two algorithms. A scoring system has been used that award points according to the three cases, for each file and scenario setup. The results of this comparison are presented in Figure 10.2.

![Figure 10.2: Summed score for the two algorithms with respect to time on each data set.](image)

As can be observed in Figure 10.2, the Petri net based technique clearly perform better in all but the least complex scenario setup, where the rule based technique performs equally well.

However, the memory consumption of the two algorithms also needs to be investigated. Since the memory usage should be the same for each algorithm and setup in each execution, it is not necessary to carry out multiple runs. It is expected that the rule based technique will have a slightly higher memory usage since more partial matches are stored in the network. The maximum and average memory usage, in terms of the number of partial matches, for the two algorithms and for each scenario setup, is presented in Table 10.11.

Table 10.11: Maximum and average memory consumption, proportional to the size of partial matches and WMEs, for the two algorithms for each scenario setup.

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PN</td>
<td>Max</td>
<td>339</td>
<td>965</td>
<td>1380</td>
<td>2145</td>
</tr>
<tr>
<td></td>
<td>$\mu$</td>
<td>125.67</td>
<td>375.22</td>
<td>653.69</td>
<td>1007.4</td>
</tr>
<tr>
<td></td>
<td>$\sigma$</td>
<td>32.70</td>
<td>76.46</td>
<td>68.78</td>
<td>132.2</td>
</tr>
<tr>
<td>RB</td>
<td>Max</td>
<td>1048</td>
<td>2599</td>
<td>3915</td>
<td>6153</td>
</tr>
<tr>
<td></td>
<td>$\mu$</td>
<td>330.62</td>
<td>1001.7</td>
<td>1765.7</td>
<td>2761.0</td>
</tr>
<tr>
<td></td>
<td>$\sigma$</td>
<td>94.04</td>
<td>217.5</td>
<td>194.4</td>
<td>379.0</td>
</tr>
</tbody>
</table>

As can be seen in Table 10.11, the maximum and average memory usage is clearly lower for the Petri net based recogniser, compared with the rule based recogniser. This was expected. To conclude the comparison of the Petri net based technique and the rule based technique, Figure 10.3 illustrates the change in time and memory consumption as the scenario complexity increases.

As can be seen in Figure 10.3 both time and memory consumption increase more quickly for the rule based technique. This can however be a bold state-
10.3. TIME AND MEMORY CONSUMPTION

Figure 10.3: Diagram comparing the time and memory consumption of the two algorithms, with respect to increasing scenario complexity.

The figure illustrates the time and memory consumed by the rule based technique, divided by the time and memory consumed by the Petri net based technique. As can be observed, the additional memory consumption for the rule based technique is quite stable but slightly increasing. The additional time consumption is however increasing, which clearly illustrates that the Petri net based technique is more efficient as complexity increases.
10. RECOGNITION RESULTS

Experiment 2

The second experiment has aimed to investigate hypothesis \( R_6 \): is it beneficial to precombine partial matches? In order for it to be considered beneficial, the time consumption needs to be lower. Moreover, the memory consumption needs to at most grow linearly proportional to the memory usage without the extension. To investigate time consumption, each data set for each scenario has been tested thirty times on each of the two algorithm setups. Table 10.12 presents the average of the total time consumption with and without the extension.

Table 10.12: Average of the total time consumption of the Petri net based technique, with and without the extension, for each scenario setup.

<table>
<thead>
<tr>
<th></th>
<th>( S_1 )</th>
<th>( S_2 )</th>
<th>( S_3 )</th>
<th>( S_4 )</th>
<th>( S_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>PN</td>
<td>( \mu )</td>
<td>0.01048</td>
<td>0.05455</td>
<td>0.1159</td>
<td>0.2256</td>
</tr>
<tr>
<td></td>
<td>( \sigma )</td>
<td>0.00288</td>
<td>0.01391</td>
<td>0.0195</td>
<td>0.0537</td>
</tr>
<tr>
<td>PN+PC</td>
<td>( \mu )</td>
<td>0.01024</td>
<td>0.05035</td>
<td>0.1144</td>
<td>0.2294</td>
</tr>
<tr>
<td></td>
<td>( \sigma )</td>
<td>0.00277</td>
<td>0.01427</td>
<td>0.0192</td>
<td>0.0535</td>
</tr>
</tbody>
</table>

As can be seen in Table 10.12, the Petri net based technique with the extension seems to be a little bit faster in the first three scenario complexity setups. In the two most complex scenario setups the Petri net based technique without the extension seems to be slightly faster. The results are however very similar. To inspect the results further, Figure 10.5 presents the score for the algorithm with and without the extension, when carrying out comparisons on each data set. Scores have been calculated as in the previous experiment, using 95% approximated confidence intervals.

![Figure 10.5: Diagram illustrating the summed score for the Petri net based technique with and without the extension and with respect to time on each data set.](image)

As can be observed in Figure 10.5, the two settings of the Petri net based technique are very similar with respect to time consumption. As the complexity increases, however, the Petri net based technique without the extension seems to
be the more efficient. Although very equal, there may be some benefit with respect to time in the less complex scenarios. It however also needs to be inspected how the memory usage changes with and without the extension. Table 10.13 presents the maximum and average memory consumption of the technique with and without the extension, for each scenario setup.

Table 10.13: Maximum and average memory consumption, proportional to the size of partial matches, for the Petri net based technique with and without the extension.

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PN Max</td>
<td>339</td>
<td>965</td>
<td>1380</td>
<td>2145</td>
<td>3674</td>
</tr>
<tr>
<td></td>
<td>$\mu$</td>
<td>125.67</td>
<td>375.22</td>
<td>653.69</td>
<td>1007.4</td>
</tr>
<tr>
<td></td>
<td>$\sigma$</td>
<td>32.70</td>
<td>76.46</td>
<td>68.78</td>
<td>132.2</td>
</tr>
<tr>
<td>PN+PC Max</td>
<td>399</td>
<td>1106</td>
<td>1559</td>
<td>2455</td>
<td>4333</td>
</tr>
<tr>
<td></td>
<td>$\mu$</td>
<td>151.06</td>
<td>452.12</td>
<td>774.27</td>
<td>1183.6</td>
</tr>
<tr>
<td></td>
<td>$\sigma$</td>
<td>40.04</td>
<td>95.45</td>
<td>73.53</td>
<td>168.6</td>
</tr>
</tbody>
</table>

As can be seen in Table 10.13, the memory consumption increases slightly when using the extension. The increase in memory consumption does not seem to be growing very much with scenario complexity. It must, however, be noted that a precombiner was only used in a single transition, since it is only one transition that has more than one input place. Transitions with only one input place would not have gained anything by using explicit precombiners. To clearly investigate the effects of the extension with increased complexity, Figure 10.6 illustrates the time and memory consumption relative to the scenario complexity.

Figure 10.6: Diagram comparing the time and memory consumption of the Petri net based technique with and without the extension, and with respect to increasing scenario complexity.
As can be observed in Figure 10.6, both of the conditions that were defined for the hypothesis fails, although only slightly. In other words, there is not always a gain in speed, and the loss in memory is not proportional to a factor of the algorithm without the extension. It can be seen in the figure that the increase in memory usage grows with complexity. Again, it is not known that the complexity is linearly increasing over the scenarios, thus, Figure 10.7 illustrates the proportional difference with and without the extension.

![Figure 10.7](image)

**Figure 10.7:** Diagram illustrating the proportional differences in time and memory consumption of the Petri net based technique with and without the extension, and with respect to increasing scenario complexity.

As can be observed in Figure 10.7, the differences are very small. The extra memory usage when precombining partial matches is approximately 1.2 times larger compared to not using the extension, for each of the scenario complexity settings. The time consumption is very similar, perhaps slightly increasing for the extended version.

### 10.3.3 Conclusions

It has been shown that the Petri net based technique requires less time than the rule based technique, as the complexity of the input data increases. For the least complex scenario, they are however equally efficient with respect to time. It has also been shown that the Petri net based technique requires less memory compared with the rule based technique for all scenarios that have been investigated. It is thus argued that hypothesis $R_5$ holds.

It has also been shown that for scenarios with low complexity, then there can be benefits with respect to time to precompute combinations of partial matches. As the scenario complexity increases however, the benefits are lost. Moreover, the increased memory consumption does not grow proportional to
the consumption of the algorithm without any extension; rather, it possibly
grows exponentially with some small factor. This indicates that it is not always
beneficial to use precombinations, and thus, it is argued that hypothesis $R_6$
should be rejected. Of course, the use of precombinations can be beneficial
on occasion. However, such benefits must be determined empirically before
including an extension such as the one that has been investigated.

10.4 Applicability

So far it has been shown that both the Petri net based technique and the rule
based technique can be used for recognising situations in simulated data of
varying complexity. It has also been shown that the Petri net based technique
can be used more efficiently than the rule based technique with respect to the
consumption of time and memory. One additional aspect of great importance
is to investigate if the Petri net based technique can be used for recognising real
situations in real data. The end goal is to have techniques that are applicable in
real world systems, and this involves recognising situations in real world data.

Although the simulator and the pick pocket scenario have been developed
to contain large amounts of noise, many objects and varying behaviours, there
may still be large differences between artificial and synthetic data. Objects may
behave differently, there may be other sources of noise or perhaps the models
have been too naïve. Still, situation recognition operates at a symbolic level
with relations that have been extracted from object level data. Thus, it is be-
lieved that given a suitable symbolic language and suitable definitions of inter-
esting situations, the Petri net based technique should be usable in real world
situations. The following hypothesis is thus suggested:

$Hypothesis \ R_7$. Petri nets can be used for recognising real situations
using real data.

10.4.1 Method

Although there may be many real world data sets containing many interesting
complex situations, such data sets are often unlabelled. This in turn results
in problems with estimating the performance of the technique that is being
investigated. One way to address this problem is to add artificial situations on
top of the real data and investigate if these can be recognised. Again, this is not
much better than working with purely simulated data since there may be many
intrinsic aspects of the real world that are missed. Another possible approach
would be to use actors to play certain scenarios in real world situations, whilst
also recording data of these scenarios. Still, it is hard to determine how much
an acted scenario resembles a real situation. It is instead argued that real world
data with real world situations should be used for investigating hypothesis $R_7$.
The piloting scenario fits this purpose well. It may seem as an uninteresting
task to recognise this type of situation since these are trivial to find due to the fact that piloting boats distribute their identity. Still, piloting situations are real situations and thus serve the purpose of the present investigations nicely.

The piloting scenario has already been described in Chapter 9. Besides the AIS data, three rectangular area objects have been inserted into the framework. One of these represents an area west of Gothenburg: the designated area for boarding. A single large area has been created to cover multiple pilot rendezvous areas and anchoring areas. This area is in the vicinity of lighthouses Vinga and Trubaduren. The area has been estimated using Eniro sea charts\(^1\). The second area represents the harbour area which inbound vessels are headed for. The area is rectangular and covers Älvsborg RO/RO harbour, Skandia container harbour and Skarviks oil harbour. The last area represents Tångudden, a small harbour where pilot boats, towing boats and bunker vessels are located.

The template for the piloting situation introduced in Chapter 9 can be converted into a Petri net as illustrated in Figure 10.8.

![Petri net](image)

**Figure 10.8:** A Petri net representing a template describing a piloting situation.

### 10.4.2 Results

In this experiment the maritime AIS data has been used for investigating the validity of hypothesis $R_7$. The scenario consists of AIS data collected between 15 and 16 February 2010 in the archipelago around Gothenburg. As discussed in Section 10.1.3, the AIS data has been preprocessed to extract a number of symbolic relations between vessels and other objects in the environment. The resulting events describing the symbolic relations have been inserted into the Petri net for modelling piloting situations illustrated in Figure 10.8.

In total, the Petri net identified 14 situations. These situations have been analysed together with experts from our industrial partner Saab AB, in order to determine their validity. It has been determined that 7 of the situations are true positives and 7 are false positives. Five of the false positives represent meetings between bunker boats, fresh water ships and pilot boats (these are also located in the same area as the pilot boats). The remaining two situations represent vessels that did not use piloting assistance. This however only tells us something

---

\(^1\)Available at [http://www.eniro.se](http://www.eniro.se)
10.4. APPLICABILITY

about the precision. In order to investigate recall, a simpler Petri net has been used to find promising situations that could represent partially fulfilled piloting situations. The Petri net for this purpose is illustrated in Figure 10.9.

InHarbour(x1,a2)\rightarrow\text{Close}(x_1,x_2)\rightarrow\text{InArea}(x_1,a_1)

Figure 10.9: A small Petri net for identifying any potential missed piloting situations.

In total, 162 potential situations where found. These situations have been inspected manually to determine their status as potential piloting situations that were not found by the more complex Petri net. Most of the matches again represent bunker ships and towing vessels. One missed piloting situation was however also found. The missed vessel represents a container freighter with a length of over 200 meters. It is thus quite possible that boarding relations are missed due to the size of the ship. Perhaps the AIS transmitter is located in the stern, whilst boarding could have proceeded in the bow. The relational data has also been inspected and no close relation between the freighter and a piloting boat could be found. Table 10.14 summarises the findings.

Table 10.14: True positives, false positives, false negatives, precision and recall for the maritime piloting scenario.

<table>
<thead>
<tr>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>1</td>
<td>0.5</td>
<td>0.875</td>
</tr>
</tbody>
</table>

As can be seen, the precision is 0.5 and recall is 0.875. Although there are very few samples, this experiment shows that the Petri net based technique can be used for recognising real situations in real data. It is also possible that there might have been more piloting situations in the data which were not found by the simple Petri net neither. To conclude this section, Figure 10.10 illustrates one of the identified piloting situations.

10.4.3 Conclusions

Seven out of eight piloting situations were identified in two days of AIS data. Thus, it is has been shown that hypothesis $R_7$ holds and that Petri nets can be used for recognising real situations using real data. The precision and recall on the maritime scenario could most likely have been increased by refining the areas that are used and by carrying out more refined closeness tests that considers typical shapes for vessels (most are long but not so wide).
10.5 Chapter summary

In this chapter we have empirically investigated the efficiency of the Petri net based technique for situation recognition. It has been shown that Petri nets can be used for situation recognition and that they can be used as efficiently as a rule based technique with respect to time. It has even been shown that as the complexity of the input data increases, the Petri net based technique can be used more efficiently than the rule based technique. Moreover, it has also been shown that the memory demands of the Petri net based technique is lower.

In the chapter it has also been shown that it is not always beneficial to precompute valid combinations of tokens. There may be some gains in time on occasion, but this also comes at an increase in memory usage. In order for such extensions to be used, it can be important to empirically investigate the effects of the time-memory trade off in the specific case.
Chapter 11
Learning results

This chapter addresses the second part of the fifth research objective in this thesis, namely, carrying out empirical investigations on the use of genetic algorithms for learning Petri net situation templates. The content of the chapter is largely based on two publications: Dahlbom and Niklasson (2009) and Dahlbom et al. (2010b). The contributions of the chapter consist of:

- Empirical results concerning the use of genetic algorithms for learning Petri net situation templates.

11.1 Experimental setup

The simulator presented in Chapter 8 has been used to construct 30 data sets of the pick pocket scenario presented in Chapter 9. These data sets have been used for evolving Petri nets and for evaluating evolved solutions.

11.1.1 Notation

The notation used when presenting results in this chapter is shown in Table 11.1

11.1.2 Scenario properties

Each data set is based on a 15 minute scenario. As discussed in Chapter 9, there are two spawn zones in the scenario. During simulation, there has been a 35% chance of creating one pedestrian and a 10% chance of creating two pedestrians, in each of the two spawn zones every 5 seconds. In case a pick pocket situation is not in progress, then there has been a 30% chance of initiating one, every 5 seconds. Events have been extracted from the resulting data files for the following types of relations: Close$(x, y)$, Approach$(x, y)$, Intercept$(x, y)$, Speedup$(x)$ and Slowdown$(x)$. Please refer to Chapter 10 for more details about the scenario and the relation extraction procedures.
11. LEARNING RESULTS

Table 11.1: The notation used when presenting evolutionary results.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG</td>
<td>Bit genomes have been used.</td>
</tr>
<tr>
<td>CG</td>
<td>Complex genomes have been used.</td>
</tr>
<tr>
<td>DCG</td>
<td>Dynamic complex genomes have been used.</td>
</tr>
<tr>
<td>+B</td>
<td>Bootstrapping has been used.</td>
</tr>
<tr>
<td>-S</td>
<td>The initial population has not been seeded with valid Petri nets.</td>
</tr>
<tr>
<td>F1</td>
<td>Fitness has been calculated using Equation 6.4.</td>
</tr>
<tr>
<td>F2</td>
<td>Fitness has been calculated using Equation 6.7.</td>
</tr>
<tr>
<td>F3</td>
<td>Fitness has been calculated using Equation 6.6.</td>
</tr>
<tr>
<td>TP</td>
<td>True positives.</td>
</tr>
<tr>
<td>FP</td>
<td>False positives.</td>
</tr>
<tr>
<td>FN</td>
<td>False negatives.</td>
</tr>
<tr>
<td>μ</td>
<td>Mean.</td>
</tr>
<tr>
<td>σ</td>
<td>Sample standard deviation.</td>
</tr>
<tr>
<td>P</td>
<td>Precision (calculated according to Equation 6.3).</td>
</tr>
<tr>
<td>R</td>
<td>Recall (calculated according to Equation 6.2).</td>
</tr>
</tbody>
</table>

Statistics have been gathered about the total number of pedestrians, the average number of active pedestrians, the total number of intentionally instantiated situations and the total number of events. The statistics extracted from the constructed data files are shown in Table 11.2.

Table 11.2: Scenario statistics

<table>
<thead>
<tr>
<th></th>
<th>μ</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of created pedestrians</td>
<td>514.3</td>
<td>24.1</td>
</tr>
<tr>
<td>Average number of active pedestrians (0 - 900 sec)</td>
<td>25.58</td>
<td>4.16</td>
</tr>
<tr>
<td>Average number of active pedestrians (60 - 900 sec)</td>
<td>26.54</td>
<td>1.05</td>
</tr>
<tr>
<td>Total number of instantiated pick pocket situations</td>
<td>11.34</td>
<td>1.03</td>
</tr>
<tr>
<td>Total number of extracted events</td>
<td>13190</td>
<td>1622</td>
</tr>
</tbody>
</table>

11.1.3 Evolutionary process

The first 20 data sets have been used for evolution and the remaining 10 for evaluation. In each evolutionary run, 2 out of the 20 data sets have been randomly selected to evolve a population consisting of 60 individuals. The reason for this approach is that the evolutionary process is very time consuming. A single evolutionary run of 250 generations for 60 individuals and two input data files can last from a couple of hours to several days. It has thus not been considered feasible to use more than two files in each run. Neither has it been considered meaningful to carry out for example cross validation. The reason for high time demands reads as follows: for two input files, the 60 Petri nets need to parse 30 minutes of data in each generation. Parsing 30 minutes of data often takes less time than 30 minutes. Still, multiply by 250 generations and multiple evolutionary runs, and the required processing time grows quickly.
The performance on the recognition task has been evaluated for each generation in each evolutionary experiment, by feeding all events in the two files to all individuals. After all events have been processed, the precision and recall of each individual has been determined with respect to the truth files containing all intentionally instantiated patterns. As an example, a specific Petri net has recognised 2000 situations after having parsed all events in the two files. 20 of the recognised situations are part of a set of 30 intentionally instantiated situations summed from the two data sets. There are thus 20 true positives (TP), 1980 false positives (FP) and 10 false negatives (FN). Recall and precision can now be calculated according to Equation 6.2 and Equation 6.3 as:

\[ R = \frac{TP}{TP + FN} = \frac{20}{30} = 0.667, \]
\[ P = \frac{TP}{TP + FP} = \frac{20}{2000} = 0.01. \]

The fitness for the individual can then be determined using Equation 6.4 as:

\[ f = 0.45P + 0.45R + f_b = 0.45 \cdot 0.01 + 0.45 \cdot 0.667 + f_b = 0.30465 + f_b, \]

where \( f_b \) varies between 0 and 0.1 according to Equation 6.10. The non-complex network contribution of the base fitness \( f_b \) depends on a complexity limit \( n_{cl} \). This value has been set to 0.06 in all experiments except where otherwise stated, i.e. the time to update the marking of a Petri net when processing an event may at most take 0.06 seconds. The average fitness, recall and precision of the population as a whole have also been calculated together with their respective sample standard deviations.

Each evolutionary run has been allowed to run for at most 250 generations. Early termination has however been used in runs where: (1) the fitness of the best individual has been above 0.2 for 25 generations and (2) the standard deviation of the fitness for the best individuals over 25 consecutive generations has been 0. The reasoning behind the use of early termination is that in case the best individual has above 0.2 fitness, then it represents a solution that is able to recognise at least some of the situations. However, in case the fitness is also too stable for too long, then the population has converged on a local optimum which it is not likely to escape. Early termination has thus been applied in order to shorten experimentation times\(^1\).

Lastly, the size of the genomes that have been evolved has been set to 10 transitions, 10 places and 3 match variables, except where otherwise stated. For the dynamic complex genome representation, these size specifications are only used to initialise genomes in the population of the first generation in an evolutionary run.

\(^1\)A few experiments have been carried out where a population has been allowed to evolve for 1500 generations. None of these evolutionary runs have however succeeded in escaping local optima, wherefore the use of early termination has been considered warranted.
11.1.4 Evolutionary parameters

As suggested in Chapter 6, elitism selection has been used to clone 5% of the present population when creating a new population, in order to assure that good solutions are not lost during evolution. The remaining 95% have been created by using roulette wheel selection to select two parents. These have then been combined using single point crossover, to result in two offspring. For genomes of static size, the split point has been selected randomly from a range consisting of the length of genomes, and for genomes of dynamic size, crossover has been carried out as described in Chapter 6.

Besides cross over, each individual in the new population that has not been selected using elitism has also been subject to the genetic mutation operator. The mutation rate $\delta$ has been allowed to vary between 2 and 12, i.e. $\delta_{\text{min}} = 2.0$ and $\delta_{\text{max}} = 12.0$, according to the discussion in Chapter 6. The limits for varying the rate, $\sigma_{\text{min}}$ and $\sigma_{\text{max}}$, have been set to 0.0001 and 0.01, respectively. In case the sample standard deviation of the average of the average fitness of the population for the past 20 generations has been below or above $\sigma_{\text{min}}$ and $\sigma_{\text{max}}$, then delta has been changed by $-1.0$ and $+1.0$, respectively.

In representations with genomes of dynamic size, i.e. the dynamic complex genome, then a chromosome mutation rate of 0.01 has been used. In other words, for each chromosome in each genome, there has been a 0.01 chance of either removing or adding a gene during each evolutionary step. Genomes selected with elitism selection have not been subject to any sort of mutation.

11.2 Promoting precision or recall

The fitness function in a genetic algorithm constitutes the main source of preference bias. It should thus guide the evolutionary search towards promising solutions in the search landscape. It can often be a laborious task to design a fitness function that effectively provides guidance. In the present case, the task that solutions are being evolved for aims at recognising interesting situations as well as possible, whilst not classifying uninteresting situations as interesting, e.g. a solution should have high recall and high precision. A fitness function was therefore suggested in Chapter 6, which promotes precision and recall equally.

The characteristics of the search landscape are however unknown and it could be more beneficial to promote one of the two measures higher than the other, with respect to the resulting solutions and with respect to the number of generations needed to find suitable solutions. As discussed in the previous chapter, Lingard and Lambert (2008) suggest that the $f$-value can be used as a combined measure of precision and recall. The $f$-value is the geometric mean of precision and recall and it was defined in Equation 10.1. The $f$-value can be defined as a function over a domain consisting of two variables: $F_v(P, R)$, where $P, R \in [0..1]$. Figure 11.1 illustrates the $f$-value function plotted over the domain $[0, 0..1, 1]$. 
As can be seen, the highest f-value occurs at the ridge where both precision and recall are as high as possible. For every solution with $P > R$ or $R > P$, there possibly exists a solution with $P \approx R$, which by definition has a better f-value. Hence, it can be argued that a setting that promotes precision and recall equally should be beneficial compared to settings where either of the two is considered more important. The following hypothesis is suggested.

**Hypothesis $L_1$:** It is beneficial to promote precision and recall equal when evolving Petri nets for situation recognition.

### 11.2.1 Method

To investigate the validity $L_1$, it needs to be investigated if it is beneficial to promote either of precision or recall higher than the other. Two alternate fitness functions were suggested in Equation 6.6 and Equation 6.7. It is suggested that the results of the evolutionary process when using these two functions are compared with the results of the evolutionary process when using the fitness function defined in Equation 6.4. To allow for only the fitness functions to be compared, it is argued that an experiment should be carried out in which seeding and bootstrapping are not used. Moreover, it is also argued that the bit genome representation should be used, since it puts fewer constraints on the search landscape, compared with the other two representations. Moreover, considering that the bit genome representation more closely resembles the common type of representation that is used in genetic algorithms, results that are achieved can possibly be considered more generally applicable. In order to allow for a quite large search landscape to be searched, the complexity limit $nc_l$ (refer to Section 6.2.4) is argued to be set to 0.06 seconds.
11.2.2 Results

In the experiment, 45 evolutionary runs have been carried out with the bit genome. In the first 15 runs, fitness was calculated using Equation 6.4, in runs 16 to 30, fitness was calculated using Equation 6.6 and in the remaining 15 runs, fitness was calculated using Equation 6.7. The purpose of the experiment was to investigate hypothesis $L_1$. Table 11.3 presents average of the best fitness, precision and recall achieved during evolution.

<table>
<thead>
<tr>
<th></th>
<th>Fitness</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG+F1-S</td>
<td>0.5650</td>
<td>0.4793</td>
<td>0.1231</td>
</tr>
<tr>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0332</td>
<td>0.0125</td>
<td>0.0133</td>
</tr>
<tr>
<td>BG+F2-S</td>
<td>0.2790</td>
<td>0.2508</td>
<td>0.0193</td>
</tr>
<tr>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0387</td>
<td>0.0086</td>
<td>0.0110</td>
</tr>
<tr>
<td>BG+F3-S</td>
<td>0.8558</td>
<td>0.7292</td>
<td>0.2101</td>
</tr>
<tr>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0385</td>
<td>0.0129</td>
<td>0.0141</td>
</tr>
</tbody>
</table>

As can be seen in Table 11.3 the achieved fitness differs largely. However, these values cannot be compared with each other since they are calculated differently. It does however not seem to be any large differences in neither best precision nor recall. Average recall does drop a little when promoting precision and there is not a similar gain in precision. However, the results presented in Table 11.3 only compare the three fitness functions with respect to performance on data that has been used during evolution. It is also of importance to achieve solutions that are able to generalise to unseen data. Thus, the Petri nets with the highest fitness from each of the three sets of runs have been evaluated on the evaluation data set. The results are presented in Table 11.4.

<table>
<thead>
<tr>
<th></th>
<th>TP</th>
<th>TP+FN</th>
<th>TP+FP</th>
<th>Precision</th>
<th>Recall</th>
<th>F-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG+F1-S</td>
<td>$\mu$</td>
<td>11.4</td>
<td>11.5</td>
<td>371.8</td>
<td>0.0317</td>
<td>0.9923</td>
</tr>
<tr>
<td></td>
<td>$\sigma$</td>
<td>0.967</td>
<td>1.08</td>
<td>78.26</td>
<td>0.0064</td>
<td>0.0243</td>
</tr>
<tr>
<td>BG+F2-S</td>
<td>$\mu$</td>
<td>11.4</td>
<td>11.5</td>
<td>371.8</td>
<td>0.0317</td>
<td>0.9923</td>
</tr>
<tr>
<td></td>
<td>$\sigma$</td>
<td>0.967</td>
<td>1.08</td>
<td>78.26</td>
<td>0.0064</td>
<td>0.0243</td>
</tr>
<tr>
<td>BG+F3-S</td>
<td>$\mu$</td>
<td>11.4</td>
<td>11.5</td>
<td>371.8</td>
<td>0.0317</td>
<td>0.9923</td>
</tr>
<tr>
<td></td>
<td>$\sigma$</td>
<td>0.966</td>
<td>1.08</td>
<td>78.26</td>
<td>0.0064</td>
<td>0.0243</td>
</tr>
</tbody>
</table>

As can be seen in Table 11.4, there are no differences when comparing the performance achieved on the evaluation data. Precision, recall and f-value of the three best evolved Petri nets with the three fitness functions achieve identical results. The weighting of precision and recall when used in a fitness function does not seem to play an important role in the resulting performance.
11.2.3 Conclusions

There are clearly no observed disadvantages of using a fitness function where precision and recall are promoted equally, compared to promoting one of the two measures higher. No advantages have been observed either, and it can therefore not be concluded that it is beneficial to promote precision and recall equally. Hypothesis \( L_1 \) can therefore neither be accepted nor rejected.

11.3 Initial seeding

There are some basic requirements that can be put on a Petri net for it to be able to recognise something. Specific places are used as initiators of matching and other places are used for finding complete matches. Through the incorporation of observations, matching proceeds by moving tokens from initial places to match places. There thus needs to be at least one valid path from start to match places. Moreover, there needs to be at least one input and one match place.

In the usual way of using genetic algorithms the initial population is generally created to consist of individuals with randomised genomes, e.g. each bit in a bit string is drawn from a uniform distribution \( \{0, 1\} \). This closely follows the Darwinian view however; many of the individuals are likely to not represent valid Petri nets. Thus, with a purely randomly created initial population, many generations are likely to be spent on finding Petri nets that possibly can be the starting point of good solutions. It was therefore suggested in Chapter 6 that the initial population could be seeded with only valid, but still random, individuals. The time spent on evaluating invalid Petri nets can possibly be lowered by using initial seeding.

A possible risk, however, is that the use of too much domain knowledge severely could impact the evolutionary process since the search space is biased. Nevertheless, inheriting useless properties is not likely to be beneficial on its own, and thus, initial seeding should allow for more quickly evolving promising individuals. The following hypothesis is suggested:

Hypothesis \( L_2 \). It is beneficial to seed the initial population with valid solutions, when evolving Petri nets for situation recognition.

11.3.1 Method

To investigate the validity of hypothesis \( L_2 \), it needs to be investigated if it is beneficial, with respect to the number of required generations needed to find promising individuals, to use initial seeding when evolving Petri nets for situation recognition. It, however, also needs to be investigated if the resulting solutions are of the same quality as when not using initial seeding. If the quality is lower, then it is not beneficial to use initial seeding. Since the choice of fitness function does not seem to affect the results, a fitness function promoting precision and recall equally will be used. To clearly highlight the effects of initial
11. LEARNING RESULTS

seeding it is also argued that bootstrapping should not be used since the quality of Petri nets used for bootstrapping can affect the results. It is argued that bit genomes should be used since it puts fewer restrictions on the search space, and since it also is the custom form of representation to use. Thus, results can possibly generalise better to other settings given that direct vector transferable genomes are deployed extensively when using genetic algorithms.

11.3.2 Results

Bit genomes have been used to carry out 30 evolutionary runs. In the first 15 runs the initial population was initialised with random genomes and in the remaining 15 runs initial seeding was used. In order to investigate hypothesis $L_2$, it is of importance to assure that initial seeding does not have a negative impact on performance. Table 11.5 illustrates maximum and average best fitness of the evolved solutions.

Table 11.5: Best fitness without and with initial seeding using bit genomes.

<table>
<thead>
<tr>
<th></th>
<th>BG-S</th>
<th>BG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total evolutionary runs</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Max</td>
<td>0.5768</td>
<td>0.5715</td>
</tr>
<tr>
<td>$\mu$</td>
<td>0.5119</td>
<td>0.5231</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>0.1033</td>
<td>0.0728</td>
</tr>
</tbody>
</table>

As can be seen in Table 11.5, fitness of the best individuals are similar when using seeding and when not using seeding. It can also be observed that the average of the best fitness over all individuals also is similar. The results presented in Table 11.5 are however only based on the data that was used for evolution. The main objective is to achieve general solutions with high precision and recall. The best evolved individual for both of the settings has thus been evaluated on the evaluation data set. The results are presented in Table 11.6.

Table 11.6: Precision and recall results for the best evolved individual on the evaluation data set, without and with initial seeding, using the bit genome representation.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F-value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\mu$</td>
<td>$\sigma$</td>
<td>$\mu$</td>
</tr>
<tr>
<td>BG-S</td>
<td>0.05679</td>
<td>0.01451</td>
<td>0.93886</td>
</tr>
<tr>
<td>BG</td>
<td>0.05033</td>
<td>0.01293</td>
<td>0.96692</td>
</tr>
</tbody>
</table>

As can be observed in Table 11.6, the results are quite similar. The Petri net evolved without initial seeding has slightly better precision and f-value, whilst the Petri net evolved with seeding has slightly better recall. It thus seems as initial seeding does not have a negative impact on the resulting performance.
11.3. INITIAL SEEDING

The purpose of using initial seeding is to possibly lower the number of generations required until finding promising individuals. In case fitness is above 0.1, then a solution has been found which correctly classifies at least one interesting situation. This can be used to measure the number of generations required until finding solutions that show merit. Table 11.7 presents the number of generations required to find promising solutions.

Table 11.7: The number of generations required until fitness > 0.1, without and with initial seeding for the bit genome representation.

<table>
<thead>
<tr>
<th></th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG-S</td>
<td>60.9</td>
<td>67.7</td>
<td>2</td>
<td>209</td>
</tr>
<tr>
<td>BG</td>
<td>13.4</td>
<td>17.0</td>
<td>0</td>
<td>63</td>
</tr>
</tbody>
</table>

As can be seen in the table, there seems to be some advantage with respect to the number of generations required until finding promising solutions, to use initial seeding. However, the extent of the advantage is not stable since the standard deviations are rather large. This suggests that more evolutionary runs are needed. A second experiment has thus been carried out. This experiment has focused on measuring the number of generations until finding at least one individual with a fitness value greater than 0.1. Moreover, since the focus is to lower the time spent until finding promising starting points, a complexity limit of 0.03 has been used, i.e. \(nc_l = 0.03\). In total, 60 evolutionary runs have been carried out, 30 for each setting. Since more runs have been carried out, Equation 10.2 can be used to approximate 95% confidence intervals with respect to the number of required generations until finding solutions with fitness greater than 0.1. Table 11.8 presents the extended results with respect to the number of generations required until finding promising individuals.

Table 11.8: The number of generations required until fitness > 0.1 for the bit genome representation without initial seeding (BG-S), and for the bit genome representation with initial seeding (BG), on an extended experiment with more evolutionary runs (30 for each setting).

<table>
<thead>
<tr>
<th></th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>Min</th>
<th>Max</th>
<th>(Min_{95%})</th>
<th>(Max_{95%})</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG-S</td>
<td>57.6</td>
<td>51.0</td>
<td>4</td>
<td>183</td>
<td>39.0</td>
<td>76.1</td>
</tr>
<tr>
<td>BG</td>
<td>22.9</td>
<td>29.6</td>
<td>3</td>
<td>144</td>
<td>12.3</td>
<td>33.5</td>
</tr>
</tbody>
</table>

As can be observed in Table 11.8, the number of generations until finding promising solutions is significantly lower when using initial seeding, compared to not using initial seeding. This statement is based on the fact that the two 95% approximated confidence intervals do not overlap and that the interval for the seeded runs is lower than the interval for the non seeded runs.
11.3.3 Conclusions

It has been observed that the number of generations until finding promising solutions can be lowered through the use of initial seeding. Moreover, there are no observed disadvantages of using initial seeding, since the resulting performance is similar compared to not using initial seeding. Although small, the decreased number of generations until finding promising individuals can thus be considered slightly beneficial. Hence, it is argued that hypothesis $H_2$ holds. Thus, when evolving Petri nets for situation recognition, it is considered to be slightly beneficial to use domain knowledge to seed the initial population with valid individuals.

11.4 Genome representations

The choice of a suitable genome representation is very important when using genetic algorithms since it determines what can be evolved. If something cannot be represented, then it cannot be learned. Moreover, the genome representation also provides a source of restrictive bias. This is an important aspect in machine learning, c.f. Mitchell (1997). The task of balancing between representative power and bias is unavoidable. The classical form of representation is to use genomes consisting of vectors of bits or numbers that is translated to problem solutions. The construction of a suitable mapping from bits to solutions is however not always a straightforward task. The Petri nets that are being evolved in this thesis consist of a graphical structure, where half of the nodes represent predicates of varying arity. This specific task thus involves learning structure and symbolic representations, and when using genetic algorithms for this task it is thus necessary to have suitable mapping functions between genomes and Petri nets. Three genome representations were suggested in Chapter 6.

The bit genome representation uses the typical method for constructing genomes, where a string of bits is used. Subvectors of the bit string are then mapped onto specific parts of an a priori sized Petri net. Resorting to bits however introduces the problem of solutions that cannot be instantiated, i.e. specific ranges of subvectors translated to some part of a Petri net may lie outside of the allowed. As an example, if there are 5 distinct events, then this would require 3 bits. 3 bits however allow for 8 distinct events to be represented, wherefore $3/8 \cdot n_t$ solutions cannot be instantiated, where $n_t$ represents the number of transitions. Another problem when using bit genomes for evolving complex structures is that distinct subparts may be deconstructed without any guidance.

The complex genome representation addresses these problems by having each gene representing a specific substructure of a Petri net, e.g. a gene representing a place, another representing a transition, and so on. This keeps the internal structure but evolves the bonds. Moreover, the content of the specific substructures are evolved within allowed limits. Still, two problems remain. The first problem is related to the number of edges in a Petri net representing
an interesting situation. The number of activated edges is large in the bit and complex representations, since these make use of connectivity matrices for representing edges. This could be solved by not using a uniform distribution in the construction of genomes, however, the search space coupled to connectivity matrices is by default large, whilst edge activation of an interesting situation is not likely to be so. Secondly, genomes are typically depicted to be of static size. This can become an issue when evolving graphs for representing unknown situations, since the size must be decided a priori. If a too small structure is chosen, then the target concept cannot be learned. If a too large structure is chosen, then the time needed for learning grows rapidly.

The dynamic complex genome representation, suggested in Chapter 6, addresses these problems by means of using specific genes for representing edges, and by also allowing the size of genomes to be learned. The dynamic complex genome representation consists of a set of chromosomes, which in turn consist of genes that represent distinct substructures of a Petri net. Structural parts are thus kept together to achieve a more directed search. It is believed that the dynamic complex genome representation represents the most efficient of the three forms of representation however, this is only a hypothesis. It may be the case that it restricts the search space too much, thereby not allowing target concepts to be represented. The bit genome representation provides the least restrictions, whilst the complex genome representation lay in between the other two on a restriction scale. The following hypothesis is however suggested:

**Hypothesis \( L_3 \).** It is beneficial to have a dynamic complex genome representation when evolving Petri nets for situation recognition.

### 11.4.1 Method

To investigate hypothesis \( L_3 \), it is necessary to investigate if it is beneficial with respect to the time needed to learn Petri nets, to use a dynamic complex genome representation. In order for the hypothesis to hold, the resulting solutions when using the dynamic complex genome representation however also needs to be at least of the same performance as solutions evolved with the other two representations. Again, a fitness function promoting precision and recall equally will be used, since it performs on par with the other two fitness functions. In Section 11.3 it was argued that it is beneficial to use initial seeding when evolving Petri nets for situation recognition. Hence, it is argued that initial seeding too should be used when carrying out the present investigation. Again, it is suggested that bootstrapping should not be used, in order to alleviate the risk of achieving results that are affected by the quality of Petri nets used for bootstrapping. To be consistent with previous experiments, the complexity limit \( n_{cl} \) has been set to 0.06 seconds.
11.4.2 Results

In this experiment 45 evolutionary runs have been carried out. In the first 15 runs the bit genome representation was used. In runs 16-30 the complex genome representation was used. Finally, in the last 15 runs the dynamic complex genome representation was used. The purpose of this experiment has been to investigate the validity of hypothesis \( L_3 \). First, it is necessary to investigate the performance of solutions evolved with the three different genome representations. Table 11.9 shows the fitness of the best evolved solutions, as well as the average of the best evolved solutions, for each of the three representations.

Table 11.9: Best and average of the best fitness for bit genomes (BG), complex genomes (CG) and dynamic complex genomes (DCG).

<table>
<thead>
<tr>
<th></th>
<th>BG</th>
<th>CG</th>
<th>DCG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total evolutionary runs</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Max</td>
<td>0.5673</td>
<td>0.5673</td>
<td>0.5792</td>
</tr>
<tr>
<td>Best fitness</td>
<td>μ 0.5217</td>
<td>0.5593</td>
<td>0.5580</td>
</tr>
<tr>
<td></td>
<td>σ 0.1008</td>
<td>0.0054</td>
<td>0.0105</td>
</tr>
</tbody>
</table>

As can be seen in Table 11.9, the dynamic complex representation is able to achieve similar fitness as the other two representations. It even performs slightly better. The complex genome representation and the dynamic complex genome representation achieve slightly higher average best fitness, compared to the bit genome representation. The fitness results are however only based on performance on the data sets that have been used for evolution. In order to really compare the quality of evolved solutions, the best evolved Petri net, for each of the three representations, have been compared with each other using the evaluation data set. These results are shown in Table 11.10.

Table 11.10: Precision, recall and f-value results for the best evolved individual on the evaluation data set, for each of the three genome representations.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F-value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>μ</td>
<td>σ</td>
<td>μ</td>
</tr>
<tr>
<td>BG</td>
<td>0.0317</td>
<td>0.0064</td>
<td>0.9923</td>
</tr>
<tr>
<td>CG</td>
<td>0.0317</td>
<td>0.0064</td>
<td>0.9923</td>
</tr>
<tr>
<td>DCG</td>
<td>0.0580</td>
<td>0.0129</td>
<td>0.9247</td>
</tr>
</tbody>
</table>

The dynamic complex genome representation performs slightly better compared with the other two forms of representation. Recall drops a little, while precision and f-value increases slightly. The differences are however not very large. Moreover, it may simply be the case that it is very hard to find those slightly better solutions. Given a large number of evolutionary runs, it is possible that similar solutions also can be found using the other two forms of rep-
11.4. GENOME REPRESENTATIONS

representation as well. There may however be benefits with respect to the number of generations that are needed until finding promising individuals.

As previously argued, a solution with fitness greater than 0.1 represents a solution in which something was recognised. Let us therefore, compare the three genome representations with respect to how many generations that are required until promising individuals are found. Table 11.11 shows the number of generations required to evolve promising solutions, for each of the three representations.

Table 11.11: The number of generations required to reach fitness > 0.1, for bit genomes (BG), complex genomes (CG) and dynamic complex genomes (DCG).

<table>
<thead>
<tr>
<th></th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG</td>
<td>18.0</td>
<td>17.7</td>
<td>2</td>
<td>64</td>
</tr>
<tr>
<td>CG</td>
<td>1.7</td>
<td>2.1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>DCG</td>
<td>4.2</td>
<td>3.5</td>
<td>0</td>
<td>13</td>
</tr>
</tbody>
</table>

In Table 11.11, it can be observed that the dynamic complex genomes representation and the complex genome representations perform better than the bit genome representation. They are both able to quickly find promising individuals that can be evolved further. Although the dynamic representation is better than the bit representation, it does not appear to be better than the fixed size complex representation. Rather, the static sized complex representation seems to be able to more quickly find promising individuals.

The number of generations required for finding promising individuals is of course dependent on the size of the genomes that are being evolved. In case a too small size is chosen, then the target concept cannot be learned, and in case a too large size is chosen, then the resulting number of generations may increase drastically. The former problem can be considered more severe, and thus, it could be argued that a larger genome size is preferable. This will however likely have an impact on the number of generations required to find promising individuals. In Chapter 6, a second potential benefit of the dynamic complex genome representation was argued to be that it does not require manual specification of size. It should be able to find promising individuals rather quickly even when large initial sizes are specified. In order to investigate this, three additional experiments have been carried out in which the initial size of the two forms of complex genome representation has been varied from the default of 10 transitions and 10 places, to 15 transitions and 15 places, and to 20 transitions and 20 places. In each of the three experiments, 60 evolutionary runs have been carried out, 30 for each of the two complex representations. Again, the complex limit \(n_{\text{cl}}\) has been set to 0.03 seconds, in order to lower the amount of time spent on evolution. In the experiments, the number of generations until finding promising individuals has been measured. The results are presented in Table 11.12.
Table 11.12: The number of generations required until fitness > 0.1 for the complex genome (CG) and the dynamic complex genomes (DCG), when varying the number of places and transitions from 10,10 ($P_{10}, T_{10}$) to 15,15 ($P_{15}, T_{15}$) and to 20,20 ($P_{20}, T_{20}$), respectively. $Min_{95\%}$ and $Max_{95\%}$ represent 95% approximated confidence intervals calculated according to Equation 10.2.

<table>
<thead>
<tr>
<th></th>
<th>$\mu$</th>
<th>$\sigma$</th>
<th>Min</th>
<th>Max</th>
<th>$Min_{95%}$</th>
<th>$Max_{95%}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{10}, T_{10}$</td>
<td>1.467</td>
<td>1.358</td>
<td>0</td>
<td>4</td>
<td>0.98</td>
<td>1.95</td>
</tr>
<tr>
<td>CG</td>
<td>4.700</td>
<td>4.103</td>
<td>0</td>
<td>18</td>
<td>3.23</td>
<td>6.17</td>
</tr>
<tr>
<td>DCG</td>
<td>5.267</td>
<td>6.812</td>
<td>0</td>
<td>33</td>
<td>2.83</td>
<td>7.70</td>
</tr>
<tr>
<td>$P_{15}, T_{15}$</td>
<td>2.667</td>
<td>2.368</td>
<td>0</td>
<td>8</td>
<td>1.82</td>
<td>3.51</td>
</tr>
<tr>
<td>CG</td>
<td>40.10</td>
<td>64.53</td>
<td>0</td>
<td>249</td>
<td>17.0</td>
<td>63.2</td>
</tr>
<tr>
<td>DCG</td>
<td>1.033</td>
<td>1.542</td>
<td>0</td>
<td>5</td>
<td>0.48</td>
<td>1.59</td>
</tr>
</tbody>
</table>

As can be observed in Table 11.12, the complex genome representation finds promising solutions significantly faster than the dynamic complex genome representation in the least complex genome setup. However, the differences are not very large. In setup $P_{15}, T_{15}$, the dynamic complex genome representation however finds promising individuals faster. Not to a significant extent though. In the most complex genome setup, the dynamic complex representation significantly faster finds promising solutions. The dynamic representation actually performs better given larger initial genome sizes. A reason for this may be that a larger initial number of nodes and edges increase the chance of randomly finding promising solutions. The results show that in case of large genome sizes, then there may be some advantage of using a dynamic complex genome representation, since it is able to very quickly find promising individuals. Moreover, and as previously argued, larger genome sizes can be considered preferable since a larger number of concepts can be represented. As the results show, the initial genome size can be kept large for the dynamic complex genome representation, thus removing the issue of determining suitable genome sizes manually.

11.4.3 Conclusions

The dynamic complex genome representation can be used for evolving promising solutions more quickly than the other two forms of representation. Moreover, it does evolve solutions of similar quality, without any observed disadvantages. By using the dynamic complex genome representation, the task of manually specifying genome size can be avoided since the initial size can be set rather large. Moreover, this does not seem to have a negative impact on the number of generations required to find promising individuals, rather the opposite. Hence, it is argued that hypothesis $L_5$ holds: it is beneficial to use a dynamic complex genome representation when evolving Petri nets for situation recognition.
11.5 Bootstrapping

One important aspect of using genetic algorithms when working with situation recognition is the ability of enhancing the performance of manually constructed Petri nets. It can be a tricky task for a human to precisely specify the contents of an interesting situation. It is likely that some important aspect of an interesting situation have been missed during template design. The task of addressing this problem is a task that could require many hours of fine tuning and elaboration. Still, important details may be missed. The machine on the other hand, is able to quickly test multiple solutions in order to find important aspects and in order to fine tune already promising aspects. Still, given the complexity of the problem, it is possible that a genetic algorithm does not have the power to escape the local optimum specified by manually constructed Petri nets. Nevertheless, in case there is room for improvement, then genetic algorithms are usually able to improve at their task, given that they are given enough evolutionary time, and given that bias allows them to do so. The following hypothesis is therefore suggested:

Hypothesis $L_4$. The performance of manually constructed Petri nets can be improved through the use of genetic algorithms.

Already, the complexity of the representation has been discussed. It may be the case that genetic algorithms are not able to discover generally applicable situation templates, due to many hard to find dependences amongst subelements in a situation. The genetic algorithm may fail to see the larger picture due to its lack of domain knowledge. Humans on the other hand, have the capability of defining components of an interesting situation in the abstract. We have the capability of grasping the larger picture, but we may have problems in specifying details. A machine is however good with details but does not have the capability of conceptualising the grander picture. Combining the strengths of both ways of template construction, could thus be argued to be beneficial. Problem specific domain knowledge could be used as guidance for the genetic algorithm. This may result in increased performance. Still, guidance provided by manually constructed Petri nets could restrict the evolutionary search too much, resulting in optimal solutions not being discoverable. It is however believed that the inclusion of guidance is beneficial due to the complexity of the representation. The following hypothesis is therefore suggested:

Hypothesis $L_5$. It is beneficial to bootstrap the initial population with manually constructed individuals, when evolving Petri nets for situation recognition.

11.5.1 Method

In order to investigate hypotheses $L_4$ and $L_5$, it is necessary to compare the performance of Petri nets evolved using bootstrapping, with Petri nets evolved
without bootstrapping and with the manually constructed Petri net. In order for hypothesis $L_4$ to hold, then the performance of Petri nets evolved with bootstrapping must be higher than the performance of the Petri net used for bootstrapping. This would indicate an improvement at the task. In order for hypothesis $L_5$ to hold, then performance of evolved Petri nets needs to be higher when using bootstrapping, compared to not using it.

Similar to before, in Section 11.2 it was established that the weighting of precision and recall does not seem to matter when used in a fitness function. Thus, a fitness function promoting the two measures equally will be used. In Section 11.3, it was established that initial seeding is beneficial. Similarly, it is thus argued that initial seeding should be used in the experiment in order to give the non bootstrapped setting a higher chance of discovery. Section 11.4 concluded that the dynamic complex genome representation is beneficial. Nonetheless, it is argued that all three forms of representation should be investigated since their varying degrees of restriction may affect the results. Similarly, it is argued that the complexity limit $nc_l$ is set to 0.06 in order to allow for more complex solutions to be searched for. The pick pocket Petri net presented in Figure 10.1 has been used in the bootstrapping experiments.

### 11.5.2 Results

In this experiment 90 evolutionary runs have been carried out, 30 for each genome representation. Of the 30 runs for each genome representation, the first 15 runs have been bootstrapped and seeded, whilst the remaining 15 runs have only used seeding. Table 11.13 presents fitness results of the evolutionary runs with and without bootstrapping, for each of the genome types.

<table>
<thead>
<tr>
<th></th>
<th>BG</th>
<th>BG+B</th>
<th>CG</th>
<th>CG+B</th>
<th>DCG</th>
<th>DCG+B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runs</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Max</td>
<td>0.5673</td>
<td>0.6357</td>
<td>0.5673</td>
<td>0.6862</td>
<td>0.5792</td>
<td>0.6333</td>
</tr>
<tr>
<td>Best fitness $\mu$</td>
<td>0.5217</td>
<td>0.5901</td>
<td>0.5593</td>
<td>0.6006</td>
<td>0.5580</td>
<td>0.5936</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>0.1008</td>
<td>0.0197</td>
<td>0.0054</td>
<td>0.0312</td>
<td>0.0105</td>
<td>0.0227</td>
</tr>
</tbody>
</table>

As can be seen in Table 11.13, it is beneficial to bootstrap the evolutionary process. The fitness of the best individual increases for each of the three genome representations. It can also be observed in Table 11.13 that the average fitness over all evolutionary runs increases for each representation when using bootstrapping. It is interesting to see where the increase in fitness stems from. To investigate this, Table 11.14 shows precision, recall and f-value results for the best individual of each setting, on the data that was used for evolution.
According to Table 11.14, the increase in fitness stems from an increase in precision, since recall is roughly 1 for the best individual for each of the evolutionary settings. Bootstrapped solutions are better compared with their non bootstrapped variant. So far, we have however only looked at results with respect to the data that was used for evolution. It is of importance to also validate the quality of the evolved solutions using the evaluation data. Table 11.15 present precision, recall and f-value results for the best evolved individual for each of the evolutionary setups, as well as for the manually constructed Petri net, on the evaluation data.

Table 11.14: Precision, recall and f-value for the best evolved Petri nets for each setting, on the data that were used for evolution.

<table>
<thead>
<tr>
<th></th>
<th>BG</th>
<th>BG+B</th>
<th>CG</th>
<th>CG+B</th>
<th>DCG</th>
<th>DCG+B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>0.0390</td>
<td>0.2471</td>
<td>0.0393</td>
<td>0.3110</td>
<td>0.0659</td>
<td>0.1912</td>
</tr>
<tr>
<td>Recall</td>
<td>1</td>
<td>0.9583</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F-value</td>
<td>0.0750</td>
<td>0.3879</td>
<td>0.0757</td>
<td>0.4744</td>
<td>0.1236</td>
<td>0.3197</td>
</tr>
</tbody>
</table>

Table 11.15: Precision, recall and f-value results on the evaluation data set, for the best evolved individual for each of the three genome representations, with and without bootstrapping, and for the manually constructed Petri net.

<table>
<thead>
<tr>
<th></th>
<th>Precision μ</th>
<th>Precision σ</th>
<th>Recall μ</th>
<th>Recall σ</th>
<th>F-value μ</th>
<th>F-value σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG</td>
<td>0.0317</td>
<td>0.0064</td>
<td>0.9923</td>
<td>0.0243</td>
<td>0.0614</td>
<td>0.0120</td>
</tr>
<tr>
<td>BG+B</td>
<td>0.1928</td>
<td>0.0594</td>
<td>0.6834</td>
<td>0.1325</td>
<td>0.2977</td>
<td>0.0818</td>
</tr>
<tr>
<td>CG</td>
<td>0.0317</td>
<td>0.0064</td>
<td>0.9923</td>
<td>0.0243</td>
<td>0.0614</td>
<td>0.0120</td>
</tr>
<tr>
<td>CG+B</td>
<td>0.1878</td>
<td>0.0391</td>
<td>0.7592</td>
<td>0.0523</td>
<td>0.2998</td>
<td>0.0514</td>
</tr>
<tr>
<td>DCG</td>
<td>0.0580</td>
<td>0.0129</td>
<td>0.9247</td>
<td>0.0737</td>
<td>0.1089</td>
<td>0.0231</td>
</tr>
<tr>
<td>DCG+B</td>
<td>0.1616</td>
<td>0.0368</td>
<td>0.8758</td>
<td>0.0741</td>
<td>0.2720</td>
<td>0.0548</td>
</tr>
<tr>
<td>Manual</td>
<td>0.1168</td>
<td>0.0267</td>
<td>0.8842</td>
<td>0.0735</td>
<td>0.2059</td>
<td>0.0437</td>
</tr>
</tbody>
</table>

As can be observed in Table 11.15, it is better to bootstrap the evolutionary process compared to not bootstrapping it, with respect to precision and f-value. This is true for all three genome representations. Moreover, each of the bootstrapped individuals performs better than the manually constructed Petri net, with respect to f-value and precision. Interesting to observe however, is that the performance (precision, recall and f-value) of Petri nets evolved with bootstrapping decreases on the evaluation data set, in comparison with the results achieved on the data that was used for evolution. This could mean that the data used for evolution does not constitute a representative sample. This suggests that more data should have been used during evolution. At present, only 2 out of 20 data sets are used in each evolutionary run. Hence, to achieve better results it could have been beneficial to carry out the evolutionary process.
using more than 2 files. It is possible that this also could have resulted in larger differences compared with the manually constructed Petri net. Still, it seems to be beneficial to bootstrap the evolutionary process since this results in much better performance with respect to precision and f-value. Recall of individuals evolved with bootstrapping drops in comparison with their non bootstrapped counterparts and in comparison with the manually constructed Petri net. Still, the increase in precision and f-value is argued to outweigh the loss in recall.

As we have seen, it is possible to slightly increase the performance of manually constructed Petri nets through the use of genetic algorithms and bootstrapping. However, due to the small size of the evaluation data set it is not feasible to estimate the extent of the gains in performance. In order to more carefully investigate the differences, 30 new data sets have been simulated using the same scenario setup, i.e. 30 files consisting of 15 minutes of data.

In order to investigate the extent of the performance gains, the best evolved Petri nets for each of the genome representations, when using bootstrapping, have been evaluated on the new data sets in order to estimate their performance. Petri nets evolved without bootstrapping have not been evaluated since their performance is lower than bootstrapped settings. Moreover, the manually constructed Petri net has also been evaluated on the new data sets. Table 11.16 illustrates precision, recall and f-value on the new data sets.

Table 11.16: Precision, recall and f-value results on the new evaluation data set, for the best evolved individual for each of the three genome representations when using bootstrapping, and for the manually constructed Petri net.

<table>
<thead>
<tr>
<th></th>
<th>Precision μ</th>
<th>Recall μ</th>
<th>F-value μ</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG+B</td>
<td>0.1929</td>
<td>0.6994</td>
<td>0.2979</td>
</tr>
<tr>
<td>CG+B</td>
<td>0.1743</td>
<td>0.7358</td>
<td>0.2773</td>
</tr>
<tr>
<td>DCG+B</td>
<td>0.1526</td>
<td>0.8578</td>
<td>0.2563</td>
</tr>
<tr>
<td>Manual</td>
<td>0.1120</td>
<td>0.8716</td>
<td>0.1971</td>
</tr>
</tbody>
</table>

As can be observed, the results presented in Table 11.16 are similar to the results presented in Table 11.15 however, on a larger evaluation set. Precision, recall and f-value drops a little for each of the Petri nets on the larger data sets, however, in a similar fashion. As can be observed, precision and f-value increases in comparison with the manually constructed Petri net. Recall however drops, for each of the three representations. Since the Petri nets now have been evaluated on a larger evaluation set, we can use approximated confidence intervals to determine if the differences in performance are of significant extent. Equation 10.2 has been used to estimate the average precision and f-value with 95% confidence intervals. Figure 11.2 illustrates the precision and recall for the four Petri nets with 95% approximated confidence intervals.
Figure 11.2: Precision and recall on the new evaluation data set, for each of the best evolved Petri nets and for the manually constructed Petri net. The boxes represent 95% approximated confidence intervals and the line in the middle is the mean.

As can be observed in Figure 11.2, precision is significantly higher for each of the evolved Petri nets, compared with the manually constructed Petri net. Recall however drops significantly for two of the evolved Petri nets, whilst it remains similar for the Petri net evolved with a dynamic complex genome. It is thus possible to improve the performance since precision can be significantly increased, whilst recall is not significantly worse. To strengthen this conclusion, Figure 11.3 illustrates f-values for the four Petri nets, also with 95% approximated confidence intervals.

Figure 11.3: F-value on the new evaluation data set, for each of the best evolved Petri nets and for the manually constructed Petri net. The boxes represent 95% approximated confidence intervals.

As can be observed in Figure 11.3, f-values are significantly higher for each of the evolved Petri nets, compared with the manually constructed Petri net. It can thus be concluded that performance can be slightly increased, to a significant extent.
11.5.3 Conclusions

It has been shown that it is better to evolve individuals when using bootstrapping, compared to not using bootstrapping, since the resulting Petri nets have higher precision and f-value. Hence, it is argued that hypothesis $L_5$ holds, i.e. it is beneficial to bootstrap the evolutionary process with manually constructed Petri nets.

It has also been shown that the performance of manually constructed Petri nets can be improved to a significant extent. The gains are however not very large. Nonetheless, it has been shown that precision and f-value can be increased slightly through the use of genetic algorithms, without a loss in recall. Although results showing larger differences would have been preferred, performance can be slightly improved. Hence, it is argued that hypothesis $L_4$ also holds. In other words, the performance of manually constructed Petri nets can be improved through the use of genetic algorithms.

11.6 Chapter summary

In this chapter the use of genetic algorithms for evolving Petri nets for situation recognition has been investigated empirically through the use of a simulated pick pocket scenario. It has been shown that the performance of manually constructed Petri nets can be increased slightly. The main contributions of the chapter consist of a number of presently valid hypotheses with respect to the use of genetic algorithms for learning Petri nets for situation recognition.

- It is slightly beneficial to seed the initial population with valid Petri nets.
- It is beneficial to use a dynamic complex genome representation.
- It is beneficial to bootstrap the evolutionary process with manually constructed Petri nets.

It has been found that the choice of weighting between precision and recall, when used in a fitness function, does not seem to have any impact on the performance of resulting Petri nets. The choice of weighting could however affect the time needed to evolve fit individuals; however, this has not been investigated in this thesis. Thus, a function promoting them equally may as well be used for simplicity. Most importantly however, is the introduction of the dynamic complex genome representation. It has been shown that this representation can be used to very rapidly find promising solutions. It may very well be the case that this representation can be used to also quickly adapt to changing definitions of interesting situations, e.g. the pick pocket and the accomplice change their behaviour and start to meet before every theft. This topic has however not been investigated in this thesis. The potential of the representation is considered promising and the benefits of such a representation need to be investigated further in future work.
Part V

Conclusion
Chapter 12
Conclusions

Situation recognition is an important problem to study since it can play an important role in supporting decision makers with achieving enhanced situation awareness. Situation recognition involves the task of recognising a priori defined situations of interest in a continuous flow of data and information. It can also be seen as the task of tracking changes in states of multiple objects over time, to recognise interesting patterns of events. The situations considered in this thesis reside at a symbolic level and can be of both concurrent and partial temporal definition. This makes the problem rather hard to solve.

12.1 Thesis problem and research objectives

To delimit the problem that has been investigated in this thesis, the focus has been put on investigating the viability of using Petri nets for situation recognition. Consequently a problem statement was formulated as follows.

**Problem statement.** Is the Petri net based approach viable for recognising situations of partial temporal definition?

A viable solution to the problem of recognising situations of partial temporal definition has in the thesis been defined as a solution that:

2. Is efficient with respect to time.
3. Allows for manually constructed situation templates to be adapted.
4. Can be used in real world systems.

While all four properties are important, this thesis has mainly been focused on requirements two and three, namely, to investigate efficiency and adaptability of a Petri net based technique. Two research questions were consequently formulated to investigate the two requirements in focus.
Research question 1. Can Petri nets be used for recognising situations as efficiently as rule based approaches using the Rete algorithm with extensions for explicitly modelling temporal constraints?

Research question 2. Can genetic algorithms be used to successfully learn Petri net based situation templates?

To guide the work required for answering the two research questions, five research objectives were formulated. The research objectives have been considered as important milestones in the process of investigating the problem outlined in the thesis. The work that has been carried out with respect to each of the objectives will now briefly be summarised.

Research objective 1

As presented in Chapter 2, situation assessment is a broad topic that covers many different aspects. The focus in this thesis has been put on situation recognition, which Steinberg (2009) identifies as an important function of situation assessment. In its essence, situation recognition is concerned with recognising the occurrence of situation types in a set of available information, where situation types are defined using any number of symbolic relations. Many interesting situations may however also play out over time. That is, they may be of partial temporal definition. This problem is, however, not well defined and this raises the need for the first research objective.

Research objective 1: Identify a suitable conceptualisation of the situation recognition problem in literature, and if one does not exist, suggest one. Furthermore, formally define the situation recognition problem and suggest a suitable representation of concurrent and partially temporally synchronised situations of interest. This objective includes analysing and synthesising existing theories.

A suitable conceptualisation and definition of situation recognition, which allows for situations of partial temporal definition, has not been found in literature. Hence, existing literature has been analysed to arrive at a conceptualisation that allows for temporal aspects to be represented. This conceptualisation interprets the world in terms of abstract processes (Section 4.2). A conception of situations as processes has previously been suggested by Lambert (2003b). In the view presented in this thesis, there are some processes going on in the world which can be described as being in certain states at different points in time. Processes in the world can be described at many different levels of abstraction. The evolutions of such processes are considered to be situations (Section 4.4). In the conceptualisation, a situation is considered as a time ordered vector consisting of sets of predicates between objects in the domain of interest. In light
of the suggested conceptualisation, a solution to the situation recognition problem was defined in Section 4.5, as a ranked list of situations that to some degree can be considered as instances of a situation template that represents a situation type.

Situation templates were defined in Section 4.5.1 to consist of a set of variables for terms in the domain, a set of constraints that can be put on predicates and a set of temporal constraints that can be put on non-temporal constraints. This effectively allows for interesting situation types to be expressed. An initial complexity analysis was carried out in Section 4.5.3, and it shows that the problem is very complex. To widen the view, the situation recognition was put in context of an information fusion system in Section 4.6. This was followed by a discussion around a number of requirements that can be put on potential solutions to the situation recognition problem in Section 4.6.2:

- **Completeness.** All instantiations of an interesting situation must be found.

- **Recognition performance.** The number of false positives should be kept low, whilst also finding all truly interesting situations.

- **Robustness.** Situations need to be recognisable even when not all necessary information is available.

- **Understandability.** It is human decision making that should be supported. Templates thus need to understandable and definable by humans.

- **Time consumption.** Situation recognition is depicted as an online system. In other words, it needs to be able to process information efficiently.

The work that has been carried out in connection to the first research objective does not directly answer any of the research questions. It is however considered vital to have a proper conceptualisation and definition of the problem, before trying to solve it.

**Research objective 2**

It was argued in Section 5.1 that the situation recognition problem can be reformulated to iteratively recognise situations. In this view, changes to the state space are used as input to a black box, which in turn should output any new instances of recognised situations. Per its definition, this kind of solution is more efficient since it is only the difference between two consecutive changes that needs to be computed. A Petri net represents an instance of such an iterative solution. Besides being able to recognise situation iteratively, it is however of importance that a solution also fulfils the requirements that have been derived. This motivates the need for the second research objective.
Research objective 2: Investigate, develop and suggest extensions to Petri net based recognition, to suit the problem of recognising situations of temporal and concurrent nature.

As established in Section 4.5.3, the situation recognition problem may in a worst case scenario grow exponentially. It is thus necessary to have access to solutions that not only are able to represent interesting situations of partial temporal definition, but which are also able to recognise such situations efficiently. It is however of importance to also take into account robustness, completeness and understandability. It was established in Section 5.2 that the object Petri net approach seems to fit the situation recognition problem. Still, four problems have been identified with the object Petri nets: (1) the complete matching space is not considered, (2) role assignment is not properly managed with respect to situation templates, (3) the matching procedure is not precisely defined and (4) incomplete matching is not allowed. These problems need to be solved in order to fulfil the requirements that can be put on solutions.

Consequently, an extended Petri net based technique was proposed in Section 5.3 and Section 5.4. This technique addresses the identified problems. In the technique, tokens represent partial matches between the flow of information and a situation template modelling some interesting behaviour. More precisely, a token represents a partial unification between the flow of information and the modelled situation type. Similar to Ghanem et al. (2004), two types of transitions are used: regular transitions and conditional transitions. Regular transitions are similar to those of regular Petri nets, and these are activated when a new set of combinable tokens exist in the input places. Conditional transitions have an additional constraint coupled to them, and these are in the suggested approach activated when external events are processed.

At the invocation of a conditional transition, new tokens are created to represent the new information. Subsequently, these new tokens are combined with every valid combination existing on the input places of the transition. The tokens resulting from this are then inserted to the output places of the transition. In contrast to traditional Petri nets, tokens are not consumed by default when they are used. The reason for this is to model the complete space of partial matches. However, this also results in that the number of partial matches grows very rapidly. To avoid this combinatorial explosion, it has been suggested that a maximum time should be assigned to every Petri net, e.g. an interesting situation should occur within a specified time limit. A sliding window approach was suggested for removing partial matches that become too old.

Research objective 3

Although an algorithm and representation may seem good in theory, solutions to computational problems also need to be investigated in light of relevant input data. In terms of situation recognition, this requires that suitable scenarios are
identified. Important in the notion of recognition are however the concepts of precision and recall. The aim is to achieve good results with respect to these two measures. This in turn however requires that there not only exists data, but also that the data is labelled. This is not often the case when it comes to the problem of recognising situations. Lastly, situation recognition is depicted as being part of a larger system residing on top of classical track based processing. These aspects motivated the third research objective.

Research objective 3: Develop a test environment that contains necessary tools for evaluation. This environment could for example include scenarios, simulators and benchmarking capabilities.

A test environment has been constructed which consist of a: development and benchmarking environment, a simulator for constructing data, a fictive pick pocket scenario and a real world maritime scenario.

The first software tool is a development and benchmarking environment for working with situation recognition (Chapter 7). This environment consists of a framework that allows processing from objects to relation extraction and situation recognition. The purpose of the development environment has been to allow for situation recognition algorithms to be compared with respect to performance and efficiency. It is possible that it would have sufficed to only implement the algorithms to compare them. A holistic view has however been advocated, which aims at recognising situations based on object level data. This allows us to not only compare the specific algorithms, but also to implicitly assess the applicability and deployability of situation recognition as a capability in a system. It is possible that the algorithms could have been implemented and compared in some existing framework. As an example, it could have been interesting to use the tools for situation awareness introduced by Kokar et al. (2009). However, this would also have required significant amounts of time to be spent on the development and construction of a suitable ontology, for each investigated case. Although important and interesting, concepts and representations would likely have clouded the algorithmic perspective.

The second tool is a simulator for quickly constructing large amounts of varying data, which contain noise at a relevant level of abstraction (Chapter 8). The simulator has been constructed in a component based fashion, in order to allow for extensions to easily be constructed and integrated. There are many other simulation environments available on the market. Any one of these could potentially have been used, but, the impact on the results in this thesis would not likely have changed very much. During the research work, two commercial simulators have been looked into. The focuses of these are to realistically simulate the real world. Situation recognition however operates at a higher level of abstraction and this also puts other requirements on simulations, for example generate noise at suitable levels of abstraction. A high-level simulator could also have been constructed which completely would have neglected the object
level to instead simulated relations directly. Such a simulator could possibly have allowed for a wider set of interesting patterns to have been explored, resulting in more generally applicable results. However, the distance to real world data would also possibly have been larger. This could in turn have resulted in results that are less directly applicable to real world systems.

The two scenarios were discussed in Chapter 9. The first is a fictive pick pocket scenario that was implemented using the simulator. This scenario consists of a large number of pedestrians that move around in a crowded environment, on top of which pick pocket situations are dynamically instantiated. The scenario allows for the creation of data with varying degrees of complexity. This can be important for properly carrying out comparisons. Lastly, a maritime piloting scenario has also been identified. This scenario can be used for verifying that the Petri net based technique actually can be used for recognising real situations using real data.

**Research objective 4**

Theory is important however, all too often are large amounts of time spent on concepts that are not applicable in practice, or which are not interesting from a real world systems perspective. Empirical investigations are therefore a necessity. Moreover, complexity and performance can often be analysed in theory. In the case of situation recognition however, complexity and performance is largely dependant on the nature of the input data, which often is a priori unknown. This too calls for empirical analysis.

**Research objective 4:** Empirically investigate and compare the efficiency of Petri net based and rule based situation recognition.

Empirical investigations have been carried out using a simulated pick pocket scenario. The Petri net based technique has been compared with a rule based technique using the Rete algorithm with temporal extensions. It has been shown that both of the techniques can be used for situation recognition, and that their recognition performance is similar (Section 10.2). It has however also been observed that their recognition performance decays drastically as the complexity of the input data increases. This means that the patterns that have been used are not restrictive enough. It does, however, not directly indicate that the techniques themselves are degraded. It was shown in Section 10.3 that the Petri net based technique is more efficient than the rule based technique, both in terms of time and memory consumption, as the complexity increases.

Although it has been shown that the Petri net based technique can be used for efficiently recognising situation, it has only been investigated in a limited setting. There can of course exist other cases in which the outcome would have been different. The no free lunch theorem (Wolpert and Macready, 1997) is likely to apply to situation recognition too, as well as to other techniques and problems. Still, the data sets that have been used contain multiple objects and
12.1. Thesis Problem and Research Objectives

Play out in dense and dynamic environments. Moreover, the data sets have been of varying complexity. These aspects thus mean that the technique has been investigated in a quite general but still demanding environment.

An extension for possibly increasing the efficiency of the Petri net based technique with respect to time was suggested in Section 5.6. This extension consists of precomputing valid combinations in the input places of transitions with multiple input places. The extension builds upon the suggestion by Ghanem et al. (2004), to keep a list of activated transitions. This suggestion in turn requires that combinations are precomputed and maintained. It has been shown in this thesis that the explicit use of precombinations does not constitute a robust solution for increasing the efficiency (Section 10.3).

**Research objective 5**

It was observed that the recognition performance of the Petri net based technique, when used with a manually constructed Petri net, decays as the complexity of the input data increase. It has been shown that this is due to the pattern emerging by chance as a result of more data. In turn, this implies that the specific Petri net that has been used is not restrictive enough. This leads to a need of refining the manually constructed Petri net with respect to data. This is also an important aspect in most types of recognition systems, since it can be difficult for human experts to precisely define the contents of interesting patterns. The fifth research objective was therefore stated as follows.

**Research objective 5:** Analyse and develop algorithms and representations for adapting Petri net based definitions of situation types, and empirically investigate if and how the performance on the situation recognition task can be improved, and/or maintained, through the use of genetic algorithms.

The use of genetic algorithms for evolving Petri nets has been investigated. Genetic algorithms should be suitable choice, since one of their strengths is to learn in situations where the objective to optimise can be hard to define, and where there are no precisely defined training examples. It is, however, not a straight forward task to construct a suitable genome representation for the task, since Petri nets for recognition consist of a graph where nodes represent predicates of varying arity. The learning task thus involves both learning the graphical structure of Petri nets, as well as the content of nodes. Due to the complexity of the description, the choice of genetic procedure and representation becomes an important aspect to analyse since different mechanisms can have different effects on the target concept that is learned.

Three genome representations were suggested in Chapter 6: bit genomes, complex genomes and dynamic complex genomes. Each of these representations have different characteristics which can influence for example: (1) the
performance that can be achieved on evolved solutions, (2) how long it takes to evolve fit individuals and (3) how often fit individuals can be evolved.

It has been shown in Chapter 11 that it is preferable to have a dynamic complex genome representation, since it allows for promising solutions to be found very quickly, without a negative impact on the quality of the resulting solutions. Moreover, it has also been shown that it is beneficial to bootstrap the evolutionary process with manually constructed Petri nets and that it is possible to slightly increase the performance of manually constructed Petri nets. Still, the performance of the resulting Petri nets is not satisfying. It is possible that the use of some other learning technique would have resulted in better results.

In the study on the use of GAs, two randomly selected input files have been used in each evolutionary run. This has been repeated for a number of evolutionary runs, for each algorithmic setting that has been investigated. It is possible that it would have been better to run each algorithmic setting on exactly the same input, since it then only would have been one parameter that differed. Given large numbers of evolutionary runs, the results should have been the same; however, due to the extensive amounts of time required for the evolutionary process, a sufficient number of evolutionary runs could not be derived.

12.2 Answers

Research question 1

It has been shown in this thesis that Petri nets can be used as efficiently as a rule based technique based on the Rete algorithm with temporal extensions. This algorithm is a priori known to be efficient and it was therefore argued that in case Petri nets can be used as efficiently, then they are efficient too. The results in the thesis however also identify the Petri net based technique as being more efficient than the rule based technique. The consumption of both time and memory grows faster for the rule based technique compared with the Petri net based technique, when the complexity of the input data increases. It is therefore argued that the answer to the first research questions is yes, and that Petri nets can be used to recognise situations efficiently with respect to time.

Research question 2

It has in the thesis been investigated if genetic algorithms can be used for learning and adapting Petri net situation templates. It is important to highlight the fact that none of the Petri nets that were evolved without manual guidance (bootstrapping), were able to recognise situations as well as manually constructed Petri nets. This may have been due to the space of interesting situations not being separable using the symbolic language that was used, or it may be due to Petri nets not being suitable for use with genetic algorithms. The results in this thesis however also show that the performance of manually constructed
12.3. **DISCUSSION**

Petri nets can be slightly improved through the use of genetic algorithms. The improvements are however small and this needs to be investigated further. A conclusive answer to the second research question cannot be determined since it is possible to slightly increase performance. There is indication of success, but it cannot be termed success since the improvements are very small.

**Problem statement**

Besides focusing on the two research questions, the thesis has also showed that Petri nets can be used for recognising situations with good performance in two scenarios. For the simulated pick pocket scenario, Petri nets were successful at its task when the complexity was not too large. For more complex settings however, the performance started to degrade. This was analysed and the reason was found to be that as the amount of data increases, so does the chance of the interesting pattern emerging by chance. Still, situations were recognised with good performance in some of the settings. Moreover, the use of the Petri net based technique has also been investigated using a maritime piloting scenario. This scenario plays out in the real world and consists of real situations. AIS data was analysed to extract relational information, which was used as input to a Petri net. The recognition performance in this scenario was satisfactory.

The investigations carried out in the two scenarios partially show that Petri nets can be used for recognising situations with good performance, and that they should be applicable in real world settings. Petri nets are thus argued to fulfil the four requirements on a viable solution to the problem. The problem statement is therefore considered answered affirmatively: Petri nets constitute a viable solution to the situation recognition problem.

**12.3 Discussion**

The task of recognising situations of temporal definition is an important, but complex, problem to solve. Recognition can essentially be viewed as either a retrospective analysis or as an online analysis. In the former case, much time can be spent on searching for interesting patterns in historical data. The focus in this thesis has however been on the latter case, i.e. information is analysed when it arrives to, in “real-time”, recognise ongoing situations. A predictive view might however also be interesting in for example early warning type systems. In such cases we would like to recognise the occurrence of a situation before it has actually occurred. In fusion terminology this is more accurately referred to as situation prediction or projection. Nonetheless, it is closely related to recognition. The models exist. Why not use them for prediction too. Interesting along these lines, and closely related to the present work, is recent work by Baumgartner et al. (2010), which successfully demonstrates the use of coloured Petri nets for prediction of critical situations in large control systems, in for example traffic management applications.
Naturally, Petri nets are not the only choice for addressing the situation recognition problem. In this thesis we have also looked briefly at rule based techniques based on the Rete algorithm with temporal extensions. Moreover, it is also possible to use temporal constraint networks. Both rule based techniques and temporal constraint networks allows for precise temporal constraints to be expressed. As an example, a template can be constructed that specifies that event $A$ should occur at least 1 hour before event $B$, but no more than 2 hours before. Such temporal constraints can be specified when working with temporal constraint networks and temporal constraint propagation, as well as when using the Rete algorithm with the temporal extensions suggested by Walzer (2009). If there is a need for expressing precise temporal constraints, then one of the other approaches may be preferable. Still, extending the Petri net based technique to also use such temporal constraints should also be possible.

Petri nets allows for quite complex scenarios to be described in a compact fashion since they allow for concurrency to be represented. This however also makes the problem much more complex to solve. The much simpler paradigm of FSA is much less complex and could possibly be used to achieve better performance. However, to represent the same types of patterns, then this would require that all possible state paths are described. In fact, this turns out to equal a serialisation of the Petri net. The result being as many needed comparisons for recognition. Also interesting however, is the use of tree automata for plan recognition as discussed in Högberg and Kaati (2010).

A problem that largely has been ignored in this thesis is the uncertainties often involved in sensing the real world. Information may be uncertain, it may be missing and it may be counterfactual. The Petri net based technique suggested in this thesis represents a quite rigid solution to the problem as it does not allow for uncertainty to be expressed. Bayesian networks, temporal Bayesian networks and hidden Markov models allows for recognition to be carried out probabilistically. Techniques such as these might therefore be considered more suitable in domains where there are large amounts of uncertain and missing information. However, if there is a need to be able to express concurrency, then it is the Petri nets that have the advantage since this actually is one of their main advantages. Moreover, Petri nets have in the past been used both stochastically and probabilistically, and extending the suggested technique to also handle uncertainties should thus be possible.

The use of precombinations did not seem to give any great boost in efficiency. Another approach would be to structure the content of places to allow for selective retrieval, i.e. instead of iterating over all partial matches in an input place, conditions for retrieval can be used to only yield interesting tokens. This can for example be done by ordering partial matches according to variable bindings, and then by retrieving all partial matches that matches bindings in the new token constructed from the information in the processed event. At first glance this may seem as a trivial problem to solve, since we simply could use some form of hashing function that depends on variable bindings. How-
ever, in a specific transition it is not known which variables that have been bound in preceding places. This could be reasoned around analytically, but in the end, all keys would need to be inspected for possible subunifications since missing information is allowed. It is, however, possible to use some form of tree structure to more quickly find matches. In the worst case, this would still result in all tokens being inspected. However, in many cases the set of interesting tokens would likely be restricted. A tree structure does not give any boost in cases where only single variables have been bound, without having nodes indexed according to variable bindings. A potential solution would be to use a tree structure over unifiable variables and to also index the content of each tree node using a hashing function over variable bindings at the next consecutive level in the tree. The gain that can be achieved with this kind of solution is also highly dependent on the input data and on the ratio between partial matches and events. Furthermore, it could be the case that too much memory is required for maintaining such a tree structure. Still, it is of interest to investigate if such a solution can be used to lower the computational demands.

Although an improvement of the performance of manually constructed Petri nets is achievable through the use of genetic algorithms, the precision of evolved Petri nets remains too low. This indicates that the target concept cannot be properly learned. There may be different reasons for this, but in terms of machine learning, the problem is related to restrictive bias, inductive bias and the data used for learning\(^1\).

In case the problem depends on the data used for learning, then there are two possible issues. The data does perhaps not represent one definition of a specific type of situation, but two or more. This might very well be the case although simulated data has been used, since the implemented situations are dynamically instantiated and depend on the “normal” model. Instantiations are thus not constructed statically. This could in turn lead to different implementations of the interesting situation. Perhaps it is necessary to learn more than one definition of an interesting concept. Still, the representations that are used, especially the dynamic complex genome representation, should be able to learn multiple definitions of an interesting behaviour if this is what the data represents. It may also be the case that the data used for evolution does not constitute a representative sample of the distribution being learned. Indications of this have been observed (see Section 11.5). However, even on the evolutionary data, the performance of the learned Petri nets is not satisfying. This indicates that this is not the main source of the problem of low performance.

In case the problem depends on inductive bias, then there may also be two main underlying reasons. The fitness function does perhaps not promote learning of the target concept. Intuitively this should not be the case since it is high precision and recall that is the aim. The deployed genetic algorithm and the genome representations does perhaps not allow for local optima to be escaped.

\(^1\)For details about these concepts, see for instance Mitchell (1997).
Perhaps other kinds of genetic approaches need to be investigated, such as the evolutionary programming approach suggested by Moore and Hahn (2004) or through the use of some other approach based on genetic programming. Still, during the many evolutionary runs that have been carried out, not a single one has been able to discover Petri nets with higher performance.

In case the problem depends on restrictive bias, i.e. the target concept cannot be represented, then there are again two possible problems. The Petri net representation might not be able to represent an interesting situation. This is however not likely since the Petri net representation is quite general. The only restriction is that events should arrive in a time ordered sequence. This is indeed the case for the data that has been used for evolution, and hence, this is not likely to cause of the problem. The second source of the problem could be that the underlying symbolic language is not expressive enough for capturing the situations of interest. Perhaps a symbolic language consisting of the predicates Close, Approach, Intercept, SlowingDown and SpeedingUp is not enough for separating intentionally instantiated situations from the normal flow of events. The reason could also depend on the actual definitions of the relations that are used. In case an IsPickPocket relation could be extracted, then quite naturally, the situations could be discovered using the Petri net based technique. Such a relation is however not very realistic and it thus becomes uninteresting to investigate if such a relation could be used to find the “perfect” definition of a situation type. Besides these reasons it could actually be the case that the behaviour is not at all separable from other activities.

It has been discovered during experimentation that it takes very long time to use genetic algorithms for evolving Petri nets for situation recognition. This is also a known problem with genetic algorithms. Several strategies for lowering the required time have been investigated. These have however only focused on lowering the number of generations required to find promising individuals. Very much time is however also spent at evolving these promising individuals to fit individuals. Some experiments have taken several weeks to complete, and this raises questions regarding the suitability of genetic algorithms for the purpose of learning Petri nets. It is the author’s belief that the use of other learning mechanisms, mechanisms that are more suited for learning concepts that involve processing large numbers of events, should be investigated.

12.4 Future work

It has been shown in this thesis that Petri nets can be used for efficiently recognising situations of partial temporal definition. This conclusion is however only based on investigations in two specific cases. Moreover, investigations have only been carried out when trying to recognise a single situation. In a real system, there may be many different situations of interest. Thus, it is important to investigate how the Petri net based technique behaves when put in this setting.
Moreover, it is also of importance to demonstrate the use of the Petri net based
technique in more kinds of scenarios.

An assumption in this thesis is that Petri nets are suitable for the task of
recognising situations since they can be easily understood and manipulated by
human users. This assumption has however not been investigated at all. During
experimentation it has been found to be rather easy to investigate variations of
different patterns by simply moving an edge in a graphical user interface. It
is thus believed that Petri nets do provide some benefits compared with for
instance rules since they allow for interaction in a more intuitive way. This
however needs to be verified using for example human user studies.

The efficiency of Petri nets have been demonstrated however, only when
using rather simple situation templates that at the most carry out binary com-
binations of paths. As the number of paths to combine increase, so does the
time consumption of the technique, exponentially. It is therefore of importance
to carry out two types of investigations with respect to this issue: (1) investigate
how the technique behaves when using more complex templates and (2) inves-
tigate potential extensions that allows for the possible increase in consumption
to be lowered, i.e. look into techniques for possibly increasing the efficiency. An
approach based on trees with indexed nodes based on unified variables was dis-
cussed in the previous section. It would be interesting to actually investigate if
this extension is suitable. Also promising could be the use of an activation net-
work (similar construct to the Rete algorithm) which only activates transitions
that fulfil basic requirements.

As discussed, observations of the real world are uncertain and the Petri net
based technique does not allow for uncertainties to be modelled. It is thus in
future work also interesting to investigate how the technique could be extended
to model uncertainty.

Lastly, the performance of the specific Petri nets that have been used for
recognition is not always satisfactory. This too calls for future research. Al-
though it is believed that the problem lies in the target concept not being sepa-
rable using the underlying symbolic language, it could also be the case that the
genetic algorithms that have been used are not able to escape local optima. It is
therefore interesting to investigate other types of learning mechanisms such as
for example genetic programming or inductive logic programming. These are
techniques that already are designed for working with higher level concepts.
References


REFERENCES


Publications in the series
Örebro Studies in Technology


