Methods for Engineering Symbolic Human Behaviour Models for Activity Recognition

Dissertation
to obtain the academic degree of
Doktor-Ingenieur (Dr.-Ing.)
of the Faculty of Computer Science and Electrical Engineering
at the University of Rostock

submitted by
Kristina Yordanova Yordanova, M. Sc.
born on 31.01.1985 in Pernik, Bulgaria
from Rostock

Rostock, 2014

urn:nbn:de:gbv:28-diss2014-0133-5

First Reviewer: Prof. Dr.-Ing. Thomas Kirste, *Mobile Multimedia Information Systems Group*, Institute of Computer Science, University of Rostock

Second Reviewer: Prof. Dr. rer. nat. habil. Adelinde Uhrmacher, *Modelling and Simulation Group*, Institute of Computer Science, University of Rostock

External Reviewer: Prof. Dr. rer. nat. Maritta Heisel, *Software Engineering Group*, Department of Computational and Cognitive Science, University of Duisburg-Essen

Date of dissertation submission: 02 April 2014
Date of dissertation defence: 30 June 2014

In memory of
Snezhana Stoyanova Spasova

Abstract

Context-aware systems are becoming an important part of our everyday life and their ability to accurately recognise the user needs plays a crucial role in their performance. Assistive software would be greatly impaired, were it unable to recognise the current user state, as it would result in inability to correctly assist her. A typical approach in such situations is the employment of probabilistic models that describe the possible states and the probabilities for going from one state to another. Usually these models are handcrafted by the system engineer and the transition probabilities are learned to fit the specific problem. However, in order to build and learn the model, a training dataset has to be collected and annotated which in itself implies finding subjects to conduct an experiment, spending time for repeatedly conducting the experiment, and even more time for annotating it. This makes the building of such models not only expensive but also leads to generalisation problems, as the model is not guided by a domain structure but rather by the underlying sensor readings, which could cause suboptimal solutions.

A different approach is to generate the probabilistic model from prior knowledge instead of learning it. One approach to generating probabilistic models could be the usage of human behaviour models that are later mapped onto a probabilistic model and an inference engine is used for estimating the user state. It exploits the additional advantage that the natural way of human thinking is based on causes and effects instead of probabilities. There are corresponding theories that it would be much easier for a system engineer to build a non-probabilistic model.

Based on the above assumption, this work investigates the ability of symbolic models to encode context information that is later used for generating probabilistic models. It also analyses the problems arising from such approach and the need of a structured development process for model based activity recognition. As a consequence, the contributions of the work are as follows: (1) it shows that it is possible to successfully use symbolic models for activity recognition in the field of activities of daily living; (2) it provides a modelling toolkit that contains patterns for reducing the model complexity; (3) it proposes a structured development process for building and evaluating computational causal behaviour models. In general, the thesis provides a practical guide to implementing and using symbolic models for activity recognition and proposes a structured process for doing it – something that is often overlooked in the field of activity recognition.

Keywords: human behaviour models, activity recognition, symbolic models, daily activities, probabilistic models.

Zusammenfassung

Kontext-sensitive Systeme werden ein immer wichtigerer Bestandteil unseres täglichen Lebens. Ihre Fähigkeit die aktuellen Bedürfnisse des Benutzers zu erkennen spielt eine entscheidende Rolle ihrer Leistungsfähigkeit. Ein Assistenzsystem würde stark beeinträchtigt werden, wäre es nicht in der Lage den aktuellen Zustand des Benutzers zu erkennen und ihn folglich nicht korrekt zu unterstützen. Eine typische Vorgehensweise in solchen Situationen ist die Verwendung von probabilistischen Modellen, die die möglichen Zustände und die Wahrscheinlichkeiten für den Übergang von einem Zustand zum anderen beschreiben. Normalerweise werden diese Modelle und Übergangswahrscheinlichkeiten gelernt und vom System-Ingenieur per Hand abgestimmt, um bestmögliche Ergebnisse zu erzielen. Um das Modell zu bauen und zu lernen muss man jedoch ein Experiment durchführen, Trainingsdaten sammeln und die Daten müssen annotiert werden. Dies macht den Bau solcher Modelle nicht nur teuer, sondern erschwert auch eine universelle Einsetzbarkeit, da das Modell nicht auf Domänenwissen basiert, sondern stark von den Sensoren abhängt und folglich zu suboptimalen Ergebnissen führt.

Eine Alternative ist, das probabilistische Modell auf Basis von vorhandenem Wissen zu generieren. Ein Ansatz zum Erzeugen von probabilistischen Modellen kann die Verwendung von menschlichen Verhaltensmodellen sein, die auf ein probabilistisches Modell abgebildet und mit Hilfe von Inferenztechniken zum Schätzen des Benutzerzustands verwendet werden können. Das reflektiert auch die Tatsache, dass menschliches Denken eher auf Ursachen und Wirkungen basiert, statt auf Wahrscheinlichkeiten. Es gibt Theorien, dass es für einen System-Ingenieur viel einfacher ist ein nicht-probabilistisches Modell zu bauen.

Basierend auf der obigen Annahme untersucht diese Arbeit die Fähigkeit der symbolischen Modelle Kontextinformationen, die später zum Erzeugen von probabilistischen Modellen verwendet werden, zu beschreiben. Zusätzlich werden die Probleme, die mit solche Ansätzen verbunden sind, analysiert und die Notwendigkeit eines strukturierten Entwicklungsprozess für die modellbasierte Aktivitätserkennung diskutiert. Als Folge davon sind die Beiträge der Arbeit wie folgt: (1) es wird gezeigt, dass es möglich ist symbolische Modelle für Aktivitätserkennung im Bereich der Aktivitäten des täglichen Lebens zu benutzen, (2) es wird ein Modellierungstoolkit vorgeschlagen, dass verschiedene Entwicklungsmuster für die Reduzierung der Modellkomplexität unterstützt, (3) es wird ein strukturierter Entwicklungsprozess für die Erstellung und Bewertung von Computational Causal Behaviour Models vorgestellt. Allgemein liefert die Arbeit einen praktischen Leitfaden und strukturierten Prozess für die Implementierung und Verwendung symbolischer Modelle für Aktivitätserkennung - etwas, das im Bereich der Aktivitätserkennung oft übersehen wird.

Keywords: Modelle für menschliches Verhalten, Aktivitätserkennung, symbolische Modelle, probabilistische Modelle.

Acknowledgements

Four years ago when I first arrived in Rostock to begin with my PhD, I did not know anyone in the city. Still from the same beginning there were people who made my work and life here a wonderful experience. Now I would like to properly thank all of them.

First and foremost, I would like to thank my supervisor Professor Thomas Kirste without whom this thesis would not have been possible. The creative and dynamic atmosphere he initiates within the Chair of Mobile Multimedia Information Systems has been the basis of interesting research, numerous discussions, and successful cooperations. I could not imagine any better place to do my research.

I would also like to thank Professor Lin Uhrmacher for her support as a second supervisor of my thesis, and Professor Maritta Heisel for the invaluable feedback she gave me. Thank you very much for helping me shape the final version of this work!

Special thanks go to Frank Krüger for always being there when I was in need of assistance or discussion, and for being such a great research partner. I would also like to thank Martin Nyolt for his invaluable feedback and help during the last one year. I am further grateful to Albert Hein for the successful cooperation we had, and to Sebastian Bader for always being supportive and having a good advice. I am also very grateful to all my former and present MMIS colleagues for making it a pleasure to work alongside them.

Furthermore, I would like to thank my former MuSAMA colleagues for the incredible time I had in the graduate college. I am especially thankful to Alexander Steiniger, Till Wollenberg, Michael Zaki, Rene Leistikow, Alexander Gladisch, Anke Lehman, and Redwan Mohammed for being not only colleagues but also such wonderful friends. Thanks also go to my office mate Robin Nicolay for always managing to create amusing work atmosphere. I would also like to thank Dortje Löper and Lars Schreiber for making me belong here in Rostock.

Apart from that I would like to thank my family in Bulgaria and Spain for always being supportive and ready to help me escape from the everyday problems. I am especially grateful to my incredible grandmother who did not manage to see this work completed, but who never stopped believing in me.

Special thanks go to Vladislav Stoyanov for being part of my life. I thank him for all the occasions in which he has been the sparkle to bring me forward, for all the patience he has shown, and for being there always when I needed him.

Finally, I would like to acknowledge that this work has been funded by the German Research Foundation.

Contents

List of Figures

List of Tables

Glossary

ACT-R **A**daptive **C**ontrol of **T**hought – **R**ational. xviii, 10, 11, 37, 80

ADL **A**ctivities of **D**aily **L**iving. xviii, 23

AR **A**ctivity **R**ecognition. xviii, 5

BDI **B**elief **D**esire **I**ntention. xviii, 6

CCBM **C**omputational **C**ausal **B**ehaviour **M**odels. xviii, 10, 11, 48, 54, 125, 133, 135

CRISP-DM **CR**oss-**I**ndustry **S**tandard **P**rocess for **D**ata **M**ining. xviii, 152

CSP **C**ommunicating **S**equential **P**rocesses. xviii, 28, 201

CTML **C**ollaborative **T**ask **M**odeling **L**anguage. xviii, 10, 42

CTT **C**oncurrent **T**ask **T**rees. xviii, 11, 40, 79

DBN **D**ynamic **B**ayesian **N**etwork. xviii, 6, 79

DEVS **D**iscrete **EV**ent **S**ystem Specification. xviii, 12

GOMS **G**oals, **O**perators, **M**ethods, and **S**election rules. xviii, 11

HBM **H**uman **B**ehaviour **M**odels. xviii, 2, 11

HMM **H**idden **M**arkov **M**odel. xviii, 56

IR **I**ntention **R**ecognition. xviii, 6

KTA **K**itchen **T**ask **A**ssessment. xviii, 23

L **L**ocations. xviii, 68

LISP **LIS**t **P**rocessing Language. xviii, 133

O **O**bjects. xviii, 67

OL **O**bjects **P**laces. xviii, 67

OPL **O**bjects **P**laces **L**ocations. xviii, 67

PDDL **P**lanning **D**omain **D**efinition **L**anguage. xviii, 10, 11, 43, 80, 133

PF **P**article **F**ilter. xviii, 51, 56

PL **P**laces **L**ocations. xviii, 67

POMDP **P**artially **O**bservable **M**arkov **D**ecision **P**rocess. xviii, 80

STRIPS **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver. xviii, 43, 133

Chapter 1

Introduction and Motivation

"Suit the action to the world, the world to the action; with this special observance, that you o'erstep not the modesty of nature."
William Shakespeare

Chapter Summary: *This chapter introduces the concept of activity recognition and human behaviour models and explains the motivation behind using human behaviour models for activity recognition. Additionally, it describes the challenges in using such approaches, gives the goal of the thesis and the problems to be discussed throughout the thesis.*

Chapter Sources: *This chapter is partly based on the technical report "Toward a Unified Human Behaviour Modelling Approach" [160] and the journal paper "Towards Creating Assistive Software by Employing Human Behavior Models" [79].*

Questions to be answered in the chapter:

What are assistive systems? (In Section 1.2)

What is context awareness? (In Section 1.2)

What are activity and intention recognition? (In Section 1.3)

What is human behaviour modelling? (In Section 1.4)

What is prior knowledge? (In Section 1.5)

What types of human behaviour models do exist? (In Section 1.6)

What challenges are there in the field of human behaviour modelling for activity recognition? (In Section 1.7)

What is the goal of the thesis? (In Section 1.8)

1.1 Introduction

With the development of new context-aware technologies and applications, activity and context recognition is a process we might not be aware of, but one we heavily depend on every time we use some application, the aim of which is to provide us with information based on our current location, or surroundings, or activities [69, 46, 27]. And, of course, one would expect such application to perform accurately and to be able to provide the ”right” information[1]. To be able to do that, beneath the ”shiny surface” we usually see, the application should possess an activity recognition component that can correctly recognise the current user actions, and even more – to be able to reason about the user situation [79].

Building such system could be a challenging task as it should be able to reason about the user's whereabouts and intentions based on imperfect user and environment observations and / or the available context knowledge associated with the given problem [80, 112, 92, 162]. Even more, gathering and including the context information into a successful activity recognition system is a challenge in itself [82]. This work deals exactly with the problem of incorporating prior knowledge in the form of symbolic **H**uman **B**ehaviour **M**odels (HBM) for activity recognition. It answers the question of how to build successful HBM for activity recognition, and discusses the problems associated with developing such models. Even more, it discusses the need of a structured development process that could improve the models, automate the model implementation, and solve different problems emerging during the intuitive model development.

This chapter presents the basic concepts associated with context-aware activity recognition. It discusses the challenges related to developing activity recognition systems able to infer the user actions and to provide additional information about the context of those activities. For that reason, we[2] look at context awareness from the viewpoint of assistive systems and its relation to activity recognition. Later we discuss how context information can be incorporated into activity recognition models, what challenges are there and which contributions this work brings.

1.2 Assistive systems and context awareness

In a world where mobile devices are everywhere around us, where new technologies and applications constantly emerge, develop and evolve, context awareness plays a central role in every system that strives to assist the human being in a way. Such systems could be a public transportation app that shows the nearby bus stops [109], or such that strives at improving the routes one takes while driving [83]. It could also be a system that monitors people with cognitive or physical restrictions and strives to provide them with appropriate assistance that would make their lives more independent of caretakers [109]. Another variation of such system would be the different kinds of smart environments – smart meeting rooms, lecture halls and classrooms [46, 47], homes [27, 30], learning environments [69] – that aim at providing the appropriate help for the users within the environment.

In general, an assistive system is any system that provides some sort of help for the user interacting with it. The assistance can be offered in different forms and degree of interaction and automation. In fact, there are different taxonomies proposed for classifying assistive systems. For example Sheridan proposes an eight level scale of automation which starts from systems

[1]Here by ”right” information, the information satisfying the user needs is meant. Such information should be accurate and clear.

[2]Throughout the thesis, the personal pronoun ”we” is used for simplicity in the sentence structure as opposed to the passive structure, and not as an indication that the work was completed by multiple persons.

that provide no assistance at all, to systems that provide full automation [132]. Wandke proposes a more complex classification system as he argues that Sheridan's taxonomy is incomplete and does not refer to all action stages or types of assistive systems. Thus he classifies assistance according to three dimensions: the stages of human-machine interaction that could be achieved, the adjustment (or how the assistive system adjusts to the user needs), and initiative (or who has the initiative for assistance – the user or the system) [124].

Regardless of the kind of assistive system, to be able to assist the user, it needs to possess context awareness, as it is the component allowing the system to help its users in a meaningful and adequate way [70]. Without information about the context of the user actions, it would be impossible for the system to discover the correct kind of solutions that could further the user actions toward achieving her[3] goals.

Dey [33] defines the notion of context in the following way.

Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

On the other hand, Indulska et al. [70] try to define *context* without the meaning of "situation of an entity" and express it in a more concise form.

The context of a computing application is a set of circumstances surrounding it that are potentially relevant to its execution.

Additionally, Dey [33] defines the concept of a system being context-aware as

A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

It is obvious that context plays an essential role in a system that provides assistive services to a user, hence the appropriate incorporation of the relevant context information will be equally important for the system's performance.

Indulska et al. [70] discuss the importance of context awareness in assistive systems and introduce several requirements for context modelling in context aware applications. Defining such requirements is of importance for any assistive system that strives to achieve maximum performance and user satisfiability. Krüger et al. [79] summarise these requirements and extend them on their own with some additional requirements a successful assistive system should possess. Below these requirements are presented.

Imperfect context information: Context-aware applications have the common problem of imperfect context information that could be due to noise in the sensor data, or sensor malfunction, or even inaccurate algorithms for extracting context information from the sensors. It could also be caused by incorrect information provided by the users, such as incomplete or wrong agenda. Thus, when modelling context, it should be able to represent information that is incomplete, imprecise or ambiguous. Additionally, it should have some sort of quality indicators, so that when a sensor is malfunctioning, it can be traced back and repaired.

[3]Throughout the thesis the personal pronoun "she" is used as a substitution of "she / he", and not as an indication that the person in question was a female.

Context histories: Often the information about the current state is not enough for the proper functioning of assistive software. It could also require information about past and future contexts. Therefore, a context modelling approach should be able not only to represent histories but also to be able to reason about them. This information is essential in assistive applications where behaviour patterns are to be detected or where the intention of a user is to be inferred.

Software engineering: A context model benefits the software development when it is introduced in the early stages of the software engineering lifecycle. Then it can be refined incrementally, thus introducing the types of context information required by the application and the data constraints. Additionally, it can be used to evaluate the suitability of the context sensing infrastructure that is already developed, and to present more software or hardware requirements. Furthermore, the context model can be used for producing different use cases for software testing of the context-aware functionality.

Runtime querying and reasoning: One of the context models forms is the runtime model, that is queried by the context-aware applications. The runtime model deals with problems such as how to represent the information at runtime so that it can be reasoned upon in order the system to provide decision making. The model should contain information about the existing context types and their characteristics, as well as concrete context information. It should also be easily extendable so that it can cope with the reasoning in evolving environment.

Interoperability: One of the characteristics of smart environments is that the context-aware applications could be faced with the problem of communicating with components that were unknown to the software designer. Such components could be new applications, or a new device, or new sensing hardware. Thus the context-aware applications should be able to exchange information with them even when the component was previously unknown. This requires either transforming the information in different representations, using a shared context modelling approach, or supporting transformation between different modelling approaches.

Recognition of semantic goals: A common practice in activity recognition is the detection of labels, i.e. a name that is associated with specific data pattern without any further meaning. However, in order a system to be able to perform strategy synthesis for assisting the user, semantic goals should be recognised. Namely, not only an activity, but also the plan (or path) that leads to achieving the goal. That way the system can generate a plan based on the user's semantic goal and assist her while achieving this goal.

It can be seen that a good context-aware assistive system should be able to cope with imperfect context information, thus it should be able to reason about imperfect knowledge about the world and the problem. It should also be able to make track not only of the current state but also of already visited states, so that it can avoid situations that are impossible based on the context histories. Furthermore it should be able to reason about the user state and actions during the model runtime and to be able to recognise not only an action label, but also semantic goals that describe the path taken from the initial to the goal state. These requirements imply that the given assistive system needs a component that is able to make use of the context information and the observations coming from the environment in order to be able to reason about the user actions, their causes and the goals she is following. This should be done in a way that is flexible enough to cope with imperfect knowledge and observations. This component is the activity recognition component that aims at satisfying exactly these requirements.

1.3 The concepts of activity and intention recognition

The previous section already introduced the general idea behind **A**ctivity **R**ecognition (AR) – to accurately recognise the user activities so that the assistive system is able to provide adequate help for the user. However, one more general question is *What is actually an activity?* Is it the movement of the body? Or is it an intentional act or manipulation aiming at achieving some goal? Is it the user behaviour or just a small part of that behaviour? Another question resulting from that is *Are the concepts "action", "activity", "task", and "behaviour" interchangeable?*

The fact is that in the field of activity recognition the words *action*, *activity*, *task*, and *behaviour* are rather loosely used and depending on the community, they could mean the same or different things[4]. To avoid ambiguity, here we attempt to define these concepts before proceeding to the concept of activity recognition.

According to the Oxford Dictionary of English, the word *action* has the following meanings [105]:

(1) the fact or process of doing something, typically to achieve an aim

(2) a gesture or movement

The second definition suggests unconscious or unintentional execution of a low-level movement like suggested by Sukthankar [142]. On the other hand, the first definition implies that the executed process is intentional and executed with a certain purpose. The latter complies with action theory that describes actions as processes causing intentional body movement that are based on certain beliefs and desires [31]. In the course of this work when using the term *action*, we refer to definition (1) and consider any lower body movement that does not have an explicit intentional cause, to be just that – a body movement and not an action.

Definition 1. ***(Action)*** *Action is an intentional process executed by the user that has a certain cause that triggers it, and that aims at achieving a certain goal.*

Furthermore, actions can be divided into two groups – atomic or composite actions.

Definition 2. ***(Atomic action)*** *Atomic action is an action that cannot be divided into more fine-grained intentional actions.*

Notice that here we do not consider as intentional actions body movements that build up a certain action, but that separately do not lead to a goal. Rather, atomic action is the smallest intentional building block of a user behaviour that cannot be decomposed into any more finer actions.

On the other hand, composite actions are built up out of atomic actions or out of other more fine-grained composite actions.

Definition 3. ***(Composite action)*** *Composite action is an action that consists of at least two actions that are either finer-grained composite actions, or atomic. These actions are causally or temporally related with each other in order to build a more complex structure. Here composite action is used as equivalent to the terms activity and task.*

[4]For example, van Kasteren distinguishes between action primitives and activities that are built of action primitives [151]. On the other hand Subramanya et al. [141] do not make use of the term *action* and instead refer to the user actions as *activities*. Alternatively, Sukthankar defines activities as higher level-descriptions composed of low-level movement data [142]. On the other hand, Trafton et al. [146] do not speak of activities but rather refer to *tasks* that build up user *behaviour*.

Based on that definition, one can then define behaviour as:

Definition 4. ***(Behaviour)*** *Behaviour is a set of atomic actions and activities that through their execution lead from the initial state of the world to the goal that the user is pursuing.*

Here, what is meant by *initial state of the world* is the state in which the environment and the user were when the latter first started pursuing her goal.

Finally, one additional concept that has to be defined is that of *intention*. According to the Oxford Dictionary of English [106], intention is explained as:

(1) a thing intended; an aim or plan.

Or in the context of multi-agent systems and the **B**elief **D**esire **I**ntention (BDI) architecture, intentions are commitments to the desires the agent has and to the plans that will achieve them [54]. Based on the above, in this work we refer to intention as:

Definition 5. ***(Intention)*** *Intention is a commitment to achieving a certain goal by executing a set of causally related actions that lead to that goal.*

Now we are ready to finally explore what exactly the field of activity recognition does. According to Sukthankar the *term activity recognition is used to describe the problem of segmenting and classifying low-level movement data into a higher-level description of the activity performed* [142]. He argues that in difference with plan recognition where the algorithms deal with symbolic data and atomic actions, activity recognition algorithms trace the human position over time in order to recognise the activity being performed. This definition however contradicts with other approaches that use symbolic action representation together with sensor data as observations for reasoning about the user activities [66, 80]. Furthermore, probabilistic approaches like some **D**ynamic **B**ayesian **N**etwork (DBN) do not make use of the time relation between the observations, but rather take each observation as independent of the previous and related only to the hidden model state [153, 133, 99]. For that reason here we give a more general definition of the term *activity recognition*.

Definition 6. ***(Activity recognition)*** *Activity recognition is the task of recognising user's atomic actions and activities based on a set of observations.*

Slightly different is the concept of **I**ntention **R**ecognition (IR). Sadri [125] defines intention recognition in the following way.

Intention recognition (...) is the task of recognizing the intentions of an agent by analyzing some or all of their actions and/or analyzing the changes in the state (environment) resulting from their actions.

As in the context of the thesis, the actions have to first be recognised, in order to analyse them, we refer to intention recognition as:

Definition 7. ***(Intention recognition)*** *Intention recognition is the task of reasoning about the goals of a user based on the actions she is executing.*

One could then ask the reasonable question *How can one reason about the user intentions based only on recognised activities? Where does the knowledge about the meaning of these activities come from?* Exactly here comes the role of models describing user behaviour. Such models make use of the available context information and incorporate this prior knowledge in the form of rules so that the system can later reason about the nature of the observed activities. Below the concept of human behaviour models is discussed in detail.

1.4 The concept of human behaviour modelling

Before discussing the different ways of modelling human behaviour, we have to answer one important question: *Why do we need to describe human behaviour at all?*

Objectively speaking, it is possible to recognise user activities without using human behaviour models. There is an increasing number of works in the field of activity recognition that are based only on the observed sensor data, and which try to recognise human activities with the help of different statistical methods [5, 1, 117]. Many of these works give promising results that encourage the development in this field of research.

However, there are several reasons for describing human behaviour, that make developing behaviour models important. From psychological point of view, human behaviour modelling is essential for the better understanding of human actions. Questions such as: *What does an action consist of?*; *What does this action imply?*; *What are the reasons and consequences of an action?* arise. Their answers could be found exactly in human behaviour models that describe not only the actions, but also whether they are composed of more fine-grained actions, what are the relations between the different actions in the context of the composed activities and the user behaviour, how these relations influence the user(s) and the environment [4, 75, 29, 144].

Another reason for modelling human behaviour is to detect the user behaviour based on a given activity. This aspect of HBM is important in systems that deal with human monitoring and rise questions such as: *If an activity is recognised, what kind of behaviour does it imply?* [114]; *Is the behaviour normal or abnormal?*; *Should the activity be reported as deviation from the expected?* [128, 146]. These systems could be in the sphere of health care where the condition of patients is monitored, or in the sphere of security where abnormal behaviour could imply intrusion.

Yet another reason for describing human behaviour is to provide assistance. In this case HBM is essential for systems that try to assist their users in accomplishing a goal. Here when an action is detected and recognised, it is important to discover not only what the action implies, but also why it is executed, what is the final goal of the user. In that way the system will be able to assist the user in reaching her goal [146, 79, 119].

The questions above imply that human behaviour models could provide invaluable information for the needs of users from different backgrounds – from helping medical doctors understand patients with cognitive restrictions, through monitoring and assisting elderly people, to providing proactive assistance in smart environments.

1.4.1 Purpose of human behaviour models

It is apparent that although all of them explain the user behaviour, human behaviour models can serve different purposes. Many of them are used just for simulation of human behaviour where a particular behaviour path is constructed independently of its occurrence probability. With the rapid development of more and more realistic human-centred games, simulation of human behaviour is thoroughly investigated and different models striving to improve the simulation realism and to better explain the agents actions are developed [44, 156].

If we go even further beyond simulation, models are used for inference. If the purpose is inference, the model not only tries to predict the most probable human behaviour, but also to infer the reasons behind this behaviour, and if possible, to discover the long term human behaviour [66, 114, 80].

For example, one could use human behaviour models to detect a specific behaviour. This is called prediction and the idea behind it is to find the most probable behaviour from a set of behaviours. In difference with simulation, models dealing with prediction usually assign a probability function to all possible behaviours, instead of just giving one solution. For example, to predict the behaviour of a user, one should look over all of the available behaviours and predict how probable they are [81].

From the above examples two main purposes for human behaviour modelling can be distinguished – simulation and inference. Furthermore, we adopt the inference types presented by Giersich [57, p. 113] and divide them into filtering, smoothing, and prediction.

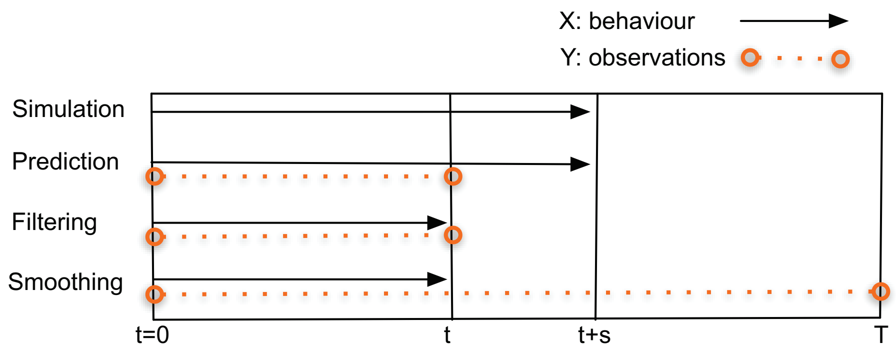


Figure 1.1: Purposes of human behaviour models. Here the estimated behaviour is represented by a solid line while the observations with a dashed line.

Simulation: Simulation is the process of imitating a real world situation or behaviour. Simulation does not take into account how probable the execution sequence is, it just gives a sample of a future trajectory. In simulation it is not possible to judge which is the best course of action, because the different samples are equally probable. Fig. 1.1 graphically shows the difference between the different behaviour purposes, where simulation is presented simply by a state sequence $X_{1:t+s}$ which is not affected by the probability distribution of the observed states[5].

Inference: Beyond simulation there is inference where we go one step further and try not only to simulate behaviour, but also to predict it and find the reasons behind this behaviour. When talking about inference, we distinguish 4 different approaches of interest.

- *filtering:* Filtering performs a transformation of the original data to approximated copy of it that is reduced of noise. In filtering an online learning is performed and the smoothing is done over only parts of observations. In Fig. 1.1 filtering is described as the process of estimating the state X_t at time t, taking into account only the observations Y_t up to time t, namely $P(X_t|Y_t)$.
- *smoothing:* Often the sensor datasets contain not only useful information but also a lot of noise. To avoid the redundant data, smoothing is employed. Smoothing is the process where an approximation of the original data is obtained, that tries to catch the information patterns but to leave out the noise or other fine-scale structures. Fig. 1.1 shows smoothing as the process where the state X_t at time t is estimated, taking into account all observations Y_T up to time T, or in other words $P(X_t|Y_T)$.

[5]Here *simulation* refers to the simulation process in statistical inference and Bayesian filters, where *simulating from a distribution* means to draw a sample from this distribution [101, p. 3]. It is not to be confused with the notion of *simulation* from the field of modelling and simulation, where it is possible to determine the probability of an observation by applying e.g. statistical model checking [87].

- *prediction:* In difference with simulation, prediction defines a probability distribution across an action space. Here we not only have a sample of a future trajectory, but also the probability of this action sequence happening. Prediction is extremely helpful for making decision about which action to support and which to discard. In Fig. 1.1 prediction is described as the probability of having the state X_{t+s}, given the observations $Y_{1:t}$, or shortly $P(X_{t+s}|Y_t)$.

In this work we are interested in human behaviour models for inference that can allow us to recognise the user actions and to further be able to reason about causes that influence those actions and parameters that are influenced by the actions. To do that, first we have to be able to identify the context information that can be incorporated in these models.

1.5 Employing prior knowledge

Prior knowledge, or context information, is the knowledge about the environment, the users and the relations between them we have prior to the present moment. In the context of activity recognition, this is the knowledge about the user activities and their relations to the environment or other users that is available prior to receiving the sensor data from which the actions have to be inferred. Additionally, it contains the information about the problem domain[6]. When creating a human behaviour model, this knowledge is incorporated into the model in order to improve the process of activity recognition and / or to provide additional information about the nature of the user actions.

As mentioned in the previous section, in the recent years there is increasing interest in using statistical methods for activity recognition. Thus the question of prior knowledge's importance arises. *Do we really need it or can we rely only on the sensor data?*

If we consider situation where the sensor data describes human behaviour in a particular domain and is collected with the same type of sensors, then a pattern extraction methods could be sufficient for learning the system to recognise future human activities. Especially, when assuming that humans are creatures of habit and exhibit certain behaviour patterns [22]. However, even changing the sensors type could be a problem for recognising the activity patterns. Even worse, a change in the domain would make activity recognition more difficult if not hardly possible. The reason for this is that by using only sensor data, a learned model is highly dependent on the observed data, so it will be difficult to use it in a different from the observed situation.

On the other hand, a human behaviour model making use of the prior knowledge could be more abstract and flexible, so that it can be used in various situations[7] and domains by utilising different types of domain knowledge. Another problem that the use of prior knowledge solves, is arriving at suboptimal solution. Geisler gives the following example with a first person shooter in Quake: when relying only on observation data to learn, it is possible that there is shooting only in 5% of the observations. Relying only on observations, the shooter learns not to shoot, thus arriving at an undesirable problem solution [53]. Of course, that also could be solved with selecting the appropriate type of training data, but as a whole it shows that one could rely on the model to cope only with situations it has already seen in the data. This problem too can be solved with obtaining more training data that however is expensive in terms of resources

[6]Here the meaning of *domain* complies with that of domain in knowledge representation where it is just some part of the world about which we wish to express some knowledge [123, p. 300].

[7]Here *situation* complies with the definition given by Russell and Norvig, where *situation* is the initial state of the world in a given domain before the agents begin performing their actions in order to achieve a goal [123, p. 388].

and time. Applying prior knowledge, on the other hand, could ensure that the model is doing whatever it is specified to do and without the additional costs for training data[8].

As hinted above, another issue with statistical methods is the expensive training data. In the recent years the sensor data to be analysed is increasing until we have come to the point where we have huge amount of observations and the process of analysing it becomes tedious and slow or sometimes even impossible [65]. A way to avoid this problem could be to employ prior knowledge that will replace the needed data with the expert knowledge of the model designer. This does not necessarily mean that the time needed for creating successful model will be shortened, but it will reduce the need of involving additional manpower for obtaining the needed amount of training data and the additional storage needed for this data.

In general, prior knowledge can be avoided in specific situations and only the sensor data can be used for model learning [5, 1, 117]. However, for applying such model on a broader spectrum of activity situations without the need of additional training data, as well as avoiding arriving at a suboptimal solutions because of insufficient training data, the incorporation of prior knowledge could be preferred.

1.5.1 Types of prior knowledge

Prior knowledge can come in different forms and from different sources. Here we propose a categorisation of prior knowledge into three groups based on the knowledge incorporated in different models.

Prior knowledge based on cognitive psychology: Cognitive psychology is the study of how people perceive, learn, remember, and think about information [139, p. 2]. Prior knowledge based on cognitive psychology consists of all the internal human states such as stress, emotions, perceptions etc. Such type of knowledge is important because cognition greatly affects human behaviour, and it is important to understand and take into account its influence on the user actions. Works that apply this kind of prior knowledge are such based on **A**daptive **C**ontrol of **T**hought – **R**ational (ACT-R) [128, 146]; such applying the BDI agent model [85]; even some Petri Nets approaches modelling emotional agents [43, 44].

Environmental knowledge: Environment is everything that surrounds a system and that exchanges different properties with it. In the context of human behaviour modelling, prior knowledge based on the environment is the knowledge about the state of the world outside the user. It includes information about the elements in the environment but also about the user interactions with it, and how this interaction changes the environment. Such knowledge is important, because it can be essential for determining different situation that may affect the way an activity is executed but still refer to the same activity. For example, environmental knowledge will describe the initial state of the user and the environment, such as what objects are there, where is the user, what has she already done etc. Approaches that employ environmental knowledge are **P**lanning **D**omain **D**efinition **L**anguage (PDDL) [114], **C**omputational **C**ausal **B**ehaviour **M**odels (CCBM) [80], **C**ollaborative **T**ask **M**odeling **L**anguage (CTML) [159].

[8]Here it should be noted that incorrectly incorporating prior knowledge or incorporating the wrong[9] kind of prior knowledge could also lead to suboptimal solutions.

[9]By *wrong* we mean knowledge that does not contribute to solving the problem or that incorrectly solves the problem.

Prior knowledge based on ergonomics: Ergonomics is the study that concerns the understanding of the interaction between human and other elements of a system and that strives to optimise human well-being and overall performance. Such type of prior knowledge is important, because it may contain important behavioural patterns that will make the recognition of a human activity easier. Approaches employing such kind of prior knowledge are **G**oals, **O**perators, **M**ethods, and **S**election rules (GOMS) [144] and **C**oncurrent **T**ask **T**rees (CTT) [58].

1.6 Types of human behaviour modelling approaches in the context of activity recognition

It was already explained that there are various types of activity recognition approaches that can basically be divided into data-driven which rely on training data in order to learn the human behaviour. Such examples are approaches that use Dynamic Bayesian Networks combined with clustering to learn the model [99], Markov models that rely on training data [133], and Hidden Markov Models [153]. The second type of approaches are model-driven which rely on underlying behaviour model in order to recognise the user activities. Examples of such are ACT-R where the user behaviour is encoded in terms of production system [66], PDDL that relies on precondition-effect rules to build the user behaviour [114], and CCBM that relies to similar rules [80]. As this work centres on symbolic models for activity recognition, here we discuss the different types of HBM formalisms that can potentially be applied to model-driven activity recognition systems.

1.6.1 Process-oriented modelling

When thinking of a process, one usually understands the act of executing a set of routine procedures in order to achieve a goal. Beaten [10] describes a process as

behaviour of a system. A system is anything showing behaviour, in particular the execution of a software system, the actions of a machine or even the actions of a human being. Behaviour is the total of events or actions that a system can perform, the order in which they can be executed and maybe other aspects of this execution such as timing or probabilities. Always, we describe certain aspects of behaviour, disregarding other aspects, so we are considering an abstraction or idealization of the real behaviour. Rather, we can say that we have an observation of behaviour, and an action is the chosen unit of observation.

The above indicates that a process is nothing more than description of the system dynamics, the actions it can execute, the order in which they can be executed, and any additional constraints or aspects that may influence these dynamics. In this context, process-oriented modelling describes behaviour through a set of actions that are temporally related.

In other words, process-oriented models answer the question *what is a user doing?*. There are two different model approaches concerning the process-oriented modelling. These are grammar-based models, where the human behaviour is described in the form of grammar and rules; and process calculi which represents a diverse family of related approaches for modelling of concurrent systems.

1.6.1.1 Grammar-based models

The human behaviour in a grammar-based model is described by a grammar that explains the behaviour. Russell and Norvig [122] define a grammar as

> *a finite set of rules that specifies a language. Formal languages always have an official grammar, specified in manuals or books. Natural languages have no official grammar, but linguists strive to discover properties of the language by a process of scientific inquiry and then to codify their discoveries in a grammar.*

Bernard Meyer [96] gives another definition by explaining that a grammar *defines the syntax of a language as a set of productions. Each production specifies one construct by describing the structure of specimens of the construct.*

In the context of human behaviour modelling, a grammar-based model describes behaviour in the sense of constructions of rules that define the dynamics of the activities constituting a behaviour. Examples of such modelling formalisms are GOMS [144, 86] that strives to model human-computer interaction, CTT that expresses hierarchical task models [58], and natural language modelling approaches to describing human behaviour [78].

1.6.1.2 Process calculus

Process calculi are various approaches for modelling concurrent systems. They provide a tool for describing high-level interactions, communication and synchronisation between different agents or processes. Another usage is for comparing and analysing independent processes. Although there are different types of process calculus, all of them share the same features. Namely, they represent interaction between independent processes as communication; they use a set of primitives and operators combining these primitives to describe the processes; they define algebraic laws for the process operators; they use equation reasoning to manipulate process expressions.

Beaten [10] describes process algebra as

> *the study of the behaviour of parallel or distributed systems by algebraic means. It offers means to describe or specify such systems, and thus it has means to talk about parallel composition. Besides this, it can usually also talk about alternative composition (choice) and sequential composition (sequencing). Moreover, we can reason about such systems using algebra, i.e. equational reasoning. By means of this equational reasoning, we can do verification, i.e. we can establish that a system satisfies a certain property.*

Examples of formalisms employing process calculus is Petri Nets [59] that among other applications are also used for simulation of social behaviour [44].

1.6.2 Causal modelling

Another perspective of human behaviour is the causal view or as Pearl explains – *our awareness of what causes what in the world and why it matters* [110]. Causality is the relationship between two events – the first being the cause, and the second – the effect that resulted from the first. In his book "Natural Philosophy of Cause and Chance" Born [16, p. 9] explains that *Causality postulates that there are laws by which the occurrence of an entity B of a certain class depends on the occurrence of an entity A of another class, where the word 'entity' means any physical object, phenomenon, situation, or event. A is called the cause, B the effect.*

In the context of Born's definition, causal models do not specify a set of actions with which a goal can be achieved, but rather define the preconditions for reaching it, and the effects after the goal has been reached, thus creating a structure of causally related states that lead from the initial to the goal state. In difference with process-based models which answer the question *what*, causal models deal with the problem of *why a user is doing something*, thus investigating the cause and effects of a given action sequence.

We should point out that when talking about causal models we do not consider those proposed by Perl where they are thought of as causal graphs with assigned probability values.

Pearl gives the following formal definition of a causal model.

> *A causal model is a pair $M =< D, \Theta_D >$ consisting of a causal structure D and a set of parameters Θ_D compatible with D. The parameters Θ_D assign a function $x_i = f_i(pa_i, u_i)$ to each $X_i \in V$ and a probability measure $P(u_i)$ to each u_i, where PA_i are the parents of X_i in D and where each U_i is a random disturbance distributed according to $P(u_i)$, independently of all other u.* [111]

In difference to this definition, we call causal models such that comply with Born's definition. Here, when talking about causal models, we consider two different types: forward rule-based models and backward rule-based models.

1.6.2.1 Forward rule-based models

Forward rule-based models deal with rules and facts. First rules and facts are defined and when the facts are true, they can make a certain rule applicable. When a rule becomes applicable, it is asserted. In difference with the process-based models, where an explicit process describes the system behaviour, the rule-based models continuously apply a collection of rules to a collection of facts. Rules can modify the collection of facts.

For example, a production system, which is a forward rule-based system, consists of two steps: the first is the prediction step or IF statement; and the second is the action step or THEN statement. This means that if the production's prediction matches the current state of the world, the production is triggered and a production's action is executed. Examples of forward rule-based models are ACT-R [128, 146], PDDL [114], and CCBM [80].

1.6.2.2 Backward rule-based models

In difference with the forward rule-based systems, where an action is triggered only if a fact is true, the backward rule-based systems use an approach called backtracking. As described in [126] backtracking *systematically searches for a solution to a problem among all available options. It does so by assuming that the solutions are represented by vectors $(v_1, ..., v_m)$ of values and by traversing, in a depth first manner, the domains of the vectors until the solutions are found.*

In the context of backward rule-based models, this means that given a problem, the algorithm goes through the present states and their relations and tries to find a solution. If any goal fails in the course of executing the algorithm, all state bindings that were made since the most recent choice-point are undone and the execution continues with the next alternative of the choice point. The most well known example of a backward rule-based formalism is Prolog [122, p. 339]. Among other applications it is used for implementing agents with advanced reasoning capabilities [118].

1.6.3 Probabilistic reasoning

So far we considered logic based approaches for human behaviour modelling that given a fully observable system would yield good results at inferring the user actions. However what happens when the user cannot be fully observed and there is some uncertainty about the observations' reliability? In such cases usually probabilistic reasoning is employed, which allows combining the capacity of probability theory to cope with uncertainty with the capacity of logic to exploit reasoning about the system structure and relations [111].

As Pearl explains, while causality connotes lawlike necessity, *probabilities connote exceptionality, doubt, and lack of regularity. Still, there are two compelling reasons for starting with, and in fact stressing, probabilistic analysis of causality...* [111, p. 1]. According to Pearl, the first reason is that even when using causal expressions to describe a given situation, it is usually the case that the situation contains uncertainty. The second reason he gives is that causal expressions are a subject to exceptions which may cause major difficulties when processed by standard rules.

This indicates that the combination of logic and probabilities should improve the model performance and increase its robustness in situations where the observations are unreliable. To develop such model one could take two different approaches – the first is using discriminative models where the relation between the model state and the observation is explicitly provided, or one could choose generative models where the Bayes rule is used to compute the probability of the state based on the observation.

More formally, if we assume we have two correlated random variables X and Y, where $P(X,Y) \neq P(X)P(Y)$ and where Y is observable while X is hidden, then we can use the observed value $y \in Y$ to infer the density $P(X|y)$.

The **generative approach** provides models for $P(Y|X)$ and $P(X)$, then the Bayes rule is used to compute $P(X|y) = \frac{1}{P(y)}P(y|X)p(X)$.

On the other hand the **discriminative approach** assumes that a model $P(X|Y)$ is directly provided.

Examples of generative approaches are CCBM that provide causal models that are compiled into probabilistic runtime models [80], an extension of PDDL that allows probabilistic reasoning based on observations [114], and an extension of ACT-R that allows computing the probabilities for each execution trace [66].

Discriminative approaches on the other hand are many ontology based approaches where the observations are matched against a library of actions or plans [104, 119]. Another approach making use of libraries of actions is proposed by Maier et al. [93] where probabilistic hierarchical constraint automata is translated into Bayesian logic network. However such approaches are inherently unable to solve the problem of the libraries completeness as it would be an impossible task for the model designer to manually model all possible execution sequences. This problem is solved by the generative approaches which based on their specification are able to automatically generate all valid plans and map them to the corresponding observations in a probabilistic manner.

1.7 Challenges with human behaviour models for activity recognition

As it could be seen from the sections above, human behaviour models are often used for activity recognition. Regardless of the variety of human behaviour modelling approaches and formalisms, there are still some challenges associated with creating successful models for activity recognition. Here these challenges are summarised and discussed.

Challenges with the training data: Many approaches for activity recognition rely on training data in order to fit the model for recognising the user actions or in order to extend the initial model [140, 104, 8]. These approaches show promising results, still there is the drawback of collecting and annotating training data. In order for the model to be able to recognise a broader spectrum of behaviour variations, it has to have been trained with training data that contains those variations. This in itself includes preparing the experiment from which the data will be collected, finding participants, conducting the experiment, and finally annotating the data. This is expensive and time consuming task in terms of manpower and time needed from the beginning of the planning to obtaining the training data. Even more, it is often the case that the data contains errors or the sensors were not working and the experiment has to be repeated. Additionally, one can rely that the system will be able to recognise only activities that occurred in the training dataset. One solution to this problem is substituting training data with prior knowledge.

Challenges with the behaviour variability: As mentioned above, the model should be able to cope with the behaviour variability, which in the case of trained models leads to the problem described above. In the case with manually encoded behaviour [94, 158], unless the problem is trivial and restricted, it is almost always the case that the model designer is unable to encode all behaviour variations. In such cases what is usually done, is that the most often occurring behaviour is modelled and improbable actions' combinations are omitted. This approach works in situations where the environment and the user actions are carefully controlled and there is no danger of unexpected behaviour. However, what happens when the user decided to complete the task in a way not encoded in the model? One solution to such problems could be the employment of generative approaches that allow based on their precondition-effect specifications to generate all possible behaviours without the need of explicitly encoding them.

Challenges with the model reusability: Often a model is developed solely for a specific scenario and it is later impossible to reuse it, or parts of it, in another model [158]. From a performance point of view this is not a problem, however developing a model from scratch means that the model designer needs more time for implementing it than if she had components that can be reused. This problem can be solved in different ways – one option would be to apply reusable action templates that are later parameterised with problem specific parameters, as proposed by Hein et al. [64]. Another option would be introducing design patterns of commonly occurring problems similar to those in software engineering [52].

Challenges with model traceability: In the field of software engineering there are already well established development methods that provide the information needed for easy tracking of software solutions that could cause problems [135]. In the field of context-aware systems and activity recognition however, that is not a common practice. As Helal et al. [70] explain, context aware applications centre more on the runtime model and not on the model development process. This could cause a variety of problems especially if the designer is dealing with a complex model. In that case it becomes almost impossible to trace the reason behind using a given modelling solution, and from that to discover design problems in the model implementation.

Challenges with results reproducibility: A problem that is often not mentioned in works about activity recognition, is the ability to reproduce the obtained results. As Gordon et al. [62] explain, *one of the major issues which we see in this field is the reproducibility of results. While methodologies and algorithms may be well defined and formalized, re-implementation is time consuming and effort intensive.* It is often the case that change in the model parameters, the evaluation procedure, or even the tool used for obtaining the results will produce different outcome. Even more, results obtained without documentation about the involved process elements (in terms of models, scripts, parameters, etc.) usually render the results unreproducible [82].

1.8 Goal of the thesis

In the previous sections we introduced the concepts related to model based activity recognition and the challenges associated with this kind of activity recognition. Based on them here we present the goal of this work and the contributions it makes to the field of model-based activity recognition.

The work aims at (1) *empirically showing that activity recognition based on symbolic human behaviour models is applicable to the domain of daily activities*; and (2) *introduces a structured development process for developing such models that produces well documented and reproducible models.*

To achieve (1), the work introduces three modelling problems from our daily life and identifies the requirements a modelling formalism needs to possess in order to successfully model the problems. Based on them a suitable modelling formalism is selected and the problems' solutions are presented and analysed in terms of model parameters and model performance.

To achieve (2), the work analyses the developed models and introduces a modelling toolkit that contains solution patterns to frequent problems in the models. Additionally, the toolkit's applicability is evaluated by applying the solutions for the three models from (1). Furthermore, based on the models' analysis and the analysis of existing development processes, a structured development process is proposed.

1.9 Outlook

The thesis is structured as follows.

Chapter 2 introduces the three modelling problems that are discussed throughout the thesis. Based on them, the requirements for human behaviour modelling are identified and candidate formalisms are evaluated.

Chapter 3 gives an introduction to the selected modelling formalism. Later it introduces the intuitive models for the three ADL problems and analyses their parameters and performance in order to identify successful modelling solutions and problems during modelling.

Based on the model analysis in Chapter 3, Chapter 4 introduces the modelling toolkit that contains solution patterns to frequent problems. The patterns are evaluated based on their influence on the model performance and parameters.

Chapter 5 introduces a structured development process based on the modelling experience made in Chapter 3. It also discusses how this development process differs from existing development processes and provides a practical guide to developing successful human behaviour models for activity recognition.

Finally, the work concludes in Chapter 6 where the process of modelling human behaviour is discussed, its advantages and drawbacks, as well as the challenges it poses for the field of activity recognition.

Chapter 2

Modelling Preliminaries

"No sensible decision can be made any longer without taking into account not only the world as it is, but the world as it will be."
Isaac Asimov

Chapter Summary: *This chapter describes the problems that are to be modelled throughout the thesis. It also discusses the choice of the use cases, and additionally provides domain analysis of the problems to be modelled. Moreover, it discusses the requirements a modelling formalism should satisfy in order to be able to model the three use cases. Furthermore, it provides an insight into the state of the art of relevant modelling formalisms and discusses their suitability for the problems at hand.*

Chapter Sources: *This chapter is partly based on the technical report "Toward a Unified Human Behaviour Modelling Approach" [160], and the journal paper "Towards Creating Assistive Software by Employing Human Behavior Models" [79].*

Questions to be answered in the chapter:

What types of behaviour dynamics are there? (In Section 2.2)

What are the problems to be modelled? (In Section 2.3)

What are the requirements for modelling these problems? (In Section 2.4)

What are the candidate modelling formalisms? (In Section 2.5)

How to choose the appropriate modelling formalism? (In Section 2.6)

2.1 Introduction

Capturing the complete diversity of human behaviour in the everyday life is probably an impossible task, as the user behaviour is dependent not only on the available actions, but also on personal preferences, the environment, the concrete situation or any number of other unforeseen factors that could influence the user decision making [34]. However, a human behaviour model is a simplified and abstracted representation of reality, thus it should be possible for one to define the purpose of the model and based on that to isolate the most relevant factors affecting the resulting behaviour. This should result in a representation of the real world that contains sufficient knowledge about the problem domain and the user in order to provide rich context information.

To represent different aspects of human behaviour that are common for the activities from our daily life, three use cases are selected that represent three common types of user behaviour correlations. The use cases are then analysed for features and factors that have to be included in the problems' solutions. Based on the analysis, the requirements a modelling formalism should possess were identified. Finally based on the requirements, a modelling formalism was chosen that is to be used for implementing the problems' solutions.

2.2 Behaviour dynamics and their dependencies

In the context of this thesis, two types of user behaviour are considered – the first is a single agent behaviour whereas the second is a multi-agent[1] behaviour.

In multi-agent systems, different types of agent behaviour and their dependencies on other agents are observed. In such system, each agent is not only interacting with the environment where she is situated but also with one or more other agents residing in the same environment. This creates a complex system where the agent state and those of the environment depend on multiple entities all of them having the ability to influence the given situation. According to Wooldridge there are various types of interaction in a multi-agent system which create four types of dependencies between the agents [155, p. 125].

Independence: There is no dependence between the agents and their actions.

Unilateral dependence: One agent depends on the other, but not vice versa.

Reciprocal dependence: The first agent depends on the other for achieving some goal, while the second depends on the first for some other goal, where the goals are not necessarily the same.

Mutual dependence: Both agents depend on each other when following the same goal. Note that mutual dependence implies reciprocal dependence.

Based on the types of dependencies, here we introduce a categorisation of human behaviour in multi-agent systems. It presents three main groups of behaviour correlations.

Uncorrelated: The uncorrelated behaviours of two or more entities have neither physical nor intentional interactions between them. This group also comprises the behaviour of single agents where there is no one else to interact with in the environment. Such correlation has the property of *independence* described above.

Physical correlation: The physical relation between the behaviours of two or more entities is based on physical interactions with the environment and or the rest of the entities, causally affecting the physical world in which all agents are located. This type of relation has the properties of *unilateral dependence* and / or *reciprocal dependence*.

[1]Throughout the thesis the terms *user*, *person* and *agent* are used interchangeably.

Intentional correlation: The intentional correlation between the behaviours of two or more entities is represented not only by physical correlation between them, but also by the conscious decision of these entities to follow a common goal or goals. This type of correlation has the properties of *reciprocal dependence* and *mutual dependence*.

Based on the types of behaviour correlation, throughout the thesis three types of problems are considered, each of them represented by a concrete use case.

Problem describing uncorrelated behaviour: The uncorrelated behaviour problem aims at describing behaviour where a single person is interacting with the environment while executing a goal oriented task. The user behaviour is independent of any other agents' actions but still depends on the environment and the person's capabilities to interact with it.

Problem describing physical correlation: During physical behaviour correlation the users do not coordinate their actions with each other and do not follow a common goal. Each user is acting independently of the rest of the agents or interacting with them in a competitive manner. The actions of the user either do not affect the rest of the users, or when they affect them, it is only to further the agent's own goal. Thus her actions either influence the rest of the users in a negative manner, or when in positive – it is an unconscious side effect of reaching her own goal and in the process affecting the physical world.

Problem describing intentional correlation: During intentional behaviour correlation the users coordinate their actions with each other while pursuing a common goal. The actions of each single user have effect on the actions of the rest of the users, and each user aims at contributing to the achievement of the common goal that is intentionally chosen by all involved parties.

2.3 Use cases – analysis

This section describes the three scenarios from the daily life domain and analyses the problem domains. The use cases are a 3-person meeting, illustrating intentional correlation; a cooking task, describing an uncorrelated behaviour and a multi-user office scenario, that represents both uncorrelated behaviour and physical correlation (see Fig. 2.1). The analysis of each use case is structured so that it gives the problem motivation and later describes the problem domain with enough details for the designer to be able to identify the elements to be modelled and the model objectives. Additionally the detailed problem description is used for deriving the requirements a modelling formalism needs to possess in order to be able to model the problems. Furthermore, the choice of sensor infrastructure is based on the problem elements that were identified to be modelled, so that a test dataset can be recorded for evaluating the model performance.

For each use case, the section is divided in 4 parts – the first is the motivation behind using exactly that problem for illustrating the given behaviour relation; the second is the problem description, where the users behaviour, objectives and the environment are described; based on that, the third section identifies the elements that are to be modelled and the objectives the given model has; the following part discusses the sensors needed to capture the desired user behaviour and the resulting dataset with which the future model is to be tested and evaluated.

The final result of this analysis is a detailed specification of all elements involved in the model implementation and evaluation – these are the model components, the sensors infrastructure and datasets and the objectives the models have.

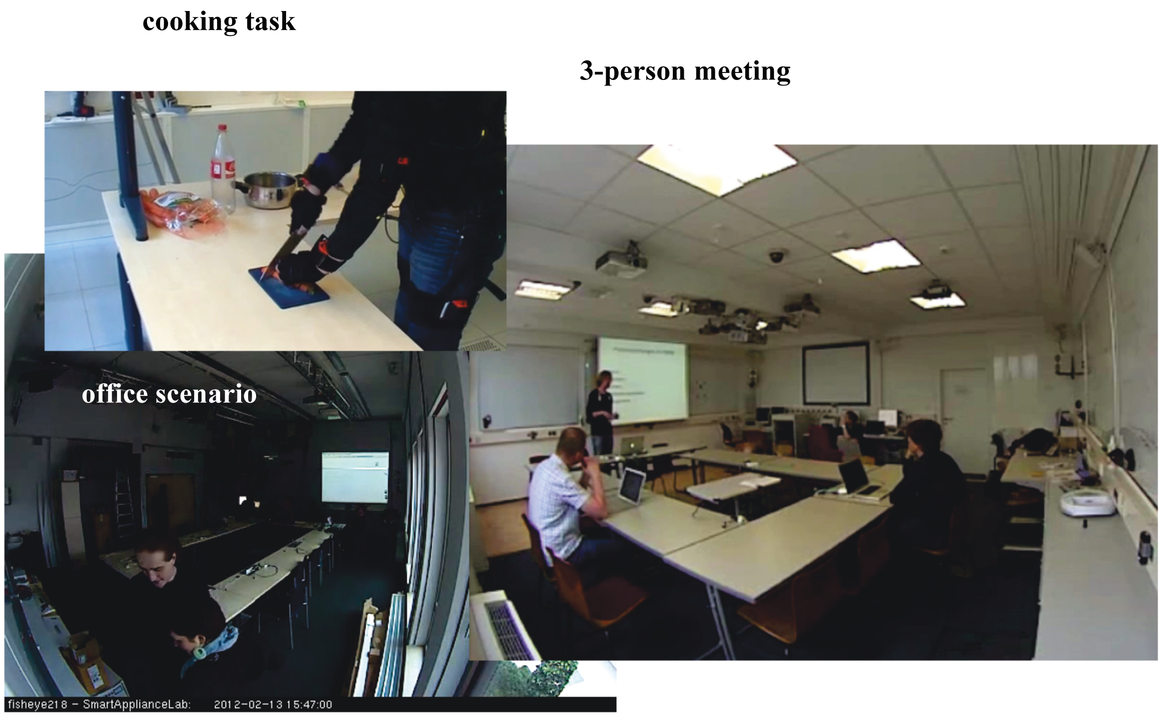


Figure 2.1: The three use cases illustrating the modelling problems. Clockwise from right to left: the three person meeting, the office scenario, and the cooking task. All experiments were conducted in the SmartLab of the Mobile Multimedia Information Systems Chair, University of Rostock [9].

2.3.1 3-person meeting

The first scenario is a three person meeting that aims at describing the behaviour of a team of users trying to achieve a common goal.

2.3.1.1 Motivation

The 3-person meeting is a typical problem from the smart environments domain where assistive software needs to know the user's current state and intentions in order to be able to proactively assist her in achieving her objectives. It is also a typical example of an intentional behaviour correlation where the three users coordinate their actions in order to achieve a common goal – namely the completing of the meeting and leaving the room. Thus, on a more coarse-grained level the users exhibit a team behaviour that is supported by the actions of the separate team members. As the users all have the same team level objective, they also do not interfere or contradict the actions of the remaining team members. However, on a more fine-grained level it is a typical multi-agent behaviour problem, where each of the users acts as a separate autonomous agent that contributes to achieving the common goal.

The reason for choosing this particular use case as an example of an intentional correlation, is that it is relatively simple in terms of environment elements and number of users, yet it contains the basic features making up an intentional correlation – multiple users, common goal, and synchronisation between the actions of the different users in order to achieve this goal.

Table 2.1: Elements of the meeting team behaviour and environment. Here the number of users, actions, objects and locations, as well as the initial and the goal state are described.

Types of elements	**Instances**
Users	one abstract user representing the team
Actions	enter; move; presentA; presentB, presentC, discuss; leave
Objects	none
Locations	door, stage, seat
Initial state	the team is outside the meeting room
Goal state	the team is outside the room after having presented 3 times and discussed the presentations
Execution length	at least 9 actions that need to be executed in order to reach the goal

2.3.1.2 Problem description

A meeting takes place in a meeting room (see Fig. 2.1, right). There are 3 participants that are in the room during the meeting, each of them is supposed to make a presentation with the option of a discussion after the end of the third presentation. At the beginning of the meeting the 3 participants enter the room, two of them go to their respective seats and one goes to the stage area in order to prepare her presentation. After the first presentation is over, the presenter goes to her respective seat, while the second presenter starts her presentation. The same procedure is repeated for the third participant and after the last presentation, the participants have the option to make a short discussion regarding the presentations. The order in which the presentations are made is arbitrary and could be performed in any order, namely if the presentation of the first user is denoted by A, of the second by B, and of the third by C, then

```
(ABC) or (ACB) or (BAC) or (BCA) or (CAB) or (CBA).
```

After the last presentation, or after the discussion respectively, the participants get up from their seats and leave the room. The goal of the participants is that all of them have presented, optionally discussed the presentations, and finally left the room.

There are several locations defined in the room: three seats, three stages, and a door area. The objective of the system is to recognise the users' current actions and to discover if they have reached their team goal at the end of the meeting.

2.3.1.3 What is to be modelled?

As the problem could be considered both as a single team behaviour, or as a multi-agent behaviour, its solution could also have two representations: one describing the coarse-grained team actions, and the other, being able to explain the single users' behaviours behind the team actions.

Table 2.1 shows the elements that are to be modelled in the case of team behaviour. It can be seen that a lot of abstractions are made: the three users are represented as a single team agent, the *move* action is represented by a single action that could consist of any or all of the users moving; also the locations are simplified and represented by just one single sitting area, and one presenting area. Such model elements oversimplify the problem, but on the other hand are more than enough to represent the team behaviour, especially when each of the agents' actions is governed by a common goal that defines their choices.

In difference with the team behaviour, the multi-agent behaviour should be able to also represent the actions of the single users in the context of the common goal. Table 2.2 shows

Table 2.2: Elements of the meeting multi-agent behaviour and environment. Here the number of users, actions, objects and locations, as well as the initial and the goal state are described.

Types of elements	**Instances**
Users	userA; userB; userC
Actions	enterA, enterB; enterC; moveA; moveB; moveC; presentA; presentB; presentC; discussA; discussB; discussC; leaveA; leaveB; leaveC
Objects	none
Locations	door, stage1, stage2, stage3, seat1, seat2, seat3
Initial state	the three users are outside the meeting room
Goal state	the three users are outside the room after each having presented and discussed the presentations
Execution length	about 20 actions that need to be executed in order to reach the goal

the elements that need to be modelled in such case. It can be seen that now the actions that have to be tracked are not for the team as a whole, but for the separate users. Of course, their actions are still synchronised or dependent on each other but their cooperation is based on the agents' nondeterminism[2]. Also now the goal consists of the separate goals of the three users and in order the overall team goal to be achieved, each of the agents has to have achieved her own goals.

2.3.1.4 Sensor infrastructure and datasets

As the application of the models is activity recognition, the final step is the decision about the sensor infrastructure which is to capture the modelled behaviour. The choice of the sensors could depend on different factors, such as the *sensors affordance* (or can we get the sensors based on our budget, the sensors availability, experimental infrastructure, or other external factors that could limit our choice); *the model objective* (or what exactly we want to observe and recognise); *sensors accuracy* (or within what limit the sensors can deviate from reality) etc.

The choice of sensors in this case was the UbiSense localisation system [147] that is an ultra-wide band indoor location system that uses active RFID tags to detect the user location. Fig. 2.2 shows an extract of the observations from the meeting dataset. Here *flag* indicates

```
flag-A  x-A       y-A       flag-B  x-B       y-B       flag-C  x-C       y-C
1       266.518   133.157   1       143.319   -170.781  1       212.95    -3.945
1       292.253   147.207   0       143.319   -170.781  0       212.95    -3.945
0       292.253   147.207   0       143.319   -170.781  1       205.842    2.184
```

Figure 2.2: Example observations from the 3-person meeting dataset. Here *flag* indicates whether there is any change in the position, x is the x-axis and y is the y-axis that together give a 2D position of the user in centimetres.

whether there is change in the user's position (1), or not (0). After the flag, for each user there are her UbiSense coordinates (in centimetres) in x and y direction.

Based on the use case and the chosen sensor infrastructure 21 experiments were conducted. 20 of them where staged[3] 3-person meetings with varying presentation order, and with a discussion at the end of the third presentation. Each of the meetings lasted about 3 minutes and was conducted in a smart meeting room supplied with UbiSense sensors. Each dataset contained between 2637 and 3176 samples. The last experiment was a real 3-person meeting that

[2]Here by *nondeterminism* we mean the agent's ability to make seemingly arbitrary choices.

[3]Here *staged* indicates that the experiment was conducted according to a predefined execution sequence.

was not staged, and continued about 50 minutes and where the users decided not to have a discussion at the end of the meeting. The meeting also took place in the same meeting room and was recorded with the same sensors. It had 62 295 samples. Additionally, the experiment was recorded with cameras so that the datasets could later be accurately annotated.

2.3.2 Cooking task

The second use case is a cooking problem where a single user is preparing a meal, having lunch and then washing the dishes. The objective is for the user to successfully complete a typical cooking task problem.

2.3.2.1 Motivation

The cooking problem is a **K**itchen **T**ask **A**ssessment (KTA) problem [14], where the aim is to detect whether the person is executing the task in the correct order, and if we stretch it further to an assistive problem, the system would like to detect inaccuracies in the user behaviour and assist her in correcting her mistakes and successfully achieving the task objective. This problem is a typical **A**ctivities of **D**aily **L**iving (ADL) problem and it has applications where the user could suffer from Alzheimer or similar diseases but the disease is still in early stages and she wants to preserve her independent lifestyle for as long as possible [128]. The solutions of such kind of problem are extremely important in Europe where the elderly population is increasing while the birth rate is decreasing [38, 13].

The cooking task is also a typical single user behaviour and thus uncorrelated, where the agent's actions do not depend on other users' behaviour but where she interacts with the surrounding environment, thus making her behaviour dependent on the environmental factors. It is also a problem where there is a lot of interaction with different objects. This requires modelling the problem on a more fine-grained level compared to the meeting scenario.

The reason for choosing this particular example is that it presents a simple cooking recipe, with just a few ingredients, yet it possesses all basic interactions with the environment taking place in a cooking situation (manipulating objects used in the everyday's tasks in the kitchen, using the cooking appliances, setting the table, eating, and cleaning). This allows us to explore the complexity of a cooking task and a real world single user behaviour without running into unnecessary details of complex meals.

2.3.2.2 Problem description

A person is cooking a carrot soup in a kitchen supplied with the necessary kitchen appliances. The person starts by washing her hands, then cutting the carrot, putting it into the pot and cooking it. Later after the meal is ready, she serves it in a plate, puts water in a glass, and sits on the table to have a lunch. Finally, after the person has eaten and drunk water, she stands up, goes to the sink and washes her utensils. There are generally the following experiment stages that should be executed in the same order.

```
wash hands -> prepare to cook -> cook carrot soup -> serve meal
               -> eat and drink -> clean up
```

On the other hand, the intermediate actions that take place (such as fill plate, fill glass, move, etc.) can be executed in any causally correct order. The locations in the kitchen are sink, counter and table, which are locations that could be reached only by walking from one place to another. Additionally, there are fixed locations, or places, which from a certain locations can be reached only by moving the hand. The places are cupboard and oven, which could be reached from the counter. Furthermore, different objects with varying functions and properties are used: cutting board, pot, plate, glass, bottle, knife, spoon, sponge, and the additional water and carrot.

2.3.2.3 What is to be modelled?

As the cooking task involves a lot of fine-grained activities, it is a more complex problem than the 3-person meeting (regardless of the fact that in the latter we have people acting in parallel). Thus also the behaviour and environment elements are much more than in the meeting scenario.

Table 2.3: Elements of the cooking task behaviour and environment. Here the number of users, actions, objects, locations, and places as well as the initial and the goal state are described.

Types of elements	Instances
Users	one default user
Actions	wash, wait, move, take, cut, put, turn on, cook, turn off, open, fill, close, sit down, eat, drink, stand up
Objects	cutting board, pot, plate, glass, bottle, knife, spoon, sponge, water, carrot
Locations	sink, counter, table
Places	cupboard, stove
Initial state	the user is at the sink
Goal state	the user is at the sink after cooking, eating the meal and washing the dishes
Execution length	about 80 actions that need to be executed in order to reach the goal

Table 2.3 shows the elements that are to be modelled. It can be seen that there are 16 actions that can take place, and most of them (except for wait, move, sit down and stand up) involve manipulating any of the given objects. Additionally, there are 10 objects in the environment that are manipulated by the user that indicates the need of a mechanism to cope with a high number of different choices the model will be faced with (See Fig. A.1 in Appendix A for the locations of the objects throughout the task).

2.3.2.4 Sensor Infrastructure and Datasets

```
##locations###
sink  moving  counter table
0.0   1.0     0.0     0.0
0.0   0.0     1.0     0.0
##fixed places###
sink  moving  counter stove  cupboard  table
0.0   1.0     0.0     0.0    0.0       0.0
0.0   0.0     1.0     0.0    0.0       0.0
##objects###
carrot  knife c_board pot    w_spoon plate glass bottle  spoon sponge
0.0     0.0   0.0     0.0    0.0     0.0   0.0   0.0     0.0   0.0
1.0     0.0   0.0     0.0    0.0     0.0   0.0   0.0     0.0   0.0
```

Figure 2.3: Example observations from the cooking task dataset. Here the observations are either 0 when no sighting at this location / place / object was observed, and 1 when a sighting was observed. One can observe the locations, the places and / or the objects.

Fig. 2.3 shows an extract of the observations from the cooking task dataset. The first row shows the observations for the locations with 1.0 indicating that the person has been observed at the given location. The second row shows the observations for the fixed places. The last row shows the observations for the objects being observed. Seven cooking tasks were recorded. Each of the tasks lasted about 7 minutes and although the task at hand, namely cooking, was staged, the behaviour of the participants while achieving their goal was left to themselves. This resulted in different execution paths leading to the goal state and increased the model variability needed to be able to recognise the correct behaviour. Additionally, the datasets contained between 636 and 1207 samples.

2.3.3 Office Scenario

The last use case is an office scenario where one to three users act independently of each other in order to achieve their separate goals.

2.3.3.1 Motivation

In the office scenario the users are acting in unsynchronised manner and each of them is following her own goal. The use case could be an example of a smart office, where the system has to support several different users who do not have a common team goal and who could act in a competitive manner. The users either do not interact with each other, or when interacting it could lead both to positive or negative effects on the separate user's goals. Although in general the office scenario is a multi-user scenario, in the scenario variations there are multi-user as well as single user instances in the cases when only one person was present in the office.

The goal of the system is to recognise the user actions and identities (namely who executed which action) based on coarse-grained information about a location being occupied. For that reason, the model should be able to represent not only coarse-grained activities but also actions that involve objects manipulation and user identification. It could be said that this use case is the middle ground between the meeting scenario and the cooking task as there are objects that the model has to reason about like in the kitchen task assessment, but on the other hand here we have a multi- agent behaviour that is observed based on location information. However, in difference with the meeting scenario, here the users do not try to cooperate or act in a synchronised manner.

2.3.3.2 Problem description

In the office scenario a varying number of users enter an office room that contains a printer and a coffee machine. The objective is to print some documents and / or to get a cup of coffee. The behaviour of the users is either uncorrelated or physically correlated and everyone is independently choosing their actions. To make the problem more complicated, it is possible that the water or the ground coffee in the coffee machine are not enough, and in such case the coffee machine has to be refilled with ground coffee and water before making coffee. Additionally, it is possible that there is no paper in the printer, or that paper is stuck inside and the printer has to be repaired.

The goal of the users is to have their documents printed and to get coffee. The order in which the actions are executed, or the persons who perform the different tasks is arbitrary.

2.3.3.3 What is to be modelled?

Table 2.4 contains the elements of the user behaviour and the environment. It can be seen that the number of users varies. This is due to the fact that the behaviour is unsynchronised and any of the users could decide for herself to appear in the office or leave it at any time. Additionally, the actions to be modelled are fine-grained actions that involve the manipulation of different objects, and any of the actions can be executed by any of the agents. It is possible that the agents incidentally help each other achieving the goal, or block each other, but otherwise each of them makes her decisions based on her own goals.

Table 2.4: Elements of the office behaviour and environment. Here the number of users, actions, objects and locations, as well as the initial and the goal state are described.

Types of elements	**Instances**
Users	one, two, or three users
Actions	move, take, put, refill ground coffee, refill water, repair printer
Objects	paper for the printer, water, ground coffee
Locations	door, printer, paper stack, coffee machine, water tap, coffee jar
Initial state	the office is empty
Goal state	the user(s) has (have) printed the documents, taken the coffee and left the room
Execution length	5 to 20 actions that need to be executed in order to reach the goal

2.3.3.4 Sensor Infrastructure and Datasets

The types of sensors used were the SensFloor sensors which uses radio modules and proximity sensors to detect user presence without providing any additional identity information [51, 138]. Using the SensFloor, it provided binary observations whenever someone was present at the given locations (namely 0 for no presence, and 1 for presence). Fig. 2.3 shows an extract

```
Door  Printer Coffee-Machine  Paper-Stack Water-Tap Coffee-Jar
0     1       0               0           0         0
0     0       0               1           0         1
1     0       1               1           0         1
```

Figure 2.4: Example observations from the office scenario dataset. Here for each location 0 indicates that there was no sighting, while 1 indicates that someone was at the given location.

of the observations from the office scenario dataset. It can be seen that there are multiple sightings at the same time step but no information about the user identity at the given location. This means that the model to be built should be able to reason not only about the users' actions but also about who is associated with which action and location.

Based on the problem domain and the sensor infrastructure 6 experiments were conducted with varying number of participants: 4 one-person datasets, one two-person and one three-person dataset. The duration of the different experiments was between 50 and 200 time steps. The users acted autonomously and in unsynchronised manner without any verbal communication. This resulted in different action execution sequences as well as in different users performing different set of tasks in the two multi-user experiments.

2.4 Requirements for HBM

In order to choose an appropriate modelling formalism for the problems, we have to first identify the dynamics and relationships between the elements that build up the given user behaviour. For that the modelling formalism should possess certain properties (or requirements) in order to be able to successfully capture the targeted behaviour. These requirements are similar to the software engineering requirements and could be considered as a special case. From a software engineering perspective a requirement was formally defined in the IEEE 610.12-1990 standard [71] in the following way.

(1) A condition or capability needed by a user to solve a problem or achieve an objective.

(2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.

(3) A documented representation of a condition or capability as in (1) or (2).

In our case a requirement is a condition or capability needed by a model designer to solve a modelling problem or to achieve an objective. This capability should be possessed by the modelling formalism used for solving the problem.

By analysing the problem domains and the modelling objectives, the set of properties can be derived that represents the requirements needed for describing the users' activities. This is done by identifying all relations between the actions derived in Section 2.3. For example, in the meeting scenario, the person enters the room and only after entering she is able to move to the seats or the stage. This indicates the properties *sequence* and *dependence* of an action on another action. Meanwhile, in the same scenario, there is another user acting in the environment, which indicates the requirement for modelling *parallel* actions. This process is repeated for all actions and their relationships with the rest of the actions in the dataset, and then for all three problems. Fig. 2.5 shows the process with which the requirements were identified. It

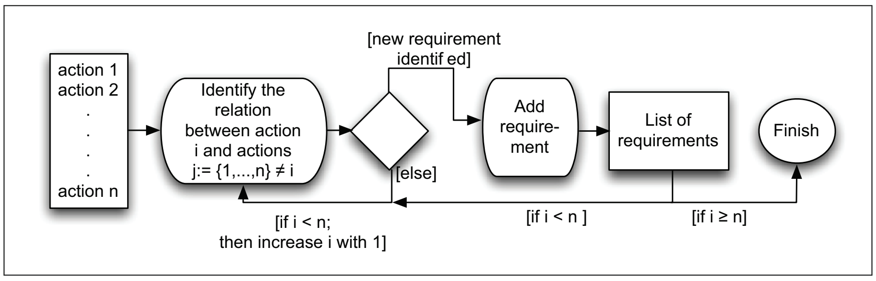

action 1
action 2
.
.
.
.
action n
Identify the relation between action i and actions j:= {1,...,n} ≠ i
[new requirement identif ed]
[else]
Add require-ment
List of requirements
Finish
[if i < n; then increase i with 1]
[if i < n]
[if i ≥ n]

Figure 2.5: The algorithm shows the procedure for identification of requirements that are needed for describing human behaviour dynamics.

can be seen that the relation between each action in a given problem and the rest of the actions for this problem is identified; later if the relation was not previously identified for other pair of actions, it is saved in a list containing all the requirements for all problems. The concrete relations between the actions are defined in the next section.

This process resulted in 14 requirements that a formalism should possess in order to be able to successfully describe the behaviour dynamics in the problems from Section 2.3. Additionally, based on the model application, four more requirements were identified that are needed for achieving the model purpose. They were derived based on the designer's experience in the field of activity recognition.

For that reason, the identified set of requirements for all three use cases is divided into two main groups – behaviour based requirements and application-based requirements.

Behaviour-based requirements

- *Requirements for procedural modelling:* composition, sequence, repetition, interleaving activities, choice, enabling, disabling, priority, independence, dependence;

- *Requirements for parallel execution modelling:* parallelism, synchronisation, suspend, resume;

Application-based requirements

- *Requirements for probabilistic modelling:* observation models, probable durations of activities;
- *Requirements for modelling purpose:* activity recognition, unobserved actions;

In the following section, the requirements are explained in detail, as well as the need for having them.

2.4.1 Behaviour-based requirements

To formally describe the requirements that represent the behavioural relations between the different actions, the notation of **C**ommunicating **S**equential **P**rocesses (CSP) introduced by Hoare [67] is used. In it a process is described as a pattern of behaviour in which a given object can be involved. Each process then consists of a set of events, or actions of interest which are called the process's alphabet [67, p. 23]. A detailed description of CSP and the corresponding notation can be found in Appendix B. Here we make use of these notations and express the requirements in terms of events and processes. To do that the notion of *action* as given in Definition 2 (in Chapter 1, page 5) is considered to be equivalent to that of *event* in CSP, and the notion of *behaviour* as given in Definition 4 (in Chapter 1, page 6) to be equivalent to *process* in CSP (for more information see Appendix B).

Sequence: The first requirement for expressing user actions is the ability to execute actions one after another. It is the most essential of the modelling requirements, as without being able to execute actions sequentially, it would be impossible to represent any execution path leading from the initial world state to the goal. For example if we first enter the room, then go to the stage, we say that these two actions are executed sequentially. In terms of CSP we express sequential action execution based on the definition of traces (see the explanation of *traces* in Appendix B, page 204).

Definition 8. ***(Sequence)*** *Given a process X with an alphabet αX, and events x and y such that $\{x,y\} \in \alpha X$, then a sequence is represented by the trace $\langle x,y \rangle$ indicating that x was executed before y.*

Parallelism: The next requirement is that of parallelism which allows executing actions concurrently. This requirement is essential in multi-agent situations where several users are acting in the same environment or when several actions can be executed at the same time. For example, if two persons are walking in the room, we say that they are executing the action walk in parallel. To express parallelism in terms of CSP we use the notion of concurrency as explained in Appendix B (page 205).

Definition 9. ***(Parallelism)*** *Given two processes X and Y, we call them parallel if the events of the first process are possible without the events of the second process and vice versa, and if both processes are executed concurrently. We denote such process as $X||Y$.*

Repetition: Repetition allows executing an action multiple times. This requirement is necessary in cases where the user is doing the same action several times like for example eating repeatedly, or washing the same utensil several times. Here we assume that the recursion is not terminated by an external factor but rather by some internal unobserved state. To define repetition according to the CSP notation we use the concept of recursion explained in Appendix B (page 204).

Definition 10. ***(Repetition)*** *Given a process X with an alphabet αX, and an event x such that $x \in \alpha X$, then we call repetition the process where $\mu X : \alpha X.(x \rightarrow X)$.*

Non-deterministic choice: Non-deterministic choice allows selecting between several actions when there are no external factors that can control the decision of which action to be selected. This requirement could be interpreted as the user's *free will* to perform a given action. For example, if we enter the room and have the option either to sit down, or go to the stage, but there is no other external factor that can influence our decision. To define the requirement according to the CSP notation we use the concept of non-deterministic choice explained in Appendix B (page 205).

Definition 11. ***(Non-deterministic choice)*** *Given two processes X and Y, then we say that X or Y is chosen if the resulting process will either behave as X or as Y and the selection is done arbitrarily. We denote non-deterministic choice as $X \sqcap Y$.*

Enabling: Enabling represents the interaction between two actions, where the execution of the first action allows the execution of the second that previously was not possible. This requirement is especially important in any case where the person is following a given goal and not just randomly executing actions, as well as in cases where there is an interaction between different users in a multi-agent setting. For example, we enter the room and want to sit down, but there is a book on the chair. We cannot sit down before the book is removed. We say that removing the book enables us to sit down. To express the requirement according to the CSP notation, we use the notion of traces as explained in Appendix B (page 204).

Definition 12. ***(Enabling)*** *Given a process X with an alphabet $\alpha X = \{x, y\}$, we say x enables y if the trace $\langle x, y \rangle$ is a valid sequence and the trace $\langle y, x \rangle$ is not a valid one.*

Disabling: Disabling represents the interaction between two actions, where the execution of the first action forbids the execution of the second. Additionally, given the first action was not executed, the second could still be executed. The requirement is the opposite of *enabling* and its existence is important in the same situations as with its counterpart. Using the example with the book, if we put a book on the chair, this will disable us from sitting down. To express the requirement according to the CSP notation, we use the notion of traces as explained in Appendix B (page 204).

Definition 13. ***(Disabling)*** *Given a process X with an alphabet $\alpha X = \{x, y, z\}$, we say that y disables z if the trace $\langle x, z \rangle$ is a valid sequence, but the trace $\langle x, y, z \rangle$ is not.*

Dependence: Dependence allows the execution of an action to be influenced by the effects caused by other actions. The requirement is extremely important in any situation where the actions are causally related or the user(s) is (are) following a common goal. It is strictly related to the requirements *enabling* and *disabling*. Basically, any action that either enables or disables other actions poses the property of dependence.

Definition 14. ***(Dependence)*** *Given a process X with an alphabet $\alpha X = \{x, y\}$, we say that x is dependent on y or vice versa, only if one of the two requirements enabling, or disabling holds for x and y.*

Interleaving: Interleaving allows expressing behaviours where actions from each of the behaviours are executed sequentially until both behaviours are completed. Such requirement is important in situations where a person cannot do two activities sequentially, but who is able to execute parts of the first activity in between parts of the second activity. For example, while we are cooking, the phone rings, so we seize our activity in order to answer the phone, then return to cooking. To express the requirement in terms of the CSP notation we represent the behaviour with a process and the actions it is composed of with events, then we use the notion of interleaving processes described in Appendix B (page 205).

Definition 15. ***(Interleaving)*** *Given two processes X and Y, we say that X interleaves Y, if events from X are executed in between events from Y. We denote this process with $X|||Y$.*

Priority: Priority represents the deterministic version of *choice*, or with other words the selection of an action from several actions based on some external factor known to the environment. Taking the example for the requirement of choice, if we beforehand know that we are the first to present, then we will prefer to go to the stage instead of sitting down. CSP defines such interaction as deterministic choice (see Appendix B, page 205) between two processes where the environment can control which of the two will be selected. This control is executed on the first action being executed and represents exactly the notion of priority.

Definition 16. ***(Priority)*** *Given two processes X and Y we say that one has priority over the other, if the executed process behaves either as X or as Y and if the environment has control over this choice. We denote priority with $X[]Y$.*

Independence: Independence allows the execution of an action not to be influenced by other actions. That means that given a set of actions, the independent action can be executed in any order and there is no action that can influence the execution of this action. Such requirement is important in cases where there are actions unrelated to the current goal, or when in multi-agent settings the agents are acting without interacting with each other. For example, within any scenario, we can stop for a moment and wait, and this action does not have any influence on the rest of the actions. To express this requirement in terms of CSP, we use the notion of traces (page 204).

Definition 17. ***(Independence)*** *Given a process X with an alphabet $\alpha X = \{x, y_1, ..., y_n\}$ where $n+1$ is the number of distinct events, we say that the event x is independent if it can be executed in any order $\langle x, y_1, ..., y_n \rangle$, $\langle y_1, x, ..., y_n \rangle$, ..., $\langle y_1, ..., y_n, x \rangle$. Furthermore, none of the y_i events has the requirement of dependence with respect to x.*

Synchronisation: Synchronisation allows setting the execution of several actions or processes in parallel only after a certain action is executed. The requirement is important for synchronised multi-agent behaviour where several users work in parallel toward achieving a common goal and where certain parallel actions can be executed only after a given activity was successfully executed. For example, in order for a presentation to start, each of the users have to sit down and prepare to listen. That means they have to synchronise their actions in order to start the presentation. To define this requirement in terms of CSP we use the above definition of parallelism (page 205) and the definition of composite sequential processes (page 206) explained in Appendix B.

Definition 18. ***(Synchronisation)*** *Given processes X, Y and Z, we say that X synchronises Y and Z, if $Y||Z$ and if X has to be executed before that in order for $(Y||Z)$ to take place. We denote this requirement as $X;(Y||Z)$.*

Suspending: Suspending allows interrupting a behaviour with another action or behaviour. The requirement is needed in multi-agent situations where the actions of one agent (or an external factor) can interrupt that of another agent. For example, while we are presenting, somebody is entering the room thus interrupting our presentation. To express suspending in terms of CSP we use the notion of interruption described in Appendix B (page 206).

Definition 19. ***(Suspending)*** *Given two processes X and Y, we say that X was suspended by Y, if while X was in progress, an event from Y was executed thus interrupting X. We denote this as $X^\wedge Y$.*

Resuming: Resuming allows the continuing of an action after it was previously suspended. The requirement is closely related to *suspending* and can be thought of as the second part of a process that first interrupts a behaviour by executing another behaviour and after the execution of the second behaviour, it resumes the first one from the state it was in before being interrupted. For example, after being interrupted, we later continue our presentation from the point where we were previously interrupted. To describe this interaction between processes, CSP uses the notion of alternation described in Appendix B (page 206).

Definition 20. ***(Resuming)*** *Given two processes X and Y, we say that X was resumed by Y, if $X^\wedge Y$ and if later X continued its execution from the state it was in before being suspended. We denote that as $X \otimes Y$.*

Composition: The last property is that describing composite behaviour, or behaviour built up of other behaviours. As such composite behaviour can consist not only of sequential composite processes that are described in Appendix B, but also of parallel or interleaving processes, we define the notion of composition based on that of parallelism (page 205), interleaving (page 205), and composite sequential processes (page 206).

Definition 21. ***(Composition)*** *Given processes X, Y, and Z, we say that X is a composition, if $X = (Y||Z)$, or $X = (Y;Z)$, or $X = (Y|||Z)$.*

2.4.2 Application-based requirements

After defining the behaviour-based requirements, below the application-based requirements are presented. To define them along with the CSP notation we introduce the notion of observation.

Definition 22. ***(Observation)*** *Given a process X with an alphabet $\alpha X = \{x_1, x_2, ..., x_m\}$, we call $O_X = \{o_{x_1}, o_{x_2}, ..., o_{x_m}\}$ the set of physical sightings that capture this process. Such sightings are produced by sensors that observe the environment.*

Additionally, we use the notion of probability and conditional probability as described in probability theory [6, p.1– 43].

Probabilistic durations: Probabilistic durations allow expressing actions' durations in terms of probabilistic distribution, i.e. describe what the probability is of the action continuing to be executed in the next time step.

Definition 23. ***(Probabilistic durations)*** *Given an event x and a time interval δ_t that will elapse between the begin and the end of x, we say that x has a probabilistic duration if it started at time t, and it is being executed until a time interval $\tilde{\delta_t} \sim p(\delta_t|x)$ has elapsed.*

Observation model: An observation model allows expressing the connection between behaviour and observations, i.e. what sensor reading is associated with which actions in the model.

Definition 24. ***(Observation model)*** *Given a process X with an alphabet αX, an event $x \in \alpha X$, and a set of observations $O_X = \{o_1, o_2, ..., o_n\}$, the observation model provides the probability of a certain observation o_i being true given x, or with other words $P(o_i|x)$.*

Unobserved actions: Unobserved actions allow modelling actions that were executed by the user but not observed by the sensors due to sensor granularity or unreliability.

Definition 25. ***(Unobserved actions)*** *Given a process X with an alphabet αX, an event $x \in \alpha X$, and a set of observations $O_X = \{o_1, o_2, ..., o_n\}$, we call an action x unobserved, if there exists no observation that gives the probability $P(o_i|x)$.*

Activity recognition: To perform activity recognition the model allows inferring activities based on a set of observations. With other words the model should be able to provide the probability that a certain activity was executed, given the observations.

Definition 26. ***(Activity recognition)*** *Given a process X with an alphabet αX, an event $x \in \alpha X$, and an observation o_t at time t, we say that the model performs activity recognition, if it provides the probability of $P(x|o_t)$.*

2.4.3 Do the requirements represent the actual need? - A study

The requirements were identified using a specific process and based on the designer's experience. Yet one obvious objection against their validity could be the possibility that they are just the designer's biased interpretation of what is needed for modelling the problems. Is it possible that the requirements in their current form are needed only by the designer herself? Would researchers from other fields of computer science understand the requirements and think them necessary at all?

To answer these questions, a questionnaire was distributed among 18 participants with different academic degrees[4] and from different fields of computer science[5] in order to identify problems in the requirements specification. To do that, they were asked questions about the requirements specification regarding the given problems; about a possible modelling formalism that could be a solution for the modelling problems; and finally for each requirement, they had to give an example from the problem domain in order for the evaluator to identify whether the participant understood the requirement at all.

To answer the questions above, several hypotheses were assumed. They were then investigated based on the answers provided by the study participants. To quantify the answers a Likert scale was used with five items from 1 to 5 where one is the lowest quantity and five the highest. These quantities are referred to as scores[6].

Hypothesis 1: *The majority of the requirements have a median[7] score of above 3.*

[4]The highest degree the participants have obtained was as follows: Abitur (2), Diplom (11), Master (4), and Doctor degree (1).

[5]The participants were from the fields of activity recognition (7), data analysis (2), electrical engineering (1), modelling and simulation (2), networking (2), software engineering (2), and visual computing (2).

[6]For more information about Likert scales see Appendix C.

[7]As explained in Appendix C, the Likert scale is assumed to be ordinal, thus we use the median and not the mean.

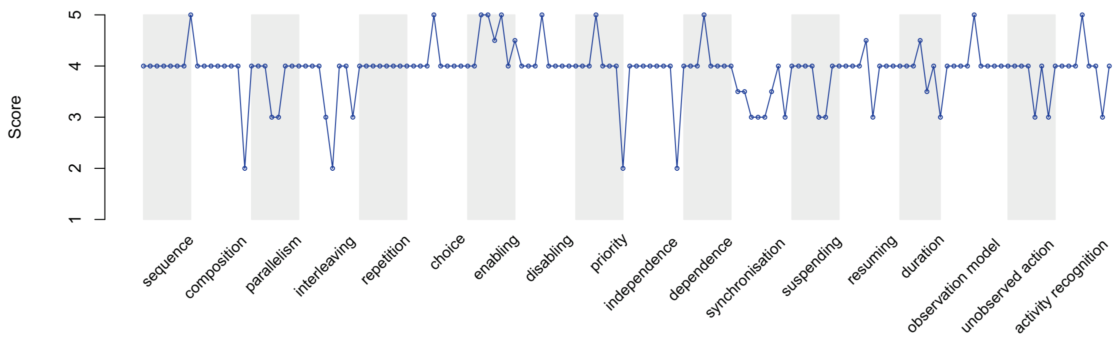


Figure 2.6: Average scores per requirement. For each requirement there are 8 features that are plotted, namely, verifiability, validity, clarity, completeness, feasibility, testability, traceability, and importance (for more information on the properties, see Chapter 5). The varying grey and white colours enclose the features for a given requirement. To calculate the average score for a given feature the median for all answers concerning this feature was taken.

This hypothesis aims at showing that the requirements identified by the model designer are understandable and clear for the majority of participants that took part in the study. Many of them were not closely related to the field of activity recognition, and the opposite could also be possible – the specifications could be not well understandable, misleading, or clashing with terms from the participants' filed of work. The results showed that the majority of the answers were of score 4 which is the second strongest answer possible. Figure 2.6 shows the scores per requirement, where the x-axis indicates the requirement with its different features, while the y-axis indicates the score it was assigned. This shows that the requirements specifications were acceptable for the participants.

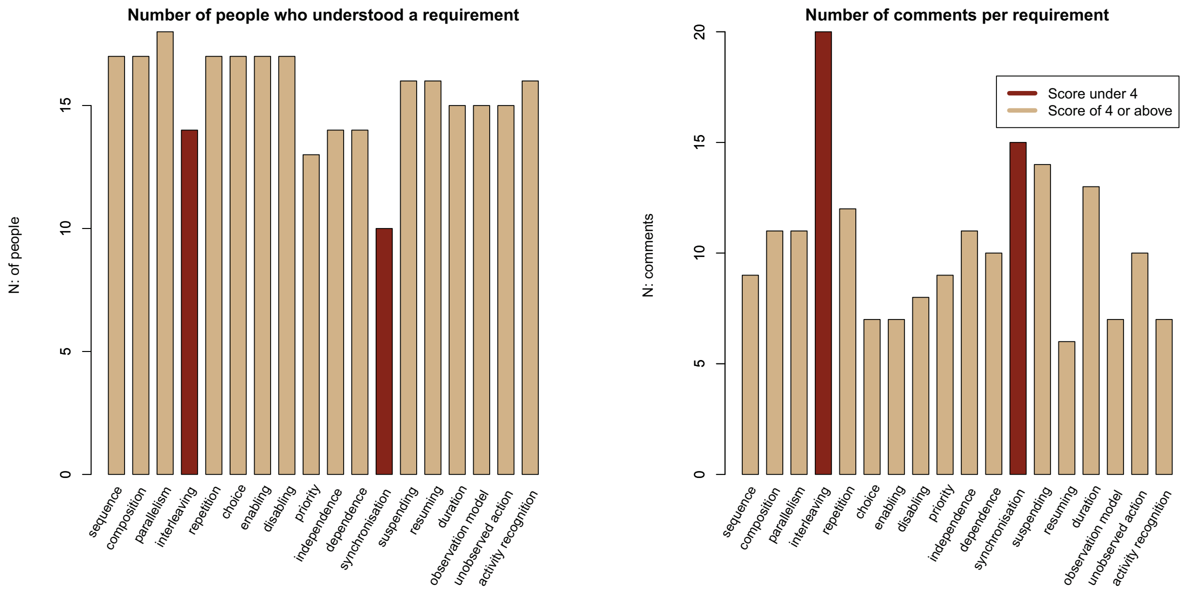


Figure 2.7: Number of people who understood a requirement and number of comments per requirement. The left figure indicates the number of people who understood the requirement where with beige, the requirements with score of 4 and above are shown, whereas with brown – those with score under 4. The plot to the right indicates the number of comments per requirement. It has the same colour scheme and legend.

Hypothesis 2: *The number of people who understood a requirement is higher compared to those who did not understand it.*

This hypothesis aims at showing that the requirements were clearly specified and most of the participants were able to understand their meaning and to give an adequate requirement example from one of the 3 problems. Fig. 2.7, left, shows the number of people who understand a requirement. It could be seen that there is no requirement that was not understood by at least half of the people (synchronisation having the lowest number of people that understood it). It can even be seen that with small exceptions, the requirements were understood by at least 15 out of the 18 participants, which stands to show that the hypothesis is true for this study. This indicates that the requirements the model designer identified are mostly meaningful for researchers from different fields of computer science. It can also be seen that the requirement *synchronisation* that was not well understood also got a lower score for its properties (with brown). This is however not the case with the requirement for *interleaving* that was understood by 14 out of the 18 participants, yet got a score under 4. This indicates the participants did not give their evaluation based on how well they understood a requirement, but rather tried to give an objective rating also of requirements that were well understood. Furthermore, Appendix C.3 contains some more requirements' evaluations[8] given by the participants. In them it can be seen that each requirement got an average score of 3 and above for its verifiability, validity, clarity, completeness, feasibility, testability, and traceability. These results further support the hypothesis that the requirements were generally understood and accepted as valid.

Hypothesis 3: *The requirement's score is inversely proportional to the number of comments for the given requirement.*

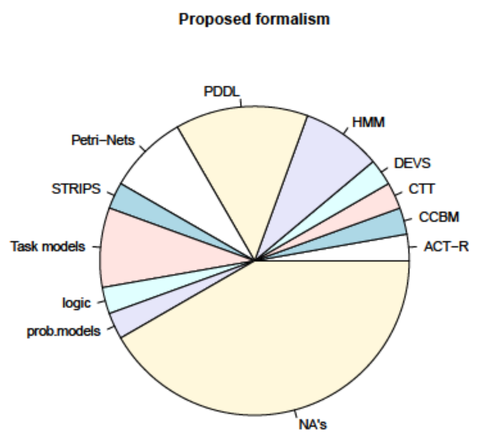
This hypothesis indicates that the more comments were given per requirement, the less clear it will be, thus the score will also be lower. Figure 2.7, right, shows that the assumption holds for the gathered data and the requirements with scores under 4 (interleaving and synchronisation) also got the most comments. It also shows that only two of the 19 requirements got average scores under 4 and also more than 15 comments per requirement, indicating that the requirements as a whole were well understood. Also the fact that there is no requirement without a comment suggests that the participants put effort in understanding and even proposing improvements for requirements that received high evaluation score.

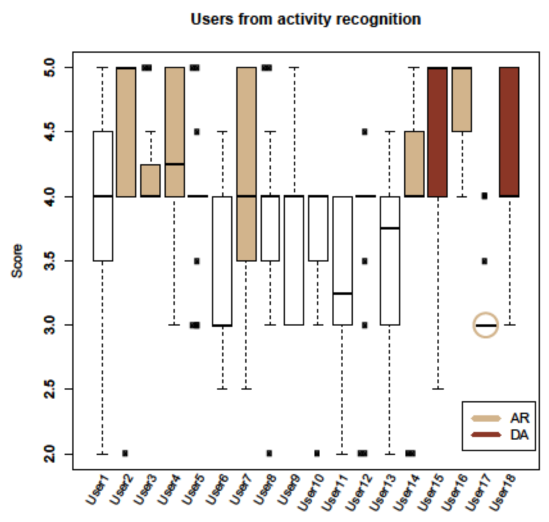
The next hypothesis aims at identifying whether the field of work influences the scoring.

Hypothesis 4: *The participants from the field of activity recognition and data analysis gave higher scores for the requirements' specifications.*

This hypothesis is based on the assumption that participants from the fields of activity recognition and data analysis will have better understanding of the concept of activity recognition and thus will more easily understand the requirements compared to those participants that come from other filed of computer science. Figure 2.8(b) shows that the hypothesis holds for all the participants except for *user 17* who had an average score of 3. However if we assume

[8]The evaluation was based on the properties a requirement has to possess according to the software requirements specification (SRS) semantic properties described in [39] and the model requirements properties defined in [11]. A detailed description of the properties is given in Chapter 5. Furthermore an excerpt of the questionnaire can be seen in Appendix C





(a) Modelling formalisms proposed by the participants (b) Average scores per user based on the user field of work

Figure 2.8: Proposed formalisms (a) and scores for users from activity recognition and data mining (b). In (b), the beige boxplots indicate participants from the field of activity recognition, while the brown ones – from the field of data mining. The white boxplots represent participants from other field of science.

that user 17 is an outlier[9], the hypothesis could be considered as valid. This stands to show that the requirements specification was well understood also by other scientists from the field of activity recognition and data mining, and did not reflect only the personal interpretation of the model designer. The fact that the requirements received better scores from people from these two fields indicates that the specifications are clearly defined for the future users of these requirements. These are exactly the designers of activity recognition systems. Based on this result, in the next section the requirements were prioritised by a group of activity recognition and data analysis experts.

From the above results, it can be concluded that the requirements for human behaviour modelling for the three problems are accepted as understandable and necessary also by other researchers from the field of activity recognition. Furthermore, participants from other fields of computer science found them generally clear, indicating that the requirements reflect the needs of the problems to be modelled even for persons who are not biased by previous experiences in the field of activity recognition.

2.4.4 Prioritisation

The results supporting *Hypothesis 4* from the previous section indicated that the researchers from the field of activity recognition and data analysis have a clear understanding of the requirements. For that reason, 8 of the participants in the study from these two fields were asked to discuss the requirements importance and priority. The reason for that is that when choosing

[9]This is suggested by the assumption that she is always choosing the middle ground. The assumption is made based on the fact that participant 17, who also claimed to have an expert knowledge, almost always gave the same score for every requirement. This could indicate that they did not really try to answer the questions, or that their actual knowledge was not sufficient for answering adequately.

an appropriate modelling formalism, if more than one formalisms cover the same number of requirements, the one with an overall higher requirements priority can be selected. The requirements' priorities can also be used during the model design phase when the impact on conflicting requirements to be implemented is taken into account. In order to be able to come to convergence about the requirements importance, their prioritisation was restricted to only three values: important, unimportant, and irrelevant.

Table 2.5: The table shows how the requirements are prioritised. The range of different priorities is limited to three values: important, unimportant, and irrelevant.

	Requirement	3-person meeting	cooking task	office scenario
Behaviour-based	sequences	important	important	important
	parallelism	important	unimportant	important
	composition	important	important	important
	interleaving	irrelevant	important	unimportant
	repetition	unimportant	important	unimportant
	choice	important	important	important
	enabling	important	important	important
	disabling	important	important	important
	priority	important	important	important
	independence	important	important	important
	dependence	important	important	important
	synchronisation	important	irrelevant	irrelevant
	suspending	unimportant	unimportant	irrelevant
	resuming	unimportant	unimportant	irrelevant
Application	prob. durations	important		
	observation model	important		
	unobserved actions	irrelevant		
	activity recognition	important		

Table 2.5 shows the requirements prioritisation. It can be seen that some of the requirements have different priority depending on the use case with suspending and resuming being either unimportant or irrelevant in each of the cases. This indicates that they can be mostly ignored for the solution of the problems. Furthermore, in the application-based requirements, the requirement for unobserved actions was deemed irrelevant, with the suggestion that such requirement could be handled by the observation model instead of the human behaviour model. For that reason, the latter will not be discussed further in this thesis.

Based on the requirements and their importance, in the next section several candidate modelling formalisms are discussed.

2.5 Human behaviour models for activity recognition

To implement a solution for the three modelling problems, first a suitable modelling formalism has to be selected. The previous section already discussed the requirements such formalism should possess in order to be successful. Here, based on these requirements and their importance, several candidate formalisms are discussed and finally the one satisfying the most requirements is selected. Additionally, the discussion is restricted to formalisms that are able to incorporate prior knowledge in order to support the decision making of the corresponding activity recognition system.

Fig. 2.8(a) shows candidate formalisms proposed by the participants of the evaluation study. It can be seen that many did not give any suggestions. On the other hand, those who proposed solution formalisms gave as examples several rule-based approaches, some probabilistic approaches and some calculus approaches like Petri Nets and DEVS that are usually used for testing the capabilities of a system. As we are interested in approaches for activity recognition that are able to incorporate prior knowledge from which probabilistic models can be generated, we consider the rule-based approaches ACT-R, CCBM, and PDDL. Additionally, concurrent task trees are considered together with CTML task models as it was already shown that one can generate probabilistic models based on those behaviour models [58, 158]. Furthermore, we regard some additional formalisms as suitable. The study participants might not be aware of them, but it has already been shown that they are applicable in the field of activity recognition and mostly comply with the requirements for context-aware systems described in Chapter 1.

2.5.1 Some formalisms that are not regarded

There are formalisms that may seem suitable for the three modelling problems, but that are not considered in this work for a variety of reasons. The first big group of such formalisms are the ontologies used for activity recognition [104, 119]. Although vastly applied to this field of science, they are not regarded here because they do not represent causal relations between actions, but rather hierarchical dependencies between classes. This, although giving a mechanism for correctly classifying the activities, could pose a problem when trying to reason about actions' histories and their causes.

This leads to the second group of formalisms that are not considered, and where ontologies also fall – these are approaches that assign just action labels but do not make use of semantic goals. Such formalisms once again have the drawback that they are not able to reason about the causes behind actions. They are also not able to provide rich context information related to the action's specific causes.

Finally, approaches that support semantic goals but were never applied to activity recognition, or which were never extended for this application, are also not regarded. Typical example of such are planning formalisms that were never applied to activity recognition (e.g. Hierarchical Task Networks with Partial Order Planning [123, p. 406–415]). The reason for that is to find a formalism that can be directly applied or needs minimum modifications in order to be able to support activity recognition. This is to reduce the designer effort in producing a successful human behaviour model for activity recognition. One could argue that many formalisms can produce promising results, provided they are extended for the purpose of activity recognition. However, the goal of this work is not to provide a new formalism for activity recognition, but rather to show that models supporting semantic goals can be successfully applied to activity recognition problems, and based on that to introduce a development process for such models for activity recognition.

2.5.2 Adaptive Character of Thought - Rational

The Adaptive Character of Thought -Rational or ACT-R is a cognitive architecture dealing with the process-level theory about human cognition. It has its roots in cognitive science and is concerned with explaining how the human mind works, and how humans think, perceive and act [2, 4]. ACT-R is a symbolic/subsymbolic production system and assumes that knowledge forms the basics of cognition and that there are two kinds of knowledge – declarative (facts), and procedural (skills and rules). Additionally, there is also the input from the outside world that can be in the form of visual, aural etc. perception. The declarative knowledge is static and is stored in the form of chunks which is one of the basic ACT-R elements and can be thought of as pieces of static memory that we can retrieve from our knowledge base. The procedural knowledge, on the other hand, is a set of production rules that explain how to use the declarative knowledge and how to react in a given situation.

ACT-R is composed of different elements – it has several different limited-capacity buffers which together build up its context. The buffers are the goal buffer, the retrieval buffer, the visual buffer, and the manual buffer. Each buffer is supported by one or more theoretically motivated modules. Each of the modules represents a specific cognitive area and has been shown to correspond to similar anatomical faculties in the brain [3]. The first module is the declarative which is responsible for the storage and management of the factual knowledge, also known as chunks. It is also responsible for the chunks' activation values which are basically functions of how recently and frequently a given chunk was retrieved. The second module is the procedural which is similar to the declarative, only it stores the procedural knowledge, or productions. The subsymbolic information about the productions is represented by an expected utility, which is learned over time based on a temporally discounted reinforcement learning function [50]. The procedural module provides algorithms for matching the contents of the buffers to production rules where the best match is selected to fire (namely to be executed) and later it handles the implementing of the actions' results. The intentional and imaginal modules cope with the task-oriented cognition. The goal buffer, associated with the intentional module is responsible for identifying the model's current goal, while the imaginal buffer provides support for intermediate state representations. The visual module is responsible for the system to see elements in the outside worlds, while the aural module recognises sounds in the environment. The temporal module, on the other hand, is responsible for keeping track of the elapsed time. Finally, the manual and speech modules are the model's actuators, namely they are responsible for the physical and audio interaction with the outside world.

Each of the modules described above is involved in complex interactions which compose the ACT-R's predictive power and the ability to explain the human thought processes. On the other hand, the modelling of the perceptions and actions is limited to how the cognition utilises them, thus some processes such as the actual execution of actions, are outside the architecture's scope.

2.5.2.1 Applications

ACT-R has variety of applications – from simulating human cognition in order to give better understanding, to providing user assistance based on those cognitive models.

For example, in his work [75], Juergen Kiefer models individual human behaviour in human multitasking by using ACT-R. He investigates individual cognitive strategies in dynamic multitasking environments and the resulting theoretical consequences for modelling. He achieves that by using a car driving simulator where the test participants executed a compound continuous task. The test results showed that under multitasking cognitive strategies are used to optimally adapt to a given situation, thus the strategies were successfully transferred into ACT-R and their usage was able to explain individual differences in dynamic task environment.

Another work based on ACT-R investigates the modelling of the progression of Alzheimer's disease with application in smart homes [128]. The authors present a way of modelling and simulating the progression of dementia and also evaluate the performance of executing an activity of daily living. In difference with other works form this area of research, the paper focuses on modelling and simulating erroneous behaviour and its progression parallel to the disease progression, rather than the modelling of normal behaviour. The simulated behaviour of 100 people suffering from Alzheimer's disease was compared with the results of 106 patients performing an occupational assessment. The comparison showed that the modelled behaviour closely resembles the behaviour of real patients and the authors concluded that the model is able to capture not only the erroneous behaviour but also its progression in the different phases of the disease.

In the field of activity recognition and assistance, ACT-R was successfully used to model a robot's understanding about human actions in a human-robot team scenario [66]. In it, a model of the available human actions is created and later different simulations run, each of which has different initial state and prior knowledge. Additionally, all possible execution sequences in a model are followed and the probability for these sequences is calculated. In that manner, when the robot observes its teammate executing a given action, the robot is able to reason about the cause of it and give advice, or if needed, explain to the human that she has made a mistake. This work is extended in [146] where the authors attempt to give the robots a deeper understanding of human cognition and fallibilities by applying cognitive models to tasks like gaze following, hide and seek, interruption and resumption.

2.5.2.2 Requirements fulfilment

In the previous section, the requirements a formalism should satisfy were discussed. Here we analyse which of them ACT-R satisfies.

Composition: ACT-R is able to express composition by defining actions in the form of IF-THEN clauses which contain causal relations between different actions.

Sequence: Basically, every formalism should be able to express sequential actions. ACT-R is no different. Actions executed in sequence can be defined using productions, in the following way.

Production 1: `IF the goal is to execute A, and B was not executed,`
`THEN execute A.`

Production 2: `IF the goal is to execute B, and A was executed,`
`THEN execute B.`

Parallelism, Synchronisation, Suspending and Resuming: Executing parallel actions or modelling users that act in parallel can be achieved by creating separate models for each agent, and then running the models simultaneously. The synchronisation between the different models is then done by a separate model that manages the interactions between the agents, similar to the communication model proposed in [95]. The same applies for suspending and resuming a composite action – this can be done by the execution of another action from the second agent which effect is communicated to the first agent via the communication model, thus interrupting the action the first agent is conducting.

Repetition: ACT-R does not have an explicit mechanism for modelling repeating actions. However, using the production rules, one can easily implement such. Additionally, a counting mechanism can be implemented in order to keep track of the number of times the action was executed.

Production 1: `IF the goal is to execute A, and A was executed, and the counter is less than MAX_COUNT,`
`THEN execute A; increase counter.`

Choice and Priority: The choice is modelled when two or more productions have the same IF clause. Then, which action is executed can be controlled by the different types of heuristics ACT-R provides. These are

Salience: An operator may be prioritised by applying a weight to it.

Recency: The most recent operator may be prioritised.

Refractoriness: An operator that was applied once should not be applied again. This strategy helps to avoid creating infinite loops.

Specificity: The operator that fulfils the most predicates of the goal state is preferred to other.

Dependence, Enabling and Disabling: Dependence in its two forms – enabling and disabling, can be modelled by using the same production mechanism. Below *Production 1* shows an example of enabling, and *Production 2* shows an example of disabling.

Production 1: `IF the goal is to execute A, and B cannot be executed,`
`THEN execute A; B can be executed.`

Production 2: `IF the goal is to execute A, and B can be executed,`
`THEN execute A; B cannot be executed.`

Interleaving: There is no explicit mechanism for modelling interleaving actions. However, they can be modelled by allowing two composite actions to be executed in an interleaving way.

Production 1: `IF the goal is to execute A and B,`
`THEN execute A1; execute B1; execute A2; execute B2.`

Independence: Independence can be easily achieved by just defining IF clause that does not depend on any other action.

Production 1: `IF the goal is to execute A,`
`THEN execute A.`

Application-based requirements: In its standard form ACT-R is not able to cope with *probabilistic durations*. Regarding an *observation model*, the standard ACT-R does not possess such. Still it was shown in [66] that it is possible to reason about the user actions' based on observations perceived by ACT-R's perception modules (like visual and audio module). Hiatt et al. [66] also showed that it is possible to apply ACT-R for *activity recognition*.

2.5.3 ConcurTaskTree

CTT which stands for ConcurTaskTree is a notation that was first introduced by Paterno [108] and which provides support for design and analysis of complex task models in multi-user environments. With it a compound activity is represented as a task tree, where each tree node represents a task which allows composite tasks to be decomposed into subtasks. Various temporal operators are used for expressing the relations between the tasks in the tree. Each task is associated with a specific type, a category, attributes, and objects it is able to manipulate. Additionally, it provides a graphical syntax that allows easier interpretation of the logical structure of a task.

Fig. 2.9 shows an example of a task model, where a simple composite task consisting of four tasks (A, B, C, and D) is represented in a CTT notation. The tasks A, B, and C can be executed in any order, which is specified by the temporal relation *order independency* ($| = |$). Additionally, task D can be performed only after all the other tasks are executed, which is specified by the relation *enable* ($>>$).

In that manner user behaviour and the interaction between different users can be expressed in a tree-like manner.

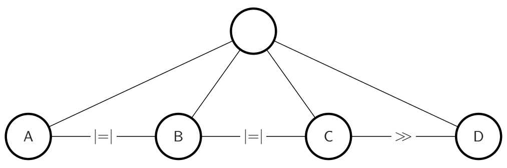


Figure 2.9: CTT for a composite task with four tasks (A, B, C, D) (Figure adapted from [58]).

2.5.3.1 Applications

CTT is mostly used in human-computer interaction problems such as building successful interface designs. For example, Li at al. [89] use CTT to generate interface model of a display/-control system. Furthermore, Klug et al. [77] extend CTT to accommodate its execution during runtime, allowing the generation of applications that adapt to the user actions and preferences.

It has also been shown that CTT can be applied in the field of activity recognition. In their work [58], Giersich et al. use CTT to model tasks from the viewpoint of mobile and ubiquitous computing. With the help of CTT they manage to derive the dialog structure of a mobile human computer interface and then use probabilistic behaviour models to assign probability distribution over the activities space in order to infer the activity of a user. More concretely, they propose the usage of priority values assigned to each sibling in a node that are relative to the priority of all the remaining siblings. Then based on the model and the priorities, the probability of the transitions from the given state to the next is calculated. This is done based on the model history allowing for probabilistic reasoning over the user actions.

2.5.3.2 Requirements fulfilment

Below the requirements that are satisfied are discussed and the manner in which they are implemented.

Composition: CTT expresses composition in the form of a hierarchical structure where each root task has as leaves the actions, or tasks, it is composed of. Fig. 2.9 shows an example of such task where the composed action consists of the four actions A, B, C, and D.

Sequence: Sequences are represented by sibling nodes in a task tree with a relationship operator assigned between them. The sequential actions can have different relations (e.g. enabling, disabling, order independence, etc.). In the example from Fig. 2.9, the actions A, B, and C are executed sequentially and have order independence, while action D is sequential to the last executed action and has the relation enabling.

Parallelism: Parallelism is achieved by the *concurrency* relation between two nodes ($A|||B$), or by using the concurrency and information exchange operator ($A|[]|B$) where the nodes can also exchange information while acting in parallel.

Repetition: CTT expresses an action repetition by simple assigning an asterisk sign to the repeated action ($A*$).

Choice and Priority: Choice in CTT is modelled by using the temporal operator for choice ($A[]B$). This indicates that both actions are executable, but when one of them is executed, the remaining one can no longer be executed. Priority in CTT is managed by the temporal operators thus, it is not available in the standard CTT formalism. However, Giersich et al. [58] extended the notation to use priority values based on which later the transition probability distribution was calculated.

Dependence, Enabling and Disabling: Enabling in CTT is handled by the enabling temporal operator ($A >> B$), which indicates that B cannot start before A was executed. It is also possible to use enabling with information passing with the temporal operator ($A[] >> B$) Similarly, disabling is modelled by the corresponding operator ($A[> B$) which indicates that A is disabled by B.

Interleaving: Interleaving in the sense described in the previous section cannot be modelled in CTT as composite actions have to execute the actions of which they are composed before another composite action is able to be executed. However, it is possible to use suspending and resuming of composite actions to achieve the effect of interleaving.

Suspend and Resume: Suspend and resume are modelled by the *suspend/resume* temporal operator ($A| > B$), which indicates that A can be interrupted by B, and later when B is executed, A can be resumed.

Synchronisation: In CTT synchronisation can be achieved by having an action enable the execution of two concurrent actions.

Independence: Independence is modelled by the *order independence* operator ($A| = |B$) which indicates that the actions can be executed in any order, but when one of them starts, it had to be finished before the second can start.

Application-based requirements: The standard CTT notation does not support *probabilistic durations*. The same applies for *observation model*. However, Giersich et al. [58] have shown that it is possible to extend the model so that it can be used for generating probabilistic models that support probabilistic durations and observations. Giersich et al. have also shown that it is possible to apply CTT in its extended form to *activity recognition* problems.

2.5.4 Collaborative Task Modelling Language

The Collaborative Task Modelling Language or CTML is proposed by Maik Wurdel [157, 159] and is used as a specification framework for collaborative applications. It is designed specifically for the needs of intelligent environments and satisfies the following requirements for such collaborative applications: it has task driven methodology; it is able to model cooperation; it is able to model the domain; and it has formal syntax and semantics. These features make CTML a good solution for activities modelling especially when team cooperation is considered. A CTML model is a tuple consisting of a set of actors, a set of roles, a set of collaborative task expressions and a set of domain objects defined by a domain model. A collaborative task expression is just another variation of task trees and is modelled as a CTT-like tree that has an identifier, precondition and effect.

2.5.4.1 Applications

In their work [159], Wurdel et al. give examples of CTML's usage in a collaborative environment. They apply the modelling formalism to a simple meeting situation consisting of a chairman, presenter and an audience. The chairman announces the talk topic, and while the presenter presents it, the audience can access additional information concerning the presentation topic on their personal devices. Subsequent talks are given in the same manner, until at the end the chairman encourages an open discussion, sums the session up and closes it. Using the CTML editor they specify this scenario and show the language's usability.

Additionally, Wurdel et al. [158] show that CTML, or its task models respectively, can be used in the field of activity recognition. The specified models are used to define the probabilities of the next possible action during activity execution. A probabilistic inference mechanism, having recognised the current state, then makes use of the task model in order to adjust the probabilities of the next state. This approach also reduces the state space growth as it removes all states that are unreachable form the current state.

2.5.4.2 Requirements fulfilment

Behaviour-based requirements: As CTML uses task trees to express the user behaviour, it could be safely said that the requirements CTML covers are the same that CTT supports. The difference between the two environments is that they have different tool support and that CTML provides additional features for smart environments. Still, these features are not applied to activity recognition problems.

Application-based requirements: Similarly to CTT, task models with CTML do not support *probabilistic durations* and *observation model*. They could, however, be extended to support probabilistic reasoning by transforming them into Hidden Markov Models [159]. For that reason they are suitable for applying to *activity recognition* problems.

2.5.5 Planning Domain Definition Language

The Planning Domain Definition Language or PDDL is initially developed for solving planning problems in the International Planning Competition [56, 55]. It has a STRIPS[10]-like syntax and expresses the actions in precondition-effect pairs that contain causal relations between the different actions. When expanded, they build an acyclic graph from the problem initial state to its goal state. Later a planner explores that graph with a suitable search algorithm and provides as an output a plan that is a possible solution to the problem.

PDDL is able to express the elements and dynamics of a domain, namely what kind of predicates are there, what set of actions are possible, what is the structure of compound actions, and what are the effects of these actions. The language supports the basic STRIPS-like actions, and in addition it has conditional effects, universal quantification over dynamic universes, domain axioms over stratified theories, specification of safety constraints, specification of hierarchical actions composed of subactions and subgoals, and management of multiple problems in multiple domains [56].

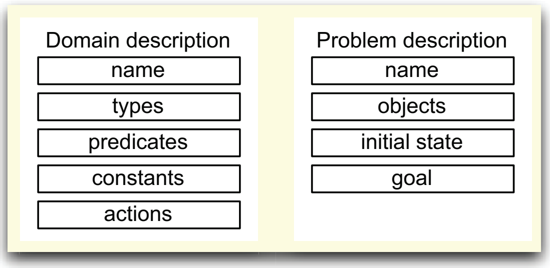


Figure 2.10: A model structure with the PDDL formalism.

A PDDL model is divided into two parts (see Fig. 2.10) – the first is the domain description that contains the action templates, the object types and description of the predicates used in the action templates. The action templates in turn are described by a name, parameters, duration, preconditions, and effects. The second part is the problem description that expresses the initial

[10]The **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver (STRIPS) is an automated planner first introduced by Fikes and Nilsson in 1971 [40]. It is the basis for many planning languages today.

world state, the constants used in the problem, and the goal of the problem. That way the first part provides an abstract description of the modelled domain, while the second gives the problem-specific details needed for solving the problem. This division allows creating abstract models that are later populated with the parameters of the specific problem, thus requiring the change only of the problem description when new problem from the same domain is present.

2.5.5.1 Applications

Although PDDL is designed for planning problems, it has been shown that it can also be applied to activity recognition problems. Burghardt et al. [18, 19] use PDDL for synthesising probabilistic models for activity recognition, where the actions are represented by preconditions and effects which allows the generation of different possible behaviours without explicitly specifying every one of them. This is done by extending the PDDL operators and generating a graph that contains all execution sequences leading from the initial to the goal state. The model is then used as part of an inference mechanism for adjusting the probabilities for the next possible action. It is also used for reducing the search space growth, as only states that are part of valid plans are considered.

Another work that applies PDDL to a plan recognition problem is that by Ramirez and Geffner [114]. In it they recognise the intentions of an agent that has an action library modelled in a PDDL-like notation. The model is shared by the observer and the agent but the actions of the agent are only partially observed. Then the policy for selecting the agent's action is based on the reward the agent will receive – thus, higher reward indicates higher probability for selecting the action.

2.5.5.2 Requirements fulfilment

As PDDL is a causal approach, it satisfies a set of requirements similar to that ACT-R satisfies.

Composition: Like ACT-R, PDDL can express composite actions by defining causal relations between actions. This is done by modelling abstract action operators that have the form of precondition-effect pair.

Sequence: Sequences are modelled in a similar to ACT-R way.

Action 1: `Precondition: A is not executed, and B is not executed, and A can be executed.`
`Effect: A is executed.`

Action 2: `Precondition: A is executed, and B can be executed.`
`Effect: B is executed.`

Parallelism, Synchronisation, Suspending and Resuming: Whether actions are executed in parallel in PDDL depend on the planner – if it is able to execute partially ordered plans, then the actions can be executed in parallel. That means, if the preconditions for both actions are satisfied and the planner allows them to be executed in parallel. For example, in their approach Burghardt et al. [18] are able to execute multiple actions in parallel. The same applied for actions synchronisation where in the case the planner allows concurrent actions, an action is executed that synchronises two other actions that are then executed in parallel. Suspend and resume are only applicable in the cases where composite actions are modelled, that can then be interrupted by another action, or by another agent.

Repetition: Like ACT-R, PDDL does not have an explicit mechanism for modelling repeating actions. Still, using appropriate predicates in the action description, allows an action to be repeated.

Action 1: `Precondition: A can be executed,`
`Effect: A is executed, and A can be executed.`

Choice and Priority: The choice is modelled when the preconditions of two or more actions are satisfied but parallel execution is not possible. Then, which action is executed can be controlled by the different types of heuristics. For example, it could be the goal distance [114], or the ACT-R heuristics saliency, refractoriness, recency, and specificity [79].

Dependence, Enabling and Disabling: Dependence can be modelled by using the same mechanism as in ACT-R. Below *Action 1* shows an example of enabling, and *Action 2* shows an example of disabling.

Action 1: `Precondition: A can be executed, and B cannot be executed.`
`Effect: A is executed, and B can be executed.`

Action 2: `Precondition: A can be executed, and B can be executed.`
`Effect: A is executed, and B cannot be executed.`

Interleaving: There is no explicit mechanism for modelling interleaving actions. However, they can be modelled by allowing two composite actions to be executed in an interleaving way. In other words, if we have a composite action $A = \{A_1, A_2\}$ and $B = \{B_1, B_2\}$, then the effects of A_1 will fulfil the preconditions of B_1, the effects of B_1 will fulfil the preconditions of A_2 and so forth.

Independence: Independence can be easily achieved by defining action's preconditions that are not dependent on the effects of any other actions. For example, if an action is executable, and there is no other applicable action able to change the fact that the action is executable, then this action is independent.

Production 1: `Precondition: A is executable.`[11]
`Effect: A is executed.`

Application-based requirements: The standard PDDL notation is able to express durative actions [134], and Krüger et al. have shown that it is possible to model *probabilistic durations* [79]. The standard PDDL does not support *observation models* but it was shown that the notation can be extended to support such [114, 18]. It was also experimentally shown that PDDL can be applied to *activity and intention recognition* problems [114, 18].

2.5.6 Asbru

Asbru is a time-oriented machine readable language developed for implementing skeletal plans in the Asgard project [74, 97]. The idea behind Asbru is to represent the domain knowledge as a library of skeletal plans that have various levels of detail and capture the structure of the modelled procedures but that allow parameterisation with different problem-specific elements. The created plans are stored in a plan library where each plan consists of a set of sub-plans that are necessary for successfully completing the plan objective. A plan that cannot be decomposed in a more fine-grained plan is then called an *action*. The plan interpreter when

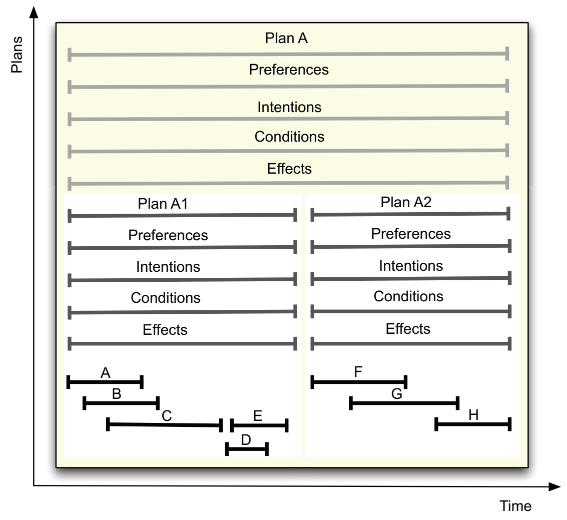


Figure 2.11: A plan structure in the Asbru formalism (Figure adapted from Miksch et al. [97]).

fed with a general plan is then attempting to decompose it into sub-plan until the level of the actions is reached. This plan consisting only of actions is then executed by the agent.

The structure of an Asbru plan can be seen in Fig. 2.11. It consists of a name, a set of arguments, including a time annotation, plan preferences, intentions, conditions, effects, and a plan body which describes the actions to be executed. A sub-plan of a plan then has the same components. It can be seen that the sub-plans are composed of actions (from A to H) which indicates that they cannot be decomposed any further.

2.5.6.1 Applications

Asbru is designed for the project Asgard that aims at supporting clinical guidelines by producing a library of skeletal plans that can later be applied to specific situations [130]. The project focuses on recognising the caretaker's intentions from their actions, and providing critique of these actions given the guidelines and the patient's medical record.

Another application of Asbru was proposed by Azam et al. [7] where they use skeletal plans for inferring user plans based on recognised actions. To achieve that, wireless proximity data is recorded and separated into tasks and subtasks using a task separator algorithm. The detected tasks then are mapped to the high level Asbru plans and together they are fed to an activity recogniser. The recogniser in turn matches the available wireless proximity data to that available in the plan library tasks, and when such are recognised, it attempts to infer the user plan.

2.5.6.2 Requirements fulfilment

Asbru consists of a library of temporally related plans and actions and can thus express many of the requirements with the help of these temporal relations.

Composition: In Asbru a composite action is represented by a plan that either has other plans in its body or consists of actions. Fig. 2.11 shows such plan structure where each plan is considered to be equivalent to a composite action.

Sequence, Parallelism, and Synchronisation: Sequences are modelled by operators in the plan body that indicate whether a plan is executed sequentially, or in parallel. The synchronisa-

[11] Here we assume that no other action has as effect *A is not executable*.

tion of two actions is also done with these operators, as the parallel operator expects the actions to start at the same time. Below *Plan 1* represents executing two actions sequentially, and *Plan 2*, the execution of the same actions in parallel that are also synchronised.

Plan 1: `(DO-ALL-SEQUENTIALLY (action A) (action B))`

Plan 2: `(DO-ALL-TOGETHER (action A) (action B))`

Suspending and Resuming: Suspending and resuming an action is modelled by defining a suspend or resume point in the plan. It is expressed in the time annotation clause by defining a time range in which the requirement should be executed, and the requirement itself together with a pointer at the action of a plan where this should happen. *Time annotation 1* gives an example of a suspended plan, and *Time annotation 2* – of such that is resumed.

Time annotation 1: `((<time-range>) SUSPENDED (action A))`

Time annotation 2: `((<time-range>) RESTARTED (action A))`

Repetition: Repetition of an action or a set of actions can be expressed by defining a cyclical plan. That is done with the *every* clause that is defined in the plan's body.

Plan 1: `(EVERY (START <start-time>) (END <end-time>) (do action A) END-EVERY)`

Choice and Priority: The choice is modelled by an operator in the plan body that allows executing the actions in any order. Priority, on the other hand is not modelled as it is handled by the time constraints and the sequential actions ordering.

Plan 1: `(DO-ALL-ANY-ORDER (action A) (action B))`

Dependence, Enabling and Disabling: Dependence can be modelled by using the time annotation which allows enabling a given plan or an action. Unless the plan has been already enabled, it is otherwise disabled, so there is no explicit definition of disabling.

Time annotation 1: `((<time-range>) ACTIVATED (action A))`

Interleaving: Asbru does not support interleaving actions, as in order to continue from one composite action (or plan) to another, all or part of the actions in the first have to executed, but there is no explicit way of forcing the model to execute the remaining non executed actions, after the second composite action is completed.

Independence: There is no mechanism for modelling independent actions as all specified actions have temporal dependencies.

Application-based requirements: Asbru is not able to model *probabilistic durations* in terms of probability distribution, but it can express uncertainty in the begin and end times of the action by defining shift periods. It does not support *observation model*, but Azam et al. [7] have shown that it is possible to map the actions to corresponding sensor readings. The standard Asbru language is used for plans generation, but Azam et al. [7] have shown that it can also be applied to *intention recognition*.

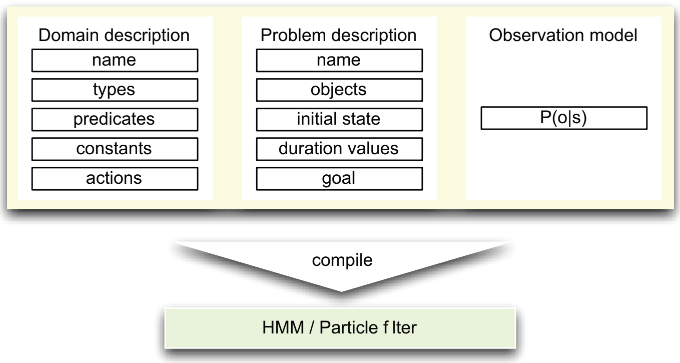


Figure 2.12: A model structure in the CCBM formalism.

2.5.7 Computational Causal Behaviour Models

CCBM are specifically designed for the purposes of activity and intention recognition [76]. The formalism combines causal models with probabilistic reasoning in order to be able to cope with the observations uncertainty. A CCBM model consists of several parts – a causal model divided in domain description and problem description, and an observation model describing the relations between the observations and the states in the causal model. A compiled model then produces a probabilistic model such as an HMM or a particle filter.

The causal model has a PDDL-like notation with several extensions providing the ability to specify different number of agents that can act in parallel, various types of duration probability distributions and different heuristics for action selection. Furthermore, an action is specified through its name, parameters, agents, duration, preconditions, effects, and observations. The observation model, on the other hand contains the information about which sensor readings and in what range are mapped to which high level actions and states. The structure and elements of a CCBM model can be seen in Fig. 2.12.

2.5.7.1 Applications

As already mentioned, CCBM and the corresponding tool, were specifically designed for activity recognition applications. Krüger et al. [80] showed that the modelling approach is suitable for problems from the meeting domain. They modelled the activities performed during a meeting in a smart environment and showed that the approach is suitable for such domains. It was tested on 21 activity datasets containing variations of a 3-person meeting and compared the results from those of a hand-crafted HMM. The comparison showed that the CCBM models are performing comparably to a hand-crafted model.

2.5.7.2 Requirements fulfilment

The causal model in CCBM uses a PDDL-like notation, thus many of the requirements are modelled in the same way.

Composition, Sequences, Synchronisation, Suspending, Resuming, Repetition, Choice, Dependence, Interleaving, and Dependence: All these requirements are modelled in the same fashion as in PDDL.

Parallelism: Parallelism is modelled by the additional *:agent* slot in the actions description. This slot is optional and when it is included in the action, it indicates that all available agents or objects of the given type are able to execute the action in parallel.

Priority: Priority can be explicitly modelled with the *:saliency* slot in the actions description. It gives then the action's priority relative to the priorities of the remaining actions. The priority is also modelled by the action selection heuristics – these are the goal distance, where the nearer to the goal the action is, the higher weight it has. Or it can be modelled by the ACT-R heuristics for action selection – recency, refractoriness, and specificity, which values are defined in the filter options.

Application-based requirements: CCBM is able to represent *probabilistic durations* with distributions such as exponential distribution, and normal distribution [76]. It also supports the modelling of an *observation model* that is separate from the observation model. The formalism is specifically designed for *activity recognition*.

2.6 Discussion – choosing a suitable modelling formalism

The above section presented the candidate formalisms for modelling the three problems. It also showed which requirements were satisfied and how (an overview of the satisfied requirements can be seen in Table 2.6). Now the only remaining question is how to select the most appropriate modelling formalism? To answer this question, here we propose a selection method based on the requirements satisfied by the formalisms and the importance of these requirements for the different modelling problems based on Table 2.5.

Table 2.6: The table shows the candidate human behaviour models that can be used for modelling the three problems. *yes* indicates the requirement is satisfied by the modelling formalism, *no* – that it is not.

	Features	ACT-R	CTT	CTML	PDDL	Asbru	CCBM
Behaviour-based	sequences	yes	yes	yes	yes	yes	yes
	parallelism	yes	yes	yes	yes	yes	yes
	composition	yes	yes	yes	yes	yes	yes
	interleaving	yes	yes	yes	yes	no	yes
	repetition	yes	yes	yes	yes	yes	yes
	choice	yes	yes	yes	yes	yes	yes
	enabling	yes	yes	yes	yes	yes	yes
	disabling	yes	yes	yes	yes	yes	yes
	priority	yes	yes	yes	yes	yes	yes
	independence	yes	yes	yes	yes	no	yes
	dependence	yes	yes	yes	yes	yes	yes
	synchronisation	yes	yes	yes	yes	yes	yes
	suspending	yes	yes	yes	yes	yes	yes
	resuming	yes	yes	yes	yes	yes	yes
Applic.	prob. durations	no	yes	yes	yes	no	yes
	observation model	yes	yes	yes	yes	no	yes
	activity recognition	yes	yes	yes	yes	yes	yes

Table 2.6 indicates whether a requirement was met by the formalism. As there is no concrete mechanism for selecting a modelling formalism in the field of activity recognition, here we propose the following criteria: the formalism that satisfies the most requirements is selected, where each requirement is weighted according to its importance.

$$\arg\max_{v_{max}} f(v_{max}) := \{v_{max} | \forall v : f(v) \leq f(v_{max})\} \tag{2.1}$$

Formula 2.1 indicates that the set of values of v where $f(v)$ attains its largest value is chosen. Here v is the value assigned to the formalism and $f(v)$ is calculated according to Formula 2.2.

$$f(v) := \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} r_j w_{ij} + \sum_{k=1}^{p} a_k w_k}{nm+p}, \tag{2.2}$$

where i indicates the index for the modelling problem, j the index of the behaviour-based requirement, and r is a behaviour-based requirement that has a value of either 1 if the requirement is met, or 0 if it is not met. The requirement's value is then multiplied by the requirement's importance w which is obtained from Table 2.5 and where irrelevant requirements are assigned a value of 0 so that they have no influence on the formalism selection, whereas important requirements are assigned a value of 1which indicates that important requirements that are satisfied produce a value of 1. In between these two values are the nice to have but unimportant requirements, which are assigned a value of 0.5 so that they would have some influence on the formalism selection but not as much as the important requirements. The weight of the application-based requirements a is calculated in a similar manner, where k is the index of the requirement. In this case however, they are not summed up over the three problems as they are all the same for the three cases. To normalise the result so that it is in the range $[0,1]$, it is then divided to the number of considered requirements multiplied by the number of use cases.

Using this formula, the scores of the different formalisms are calculated and the resulting values are shown in Table 2.7. The resulting scores show that CTT, CTML, PDDL, and CCBM

Table 2.7: The table shows the candidate human behaviour models that can be used for modelling the three problems and the scores they received after summing up the requirements that were met.

Modelling formalisms	ACT-R	CTT	CTML	PDDL	Asbru	CCBM
$f(v)$	0.80	0.82	0.82	0.82	0.63	0.82

have the same values which is not surprising as they satisfy the same set of requirements regardless of the fact that depending on the formalism, they are implemented in a different manner. Asbru, on the other hand has a slightly lower score which is due to the fact that it does not support representation of interleaving actions. ACT-R has a score between Asbru and the remaining formalisms. This is due to the fact that it supports all requirements necessary to express the user behaviour in the three problems, but to our knowledge is unable to represent durative actions in probabilistic manner.

The results from the table also show that the requirements in question are not enough for selecting appropriate formalism. All formalisms with the exception of Asbru are able to represent the underlying behaviour and 4 of them also satisfy the full set of application-based requirements. For that reason, below we introduce several additional application-based requirements that were previously not considered.

- **Ability to support large state spaces:** As human behaviour contains by default high variability, this also results in large set of ways the same task can be executed. This in turn results in large state spaces. A formalism should be able to cope with such large state spaces.

- **Ability to support long observation sequences:** As real world activities usually take more than just a few minutes, the modelling formalism should be able to support reasoning given a long sequence of observations (e.g. in the 3-person meeting we have one meeting with 62 000 observations).

- **Factored action representation:** As the user behaviour in all three problems is goal oriented, reasonable heuristic for the correct selection of actions will be the goal distance from the initial to the goal state. Formalisms that rely on variable-free representation of the actions and the states run into the problem of being unable to find a path to the goal when the problem is more complex. A representation that relies on a collection of variables (namely factored representation), on the other hand is able to cope with that even in large problems [123, p. 366].

Table 2.8: The table shows the additional requirements the candidate human behaviour models have to satisfy in order to be used for modelling the three problems. *yes* indicates the requirement is satisfied by the modelling formalism, *no* – that it is not.

Features	ACT-R	CTT	CTML	PDDL	CCBM
large state-space	no	no	no	yes	yes
long observations	no	no	no	no	yes
factored representation	no	no	no	yes	yes

Table 2.8 shows the additional requirements that the formalism support[12]. It can be seen that large state spaces and factored action representation are supported by PDDL and CCBM which is due to the fact that the actions are represented in terms of templates with variables that can later be replaced by the corresponding constants. On the other hand ACT-R, CTT and CTML rely on variable-free representations making them unpractical for complex problems where many different variations of the user behaviour are involved. The last requirement – long observation sequences – has been shown to be supported only by CCBM in [80] where it was able to recognise the activities of a meeting that had 62 000 observations.

Table 2.9: The table shows the candidate human behaviour models – with the exception of Asbru as it is not able to support all behaviour based requirements – and the scores they received after summing up the initial and the additional requirements.

Modelling formalisms	ACT-R	CTT	CTML	PDDL	CCBM
$f(v)$	0.75	0.77	0.77	0.81	0.83

Table 2.9 shows the score for the modelling formalisms given the additional requirements and assuming that all three requirements are important. It reflects the fact that CCBM is the only one of the formalisms that satisfies all requirements thus has the highest score. As CCBM can be considered to be an extension of PDDL, we can conclude that for the given problems and assuming behaviour variability and goal oriented behaviour, planning languages seem to be the most suitable choice. This is due to the fact that they are able to express any logically correct variation of behaviour just by using action templates. This also makes them suitable for problems with large state spaces. The combination of planning language and probabilistic mechanisms like the **P**article **F**ilter (PF) theoretically allows them to be applicable also to infinite state-spaces as they do not need to expand the whole state graph in order to find a solution.

As a conclusion, based on the use cases and the corresponding requirements, CCBM can be considered as the preferred modelling choice. Thus it is the formalism that will be applied throughout the rest of the thesis. Chapter 3 gives some more insight into modelling with CCBM and presents the corresponding models that are intuitive solutions to the problems.

[12]Asbru is left out because it does not support all behaviour-based requirements thus we assume it is unable to represent the underlying user behaviour

2.7 Outlook

The chapter presented the preliminaries necessary for modelling human behaviour for AR. For that, the three modelling problems were introduced and the information that is to be incorporated into the models, was identified. Additionally, the corresponding test datasets were described.

Based on the modelling problems, the requirements for modelling them were identified. A formalism is then eligible for modelling the problems if all (or most) of the requirements are satisfied. Depending on the modelling needs, several formalisms were regarded, and the requirements they satisfy were discussed.

Later, a method for choosing the most appropriate formalism was introduced. Based on it, CCBM was suggested as the most fitting formalism.

In general, the chapter described the preliminaries needed for selecting an implementation language and technique for a given modelling problem for activity recognition. The output of this chapter is used as a basis for the models to be developed in the next Chapter 3.

Chapter 3

Modelling Human Behaviour with Computational Causal Behaviour Models

"The data is all over the place, the insight is yours, and now an abacus is at your disposal, too. I hope the combination amplifies each of these components."

Judea Pearl

Chapter Summary: *This chapter introduces the modelling with CCBM and presents the models that were developed as solutions for the problems in Chapter 2. Later, the models are analysed for identifying successful practices as well as modelling problems and the intuitive modelling approach is discussed. These practices, or patterns, and the problems are the basis for the modelling toolkit presented in Chapter 4. Furthermore, the intuitive modelling process is identified and analysed so that it can be later used as a basis for the development process proposed in Chapter 5.*

Chapter Sources: *This chapter is partly based on the paper "Plan Synthesis for Probabilistic Activity Recognition" [80]*

Questions to be answered in the chapter:

What are CCBM models for activity recognition? (In Section 3.2)

How were the three problems modelled? (In Sections 3.3.1, 3.3.2, and 3.3.3)

What context information, dimensions and performance does the meeting model have? (In Section 3.3.1)

What context information, dimensions and performance does the cooking task model have? (In Section 3.3.2)

What context information, dimensions and performance does the office model have? (In Section 3.3.3)

Is there a need for structured modelling process? (In Section 3.4)

What intuitive phases can be identified during modelling for activity recognition? (In Section 3.4)

3.1 Introduction

The previous Chapter 2 concerned itself with the preliminaries for modelling the three daily life problems. This chapter uses the preliminaries as a basis for implementing the problems' solutions. It provides some more details about Computational Causal Behaviour Models (Section 3.2) while a detailed introduction into modelling with CCBM is given in Appendix D. Later it discusses the intuitive solutions of the modelling problems (Section 3.3). Furthermore, to discover model pitfalls as well as useful modelling practices, it analyses the resulting models and discusses the different modelling problems. This information will later be used as the basis for developing modelling patterns that can be applied to different problems. Finally, the intuitive development process is analysed to be further extended in the following chapters (Section 3.4).

3.2 Computational Causal Behaviour Models

In the application domain of activity recognition we needed a modelling formalism that allows encoding prior knowledge about the problem domain that can be used during the inference phase to provide additional context information about the user's current state and intentions. Moreover, it had to be able to establish the bridge between causal modelling and probabilistic inference allowing a way for coping with uncertainty and the sensors' unreliability, and in the same time providing a means for building rich user models. The modelling formalism we chose is Computational Causal Behaviour Models because it satisfies all these requirements [76].

3.2.1 Causal Models

CCBM is a formalism that allows expressing human behaviour with a set of causally related rules using a PDDL-like notation. Every action in the causal model is described in terms of precondition-effect pair taking care that only actions which preconditions are satisfied could be executable. The possible user actions are expressed as abstract templates that are later parameterised with problem specific constants resulting in a set of grounded actions[1]. More formally, given a set of predicates $P := \{p_1, p_2, ..., p_n\}$, states x and x', and an action $a = (V, P_{pre}, P_{eff-}, P_{eff+})$, where V is a set of parameters used for parameterising the action's predicates, $P_{pre} \subset P$ is a set of preconditions, $P_{eff-} \subset P$ is a set of negative effects, and $P_{eff+} \subset P$ is a set of positive effects, an action a can then be specified as a mapping from state x to x'. To be executable in x, the preconditions of a have to be true in x, namely $P_{pre} \subseteq x$. Furthermore, after a takes place, the negative effects of a are excluded from x' and the positive are part of x, namely $x' \cap P_{eff-} = \emptyset$ and $P_{eff+} \subseteq x'$. In other words, every action in the described problem domain is represented as a transition from a certain state of the world to a new state of the world, thus allowing the reasoning about the current state's history and the actions that led from the initial state to it.

Of course, in order to be able to represent the available context information, the causal model contains not only the actions' templates, but is also populated with different problem-specific parameters that describe aspects of the available prior knowledge. Furthermore, the initial world state $x_{init} := \{p_{1\_init}, p_{2\_init}, ..., p_{n\_init}\}$ is described, which is taken as the starting

[1]Throughout the thesis the terms *grounded action* and *grounded predicate* comply with the notion of a *ground term* used in logic. Such term is one that contains no variables [123, p. 295]. For example, the action template *(move ?from ?to)* contains two variables, namely the begin and end positions. When it is grounded (or instantiated), the variables will be replaced by constants, resulting in e.g. *(move sink table)*, *(move stove sink)*, etc. The same procedure is applied to the predicates.

point for further causal reasoning. The simple representation of the abstract action templates, combined with the problem-specific parameters allows the model designer to build a relatively small description, that later is automatically grounded with the help of a planner, resulting in the model state-space. Thus, by modelling just a few abstract actions, it is possible to create models with state-spaces as large as several thousand to several million states [80].

3.2.2 Acting under uncertainty

Logic-based models alone could be good at inferring human behaviour, assuming that the states of the world were fully observed and the observations completely reliable. However, in reality that is hardly the case – usually, we are left with a set of observations taken from noisy sensor readings. As we mentioned in Chapter 1, to cope with this problem, one could combine causal models with probabilistic inference. This is done by grounding every feasible predicate from the causal models with every applicable parameter. This results in a world state where one possible occupancy of all predicates and the corresponding number of world states is defined by the number of all grounded predicates together with the number of applicable operators. Later, the state space is computed by the so called "reachability analysis" which is performed by first computing ground operators, and then expanding the state graph from the initial state. Every vertex of the state graph then represents a possible state in the Markov model, every edge is a non-zero entry in the transition matrix. The prior state distribution is calculated from the initial world state of the causal model together with the valid operators for this state. The transition function is generated by a planner that expands all possible plans leading to the goal and generates a directed acyclic graph with transition probabilities based on Formula 3.1 which states that for an action a and states x, x' such that $x' = a(x)$, the probability of selecting a in state x is then proportional to the influence of the revisiting factor (or was the action visited before), the goal distance (or how many actions have to be executed until the goal is reached), and the saliency (or what weight the action has in relation to all the remaining actions):

$$p(a\,|\,x) \quad \propto \quad \exp(\sum_{k=1}^{3} \lambda_k f_k(a,x)), \tag{3.1}$$

where $(f_k(a,x))$ is defined by

$$f_1(a,x) \quad = \quad \log\gamma(a(x)), \tag{3.2}$$
$$f_2(a,x) \quad = \quad \log s(a), \tag{3.3}$$
$$f_3(a,x) \quad = \quad \delta(a(x)). \tag{3.4}$$

Here $\gamma(a(x))$ is the revisiting factor that by default is 0, if the resulting state of applying the action a to the state x has already been visited. In other words, if the action was already selected once, it cannot be selected again. The value of the revisiting factor can be increased so that already visited states are allowed to be visited again. This factor is determined by the history of each single running hypothesis. Furthermore, $s(a)$ is the saliency of the action a that is specified in the action template specification. It allows the assigning of weights to the different actions, thus increasing their probability with respect to the rest of the available actions. The third feature $\delta(a(x))$ is the goal-distance of state $x' = a(x)$ that will be reached if action a is applied to state x. When using the goal distance as heuristic, the less actions that have to be executed before the goal is reached, the more probable the current state will be. In other words, assuming the agent is following some goal, she will try to reach the goal by following the shortest execution path. For that reason any actions that deviate from that goal or increase the goal distance, will have lower probability. By using λ_k, each feature can be weighted.

Furthermore, the formalism makes use of probabilistic action durations, which introduces the option of encoding a priori knowledge about the action duration to its definition. To do that a duration density function is assigned to each action. The probability of finishing the execution of an action α in the state s in the time interval (a, b) is then given by Equation 3.5.

$$P(a < \delta < b | \delta > a) = \frac{F(b) - F(a)}{1 - F(a)} \tag{3.5}$$

where F denotes the cumulative density function $p(\delta|\alpha, s)$. This enables each hypothesis to sample whether the action should continue or be aborted from F in each step.

This approach to combining causal models with probabilistic reasoning allows us to build relatively small causal models that are compiled into huge probabilistic models without the need of losing context information for the sake of simplicity. It also allows us to be able to reason about actions, places and objects that were never observed but that were encoded in the model and that are causally related to the observed world.

The resulting probabilistic model can have two forms – either a **H**idden **M**arkov **M**odel (HMM) or a PF. The HMM is a Markov model where the state is not directly visible, but the observed output, that is dependent on the state, is visible. One can then infer the hidden states based on the observed variables. HMMs are used in the case of a small causal model that generates probabilistic model where the whole state graph can be extended. In that situation an exact inference is performed as the probability of the states can be computed analytically. In the case, the model is too big to do that, the particle filter is used where the model state is approximated. The general probabilistic structure of the model can be seen in Fig. 3.1. In it

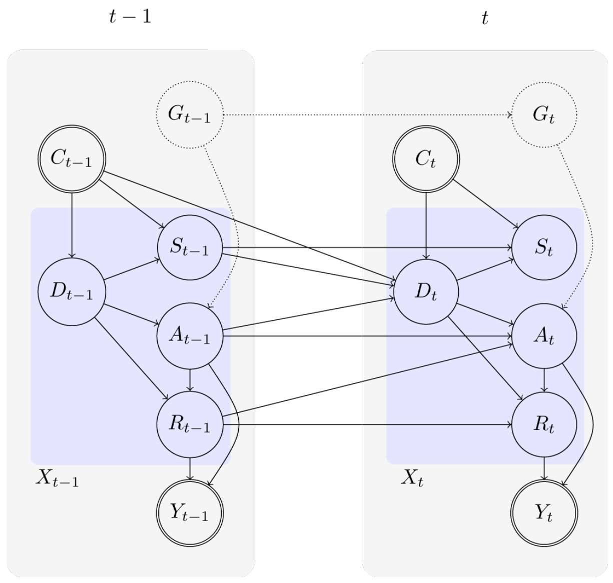


Figure 3.1: General probabilistic structure of Computational Causal Behaviour Models. In it C_t is the current observation time; G_t is the current goal; Y_t is the current observation. The current state X_t is captured by four features: D_t is the flag indicating whether the action should terminate in the interval between the current observation time and the previous observation time, A_t is the current action; S_t is the starting time for the current action; R_t indicates the new state for time stamp t.

we see two time slices $t-1$ and t. Each of them contains the following elements: Y_t is the observation data for time step t. C_t is the current observation time with the requirement that

$C_{t-1} < C_t$. D_t is the flag indicating whether the action A_{t-1} should terminate in the interval $(c_{t-1}, c_t]$. In case $D_t = 1$ new values are assigned to A_t, S_t, and R_t. Otherwise the random variables carry the values from the previous state. S_t captures the starting time for the action A_t. R_t gives the new state for the time step t either by applying the new action to the previous state, or by carrying over the old state when the action has not changed. G_t represents the current model goal. The variable is constant over time, allowing the computation of the goal distance. The figures indicates that given the current goal G_{t-1}, an action A_{t-1} is executed to follow this goal which results in the state R_{t-1}. Then depending on the flag D_{t-1} which is based on the time elapsed since the beginning of the action, the action will either terminate resulting in a new action A_t with a new state R_t, or it will be copied to the next observation slice so that the action's execution can continue until the flag D_t indicates that the action has to terminate. Then a new action is selected.

From the above we can summarise that CCBM provides the ability to model causally related actions in terms of precondition-effect templates that are later automatically grounded with problem specific parameters. This results in a state space graph that leads from the initial to the goal state. To allow probabilistic inference, the prior states probability is calculated based on the predicates that are true in the initial world state and the mechanism for selecting the next action to be executed is calculated with Formula 3.1 which ensures that always the most probable action will be selected until the goal state is reached. Finally, as each action has its own probabilistic duration, the decision whether the action will terminate in the next time step, or whether it will continue its execution, is defined by the decreasing action probability since the action execution has started. The above indicates that there are three different factors that play role when performing activity recognition with CCBM – these are the causal structure of the model, the action selection heuristics, and the actions durations. They all play an important role in the model's ability to correctly recognise the user actions.

3.3 Modelling the problems with Computational Causal Behaviour Models

Appendix D provides detailed introduction into modelling with Computational Causal Behaviour Models, while in this section the modelling formalism is practically applied to the three modelling problems described in Chapter 2. The provided models contain the intuitive solutions of the model designer to the problem at hand, which are later discussed in order to identify the modelling mechanisms used, the questions the model can answer, and the model performance. Each of the three problems is divided into several parts – first the model with its parameters is presented, then the model is analysed for problems and successful practices; finally, the model is evaluated in terms of activity recognition performance.

3.3.1 Modelling the 3-person meeting problem

Based on the problems' analysis performed in the previous chapter, here two different models are presented – a team model and a multi-agent one.

3.3.1.1 Model

Team model: The team model describes the single agents' behaviour as resulting from the team behaviour. Namely, the actions of the team define the behaviour of the single users, or with other words the effects of the team actions cause the separate user behaviour. It implements the elements presented in Table 2.1 in Chapter 2. Table 3.1 gives information about the actions

Table 3.1: Actions in the team model with the corresponding predicates that define their preconditions and effects, as well as the parameters that are used in the actions.

Action	Parameters	Predicates
move	*none*	(doing ?a - activity) – for all activities ?a; (at ?p - person ?l - location) – for all persons and locations
present	*?p - person*	(has-presented ?p); (doing ?p) (at ?p stage); (at ?p seat)
discuss	*none*	(have-discussed); (has-presented ?p - person) – for all persons; (doing discussing); (at ?p - person seat) – for all persons
leave	*none*	(doing moving); (at ?p - person door) – for all persons

modelled in the team model and the parameters and predicates each of them has. It could be seen that there are only five actions where only one of them has a parameter – that is the action *present* and the presence of the parameter is to indicate that there are three presentation slots each corresponding to one of the three participants. However, the parameter does not mean there are three agents that can execute the action. It can be executed just by one (the default agent) and simply indicates that there are three instances of the same action with a different user as parameter. The remaining actions, on the other hand, apply their effects to all modelled users, so that when the action is executed, it affects everyone in the environment. Thus it is impossible to track the behaviour on a single-user level.

Table 3.2: Model parameters for the team and the multi-agent models.

Parameter	Team model	Multiagent model	Description
# operators	6	88	grounded actions after the model compilation
# predicates	19	72	grounded predicates after the model compilation
# object types	3	4	see Fig. 3.7
# persons	3	3	objects of type person
# locations	3	7	objects of type location
# activities	6	6 team + 10 single-user	activities to be estimated
# states	31	5568	state-space of the model
# valid plans	13	3515	valid plans leading from the initial to the goal state
# hierarchy level	2	3	levels of the type hierarchy
# goal distance	10	48	minimum distance from the initial to the goal state
# max. branching factor	4	9	maximum number of possible actions at a given time

Furthermore, Table 3.2 contains some additional information about the model characteristics. The first column describes the name of the given model parameter, the second the value associated with it, and the third – the value for the multi-agent model. The considered characteristics are the number of operators which represents the number of grounded actions; the number of predicates that shows how many predicates are there after they were grounded with the available objects; the number of object types, which presents in how many categories were the objects divided; the number of persons, locations, and activities represents how many constants were there from a given type; the number of states represents how many states are there after the model graph was expanded; the number of valid plans represents all plans that lead from the initial to the goal state; the hierarchy level describes the hierarchy of the object types with the highest level being the default *object* type; the goal distance that shows how long is the shortest path from the initial to the goal state; it is later used for action selection heuristic (according to Formula 3.4); and finally, the maximum branching factor shows how many actions at most are executable from a given state. We consider these parameters as they give us information about the model size and complexity.

It can be seen that the model is comparatively small with only 6 grounded operators and 19 grounded predicates. It has only 3 object types that describe the persons that participate in the meeting, the locations, and additionally one type for the actions to be estimated. It can also be seen that there are only 3 objects of type location, which is due to the fact that the locations were simplified and instead of taking each seat or stage as a separate location, they were combined in one seats location, one stages location and a door. After the model compilation the abstract model definition resulted in 31 causally related states and 13 valid plans could lead from the initial to the goal state[2].

Multi-agent model: Table 3.3 shows the actions the model has. The model implements the elements presented in Table 2.2 from Chapter 2. It can be seen that each of the actions has at least one parameter of type *person* and additionally each of the actions could be executed by any of the agents in parallel with the actions of the rest of the agents.

Table 3.3: Actions in the multi-agent model.

Action	Parameters	Predicates
start-enter	*?p - person*	(entering ?p); (at ?p door); (entered ?p); (idle ?p)
finish-enter	*?p - person*	(entering ?p); (idle ?p)
sit-down	*?p - person ?s - seat*	(at ?p ?s); (idle ?p); (seated ?p); (has-discussed ?p)
get-up	*?p - person ?s - seat*	(at ?p ?s); (idle ?p); (seated ?p)
walk-to-seat	*?p - person ?from - location ?to - seat*	(idle ?p); (entered ?p); (is-seat-for ?to ?p); (at ?p ?from); (walking ?p ?to)
walk-to-stage	*?p - person ?from - location ?to - stage*	(entered ?p); (is-stage-for ?to ?p); (has-presented ?p); (at ?p ?from); (idle ?p); (walking ?p ?to)
walk-to-door	*?p - person ?from - seat*	(idle ?p); (entered ?p); (at ?p ?from); (has-presented ?p1) – for all persons; (walking ?p door)
arrive	*?p - person ?from ?to - location*	(walking ?p ?to); (at ?p ?from); (at ?p ?to);
exit	*?p - person*	(idle ?p); (at ?p door); (entered ?p); (has-presented ?p1) – for all persons
start-presentation	*?p - person ?s - stage*	(has-presented ?p); (decided-to-present ?p); (idle ?); (at ?p ?s); (is-presenting ?p)
finish-presentation	*?p - person*	(has-presented ?p); (decided-to-present ?p); (idle ?); (at ?p ?s); (is-presenting ?p)
prepare-discussion	*?p - person*	(discussing ?p); (has-discussed ?p); (has-presented ?p1) – for all persons; (prepared ?p)
start-discussion	*?p - person*	(prepared ?p); (discussing ?p)
finish-discussion	*?p - person*	(discussing ?p); (has-discussed ?p1) – for all persons

Additionally, Table 3.2 shows the model dimensions and characteristics. It could be seen that the model is considerably larger than the team model. Whereas the team model has only 31 states in its state space, the multi-agent has 5568 which would be an impossible task to build by hand. This is due to the fact that the model has much more functionality than the team model and is able to explain the behaviour also on the single-agent level and to provide additional context information.

3.3.1.2 Model analysis

In order to identify different modelling practices and their influence on the model, here we analyse the two models. First the way in which the actions are modelled is discussed; later it is explained what kind of context information the model can provide; and finally the model parameters, presented in the previous section are analysed.

[2]Valid here indicates that it is causally possible, however that does not mean all of these plans correspond to the actual actions execution or that they make sense from a human point of view.

Modelling the user actions: In the team model just the team actions are defined at the model level, the effects of which capture the joint effects of the different individual actions (which are invisible at the modelling level). In other words, the action's effects force the agents to exhibit specific behaviour regardless of their individual desires. Furthermore, this approach to modelling team behaviour does not allow unsynchronised actions between the agents. The individual actions executed in parallel have the same duration, as well as the same begin and end times. To illustrate this approach, Fig. 3.2 gives an example with the action *present*. Here the model forces all team members to simultaneously take a seat no matter where they are located at the given time and then start the presentation. This type of modelling is not very suitable for multi-agent behaviour modelling as it lacks the flexibility needed to express the individual dynamics. On the other hand, it is relatively simple approach that is able to successfully identify the team actions in cases where the reasoning over the single agents' behaviour is not needed.

```
(:action present
  :parameters (?presenter - person)
    :duration (gaussian (presentation-duration ?presenter))
    :precondition (and (active-moving)
            (not (has-presented ?presenter)))
    :effect (and (not (active-moving))
                      (active-presentation)
                      (forall (?p - person)
                              (when (not (= ?p ?presenter))
                                    (at ?p seat)))
                           (at ?presenter ?stage))
)
```

Figure 3.2: Present action template for the team meeting model.

In difference with the team model, the multi-agent model represents the team behaviour as an effect of the single user behaviour, or in other words, the single agents decide for themselves how to act and by their actions define the team behaviour. For example, consider starting a presentation in the multi-agent model. The action is shown in Fig. 3.3.

```
;;; begin action part ;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;

(:action start-presentation
  :parameters (?presenter - person)
  :agent ?presenter
  :duration (gaussian (presentation-duration ?presenter))
  :precondition (and (at ?presenter stage)
            (forall (?p - person) (when (not (= ?p ?presenter))
                (at ?p seat)))
  :effect (active-presentation)
)
;;; end action part ;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;

(:action finish-presentation
  :parameters (?presenter - person)
  :agent ?presenter
  :precondition (and (is-presenting ?presenter)
          (forall (?p1 - person)(not (may-walk ?p1))))
  :effect (and (has-presented ?p)
        (not (is-presenting ?p))
        (may-walk ?p)(idle ?p))
)
```

Figure 3.3: Present action template for the multi-agent meeting model.

Here, the condition of all non-presenting persons being seated is not *established* as joint effect of a team action, it is rather a *precondition* the presenter has to obey before she can (reasonably) start her presentation. In a multi-agent model, agents have to rely on other agents to cooperate – on the other hand, this gives a single agent more degrees of freedom regarding her individual behaviour. This comes as a disadvantage for the model designer: in a team model, social protocols (e.g., not getting up in the middle of a presentation) can be enforced simply by not considering such *misbehaviour* in the joint actions. In a multi-agent model, where each agent has the choice to freely select any applicable action, explicit locking mechanisms have to be provided that allow any agent to *disable* actions of other agents which are not appropriate in the current situations.

Furthermore, the action *present* is divided into a *begin-end action pair* (or a macro) consisting of *start-presentation* and *finish-presentation* which allows the definition of actions that do not depend on other team actions for another activity to take place. This results in every agent having the ability to choose her own action that is independent of the team behaviour. For example, in *start-presentation* the presenter can start her presentation only after all the other users are seated and she is at the stage, which allows the separate participants to decide on their own to sit for the presentation. The presenter alone cannot force the *present* team activity on the single agents like in the team model. After the preconditions for *start-presentation* are satisfied and the action is executed, the agents are locked in the *present* phase for the duration of the activity. Afterwards, the only possible action is *finish-presentation* that unlocks the users and allows them to choose new actions independent of the team behaviour. This kind of modelling allows a serious degree of freedom for choosing the next possible action for every single user and could reduce the goal oriented team behaviour to nonexistent. To solve this problem we introduce a lock mechanism that reduces the set of actions executable from a certain state. For example, an agent arriving at a place, sets the lock flag *(idle ?p)* to true which allows her to choose another action to perform. On the other hand the lock flag *(may-walk ?p)* is set to false which stops her from moving to another location, thus forcing her to wait until the other agents have made their choices and have been forced to arrive at a state of the world where only the action *start-presentation* can be performed. Using such mechanisms for every action in the model, allows us to capture the independent behaviour of the single agents and in the same time still to be able to detect the team behaviour.

Available context information: As one of the goals of the models is to support context information, each of them can provide more than just the actions that are being executed. The team model is relatively simple and consists of only four actions. Three of the four actions do not have parameters (*move*, *discuss*, *leave*) which indicates that they can give us only information about the actions themselves. This is caused by the need to model the team as a single entity which does not allow pinpointing additional information about the single users. The fourth action *present* has one parameter of type *person* which indicates that the model is able to provide information about who of the users is the one currently presenting.

Furthermore, it can be seen from the model predicates, that there is not much more context information that can be obtained from them – the only additional knowledge there is the user position, but as it is forced on the whole team, the predicate cannot say whether the users locations actually differed when the action was executed. For example, during the *discuss* action, the predicate *(at ?p seat)* holds for all users and it is not clear who sits where exactly as the seats are represented by one abstract *seat* object.

In difference with the team model, the multi-agent model is able to support much more context knowledge. There are 14 actions modelled and each of them has at least one parameter (see Table 3.3). Some of these actions are combined into *begin-end* action pair which as explained above allows the single users to be independent of the team behaviour and to make decisions to follow the team goal on their own. The first action pair is *enter* which has a parameter of type *person*. This indicates that the model is able to provide knowledge about which agent exactly entered the room at a given point. The same applies for the action *exit*, as well as for the action pairs *present* and *discuss*. Additionally, as each of these actions is bound to a certain location, the model is also able to provide the separate users locations while executing an action. The

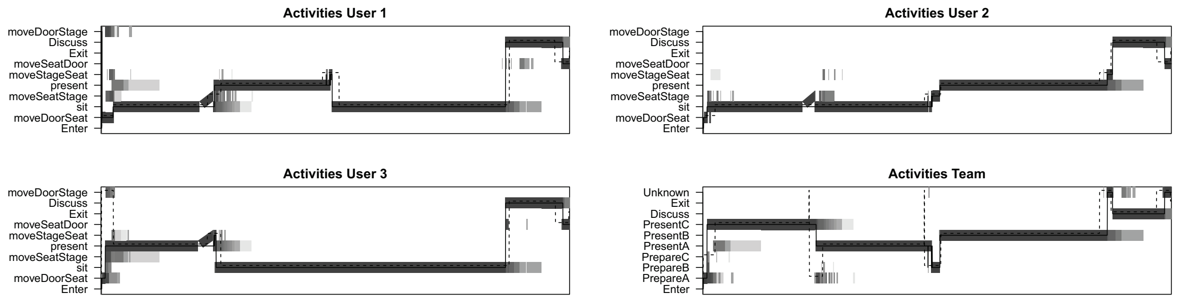


Figure 3.4: The figure shows the users separate actions as well as the team actions during the meeting. Here the darker the region, the more probable it is that at that time the user was doing the corresponding actions. Furthermore, the solid black line indicates the estimated state, while the dashed line shows the action that was actually executed.

same applies for the actions *sit down* and *get up* which are able not only to pinpoint to a concrete user but also to the specific seat she is sitting on. Similarly, the action pairs *begin-end move* are able to give information about who is moving, from where she started, and what is her destination. Based on these action templates the model is able to give information about what exactly the user is doing and what is the corresponding team action to the separate user actions. Fig 3.4 shows the probability for the user actions and the team actions during the meeting.

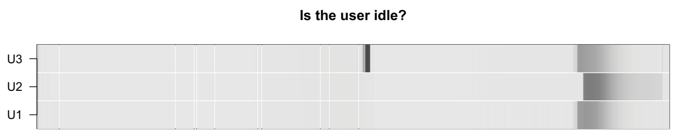


Figure 3.5: The figure indicates the idle states of the users throughout the meeting. The darker the colour, the more probable it is that the user was idle at that time point.

Additionally, the model predicates can provide knowledge about whether the person is idle at the moment or doing something toward achieving the team goal, which can be seen in Fig. 3.5. They can also tell us what is the user destination at a given point in time as can be seen in Fig. 3.6.

Modelling such context information, the model is no longer just a means for recognising activities, but also for providing any kind of prior knowledge the designer might deem important to infer.

Model parameters: The models' parameters were already presented in the previous section in Table 3.2. It can be seen that while the team model has only 6 grounded operators, the multi-agent has 88. Thus together with the increased number of predicates (72 against 19 for the team model) and the increased branching factor (9 against 4), the model implies much larger state-space (5568 against only 31 for the team model) and a higher degree of freedom for the agents. This indicates that the multi-agent model is able to explain much more variations in the

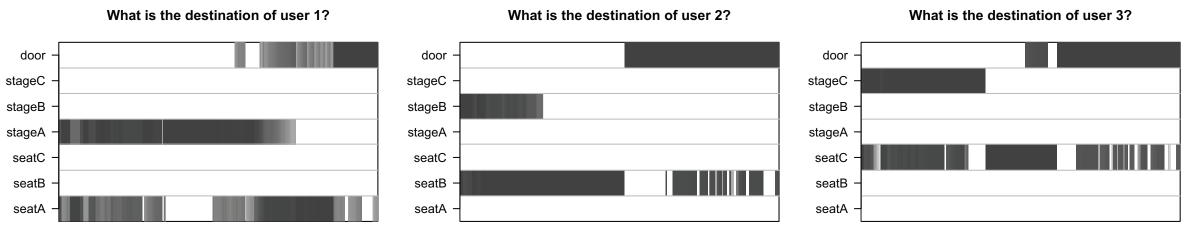
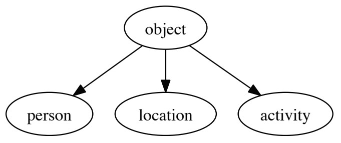


Figure 3.6: The figure shows the users destination during the meeting. Here the darker the region, the more probable it is that at that time the user is heading for the given location.

behaviour of users (3515 valid plans generated by the multi-agent model vs. 13 valid plans for the team model) but also the danger of inferring the wrong action increases[3]. The high degree of freedom and the increased state-space also influence the goal distance that shows the shortest path from the initial to the goal state. In this case it has a length of 48 compared to the just 10 different states through which the team model has to pass. This is a result of the begin-end action pairs modelled in the multi-agent model and the additional help actions such as *prepare to present* and *prepare to discuss* which increase the number of actions that have to be executed before the goal is achieved. Additionally, each action is now parameterised for three users and not just for the team.



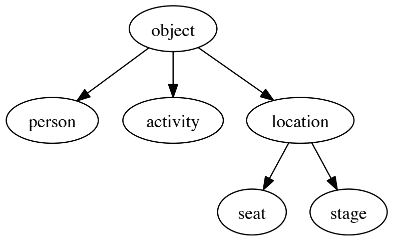


Figure 3.7: The figure shows the type hierarchy for the meeting models. To the left is the type hierarchy for the team model and to the right – for the multi-agent model.

The multi-agent model has almost the same object types as the team model with the difference that the location type is divided into two additional subtypes (see Fig. 3.7). In large models, such specialisation leads to reducing the state-space as a given action will be grounded only with the specific subtype of parameters resulting in less grounded operators. For example, if we take the action *sit down*, in the current multiagent model it has as parameters variables of types *person* and *seat*. When grounded, this will result in 9 operators. On the other hand, if the *seat* type is replaced by the more general *location*, it will result in 21 grounded *sit down* operators. This is explained by the fact that while in the team model there are only 3 different locations, in the multi-agent there are 7, so the type hierarchy takes care of removing unwanted grounded actions. The team model, on the other hand, did not make use of the objects type hierarchy, simply because with the exception of the three user objects, there is only one object representative from a given type.

[3]Here also stands the question whether 3 persons are able to execute a given plan that consists of just 10x3 activities in 3515 reasonable ways. This also leads to the question whether we need such high degree of freedom when a more limited model will be also able to explain most of the reasonable (from a human point of view) execution paths.

3.3.1.3 Performance evaluation

To evaluate the model performance, the two models were compiled using the same type of observation model in which the stages area was described as a rectangular region and the seats as circles. Each runtime model was then used to perform state estimation, where in the case with the team model, it was compiled for both exact inference (an HMM) and approximate inference (a particle filter). That was due to the fact that the small state space allows compilation for exact inference. In the case with the approximate inference, to diminish the influence of a given random seed, the state estimation was performed 50 times for each of the meeting datasets, and later a majority voting performed in order to obtain the most likely execution path.

To evaluate the model performance, three different measurements were used – these are accuracy, precision, and specificity. Accuracy is the degree to which the estimated behaviour represents the real behaviour and is computed according to Formula J.1 on page 245. Precision represents the degree to which a positively inferred behaviour represents the underlying behaviour it is being tested for. It is computed by Formula J.4 on page 245. Specificity represents the degree to which inferred negative instances reflect the actual negative instances and is calculated according to Formula J.7 on page 246.

Furthermore, a Friedman test was performed to identify whether the results for the different meetings from the same model significantly differ from each other. Description of Friedman test can be found in Appendix H. A detailed description of the above measurement metrics can be found in Chapter 5.3.

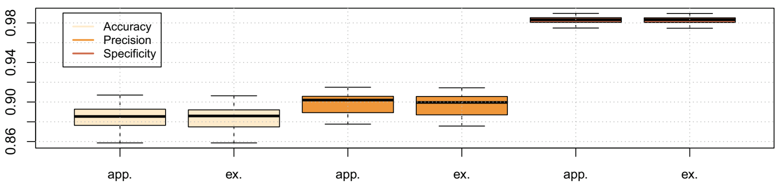


Figure 3.8: Overall model performance of the team model for the 20 meetings. The performance of exact and approximate inference is compared.

Fig 3.8 shows the results from the team model compiled into an HMM compared to those from a particle filter with 10 000 particles. It performed an exact inference to the 20 small datasets describing the meeting scenario and inferred the team actions based on the sensor data. It could be seen that the overall performance is about 89% which is comparable with the results obtained by using a hand-crafted HMM presented in [80] that run on the same dataset and managed to achieve a maximum accuracy of 92%. It can also be seen in Fig. 3.8 that the model performed with similar accuracy, precision, and specificity in the form of particle filter. This shows that even when using approximate inference the model is able to achieve high performance level (at least for small state spaces). Furthermore, Fig 3.9 shows the performance for the different meetings for all 50 runs in the case of the particle filter. It can be seen that for the different runs there is no high variance in the performance. It further shows that the model is relatively independent of the random seeds used in the filter. This result stands to show that the performance of a generated probabilistic model is not significantly worse than that of a hand-crafted and trained model. This indicates that provided a designer with knowledge about the problem context, the training of a probabilistic model can be substituted with a priori

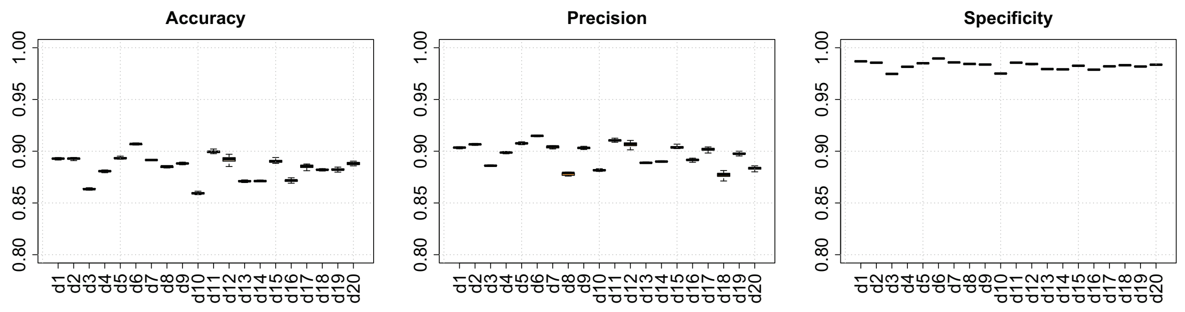


Figure 3.9: The approximate model performance for the different meetings. For each meeting, a box represents the performance for the 50 runs.

knowledge allowing the encoding of (sometimes) much more information than can be achieved by hand.

To prove that statement, Fig. 3.10 shows the results from the multi-agent model which has much more states compared to the team model (5568 states to 31 in the team model). It can be seen that the performance of the multi-agent model on a team level is comparable to that of the team model (with about 89% accuracy). This indicates that regardless of the much larger number of states, the multi-agent model does not decrease the recognition rate. In the multi-agent case however, the model is able to provide the additional information about the single user activities (as already shown in Fig. 3.4). The model was also able to estimate the separate users' states with about 92% recognition rate. Additionally, it was able to provide information about which user is doing exactly what, the concrete begin point and destination for the *move* actions, the concrete user and her position while sitting or presenting, instead of just the more abstract team state, associated with the combination of the actions of the three users.

One drawback to the multi-agent model that can be seen in Fig. 3.10 is that the different runs for dataset 10 have much higher variance than in the team model. This indicates that the high degree of freedom combined with this inference mechanism, also makes the model much more dependent on the random seed used for the given run. It also implies that the model and the corresponding inference engine did not capture the behaviour of dataset 10 in an optimal manner. This artefact can be reduced either by introducing more model constraints or by better action selection heuristics[4].

Finally, the same multi-agent model was able also to recognise the activities from the long meeting dataset that had slightly different agenda (the discussion was not performed) and that was recorded while a real non-staged meeting took place. It can be seen in Fig. 3.11 that the recognition rate is higher than that for the small meeting. However, that can be explained with the fact that the long meeting lasted about 50 minutes compared to the 3 minutes per meeting for the small meeting datasets. That resulted in long time slots without state change, thus the recognition increased compared with the more often occurring state changes in the small model. Furthermore, this also resulted in less performance variability for the different runs compared to the small meetings (a variance of less than 1% for the long meeting compared to about 6% for the small meetings).

To discover whether the model performance significantly differ for the different datasets, Friedman test was applied [49]. Table H.2 and Table H.4 in Appendix H show the results from the test. The results for this model are labelled *intuitive*. It can be seen that there was significant difference in the accuracy and specificity for the different datasets. On the other

[4]It should be noted that in this case the system model introduces additional modelling mechanisms in order to fix problems with the probabilistic inference engine, and not because of shortcomings in the system model.

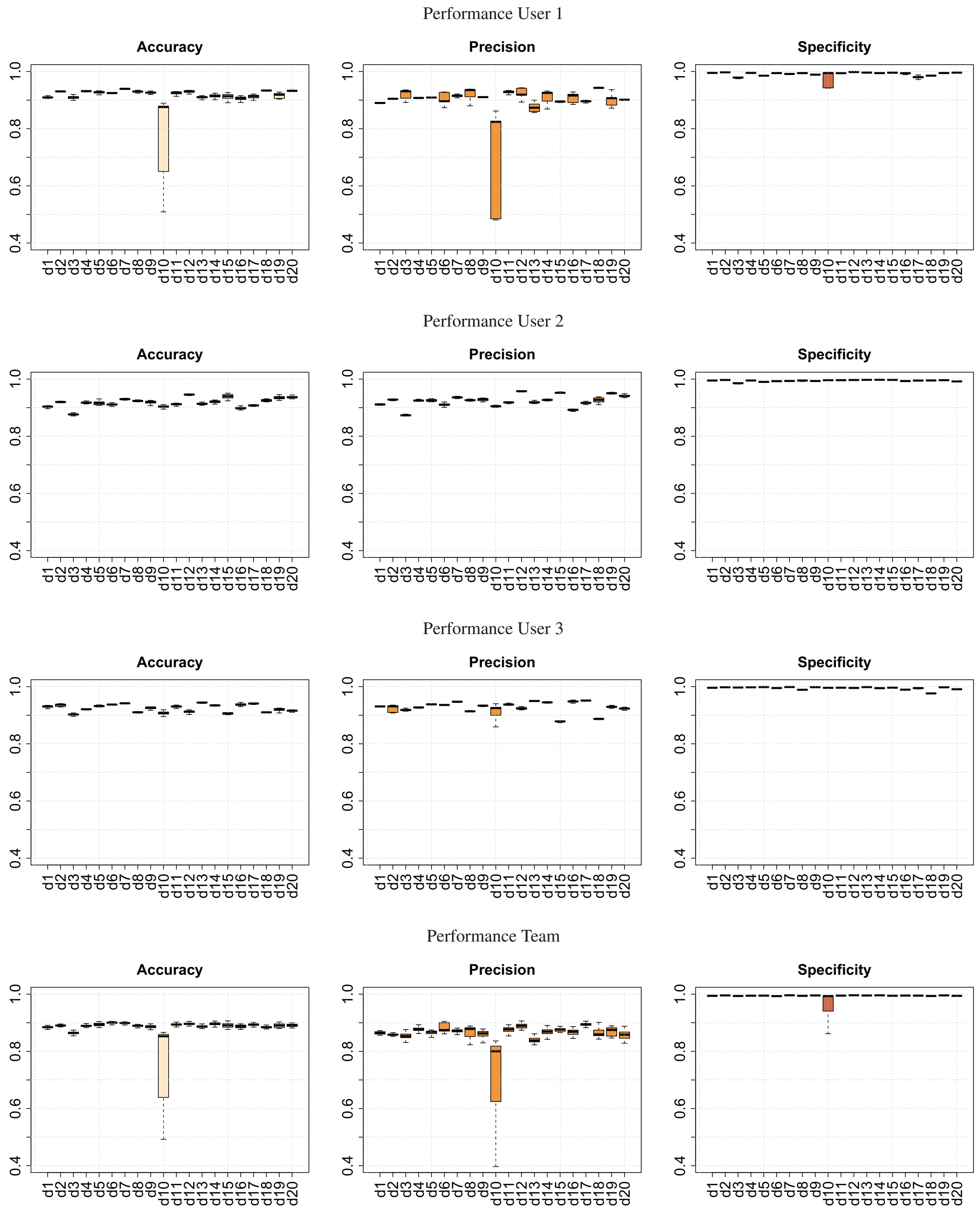


Figure 3.10: Model performance for the multi-agent model. Here for each of the users as well as for the team the accuracy, precision, and specificity are plotted. For each meeting the results for 50 runs are plotted as boxes.

hand, the precision was not influenced and any differences there could be regarded as caused by chance. Taking as an input the results of all performance evaluation metrics (accuracy, precision, and specificity), the Friedman test showed that there is a significant difference in the model performance across the different datasets. This indicates that the model was influenced by variations in the data, which implies that its ability to represent the behaviour variability

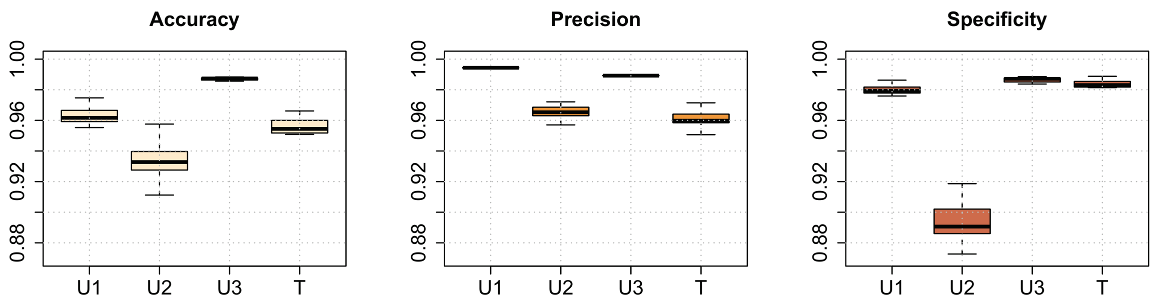


Figure 3.11: The figure shows the model performance for the long meeting. Here the x-axis represents the three users (U1, U2, U3) and the team (T). The performance for 50 runs is plotted as a box.

influences the model performance. This can be interpreted as caused by model overfitting where the relatively high model performance makes the model sensitive to variations in the data. That shows the effect of the bias-variance dilemma, where the reduction of the model bias results in increase of the model variance.

3.3.2 Modelling the kitchen task assessment problem

3.3.2.1 Model

The cooking task problem is a single agent problem that consists of one person interacting with several different objects that can be positioned at multiple locations or which could be used for completing a variety of actions. This makes the problem relatively complex because the high number of environment elements and the user nondeterminism produce a high degree of freedom in the possible set of executable actions. This also results in a huge state space, making it difficult for the inference engine to infer the correct action being executed. It turns the solution of the problem into a challenge, as the described behaviour relates a real world situation where a person can interact with any number of environment elements and choose from multiple completely different actions.

Table 3.4 shows the actions used in the model to describe the user behaviour. It implements the elements from Table 2.3 from Chapter 2. It could be seen that there are multiple variations of one action that create specialised action templates. The actions are 27 alltogether where there are 4 variations of *take*, three of *put*, three of *wash*, three of *fill*, and two of *open* and *close*.

Table 3.5 shows the parameters the cooking task model has. It can be seen that the model has 111 grounded operators and 129 grounded predicates which results in extremely huge state space (more than 600 million states[5]), thus the assumption of the state space being bigger than the given values. It also has 23 object types; one person, 4 locations, 10 objects, 6 places, and 16 activities to be estimated. There are also more than 687 million valid plans that lead from the initial to the goal state, and a complex type hierarchy of 4 levels is used. The minimal goal distance could not be calculated as the reachability analysis was never completed but it could be approximated based on the actions that had to be executed in the experiments. The maximum number of actions from which can be chosen in a given state is 10 in the case of **O**bjects **P**laces **L**ocations (OPL) as observations. In the case of **O**bjects **P**laces (OL) the branching factor was 12; in the case of **O**bjects (O) – 24; in the case of **P**laces **L**ocations (PL) – 32; and in the case

[5]The state space was analysed until it reached 675 million states due to the limited computing power available for the calculations (120GB of RAM).

Table 3.4: Actions in the cooking task model. For each action also the predicates used in the preconditions and effects are given, as well as the parameter used in the action.

Action	Parameters	Predicates
move	*?from ?to - location*	(isat ?from); (isat ?to); (adjacent ?from ?to); (reachable ?p1) – for all places adjacent to ?to
take-one-object	*?m - movable ?from - fixed-place*	(taken ?m); (taken ?from); (can-be-at ?m ?from); (at ?m ?from); (reachable ?from)
take-food	*?m - meal*	(taken ?m); (isin ?m cutting-board); (reachable cutting-board)
take-from-container	*?m - movable ?from - special-container*	(taken ?m); (taken ?from); (can-be-at ?m ?from); (at ?m ?from); (reachable ?from)
take-two-objects	*?m - twomovable ?from - fixed-place*	(taken (object1 ?m)); (taken (object2 ?m)); (can-be-at (object1 ?m) ?from); (can-be-at (object2 ?m) ?from); (reachable ?from)
put-one-object	*?m - movable ?to - fixed-place*	(taken ?m); (taken ?to); (can-be-at ?m ?to); (at ?m ?to); (reachable ?to)
put-into-container	*?m - movable ?to - special-container*	(taken ?m); (taken ?to); (can-be-at ?m ?to); (at ?m ?to); (reachable ?to)
put-two-objects	*?m - twomovable ?to - fixed-place*	(taken (object1 ?m)); (taken (object2 ?m)); (can-be-at (object1 ?m) ?to); (can-be-at (object2 ?m) to); (reachable ?to)
wash-hands	*?h - hands*	(isat sink); (washed ?h)
wash-food	*?m - meal*	(isat sink); (is-cut ?m); (washed ?m); (taken ?m)
wash-object	*?m - movable*	(isat sink); (taken ?m); (taken sponge); (washed ?m)
cut	*?m - meal*	(is-cut ?m); (taken knife); (at cutting-board counter); (isat counter); (washed ?m); (isin ?m cutting-board)
fill-meal	*?m - meal ?c - other-container*	(isin ?m ?c); (taken ?m); (isat counter); (is-cut ?m)
fill-from-liquid-container	*?d - drinkable ?c1 ?c2 - liquid-container*	(isin ?d ?c1) (isin ?d ?c2); (opened ?c1); (opened ?c2); (isat counter); (can-be-filled-from-to ?d ?c1 ?c2)
fill-from-container	*?m - meal ?c1 ?c2 - other-container*	(isin ?m ?c1) (isin ?m ?c2); (isat counter); (can-be-filled-from-to ?d ?c1 ?c2)
turn-on-stove	–	(turned-on); (was-turned-on); (isat counter)
turn-off-stove	–	(was-turned-on); (isat counter)
cook	*?e - eatable*	(empty pot); (isin ?e pot); (turned-on); (at pot stove); (taken wooden-spoon); (isat counter)
open-container	*?o - openable*	(opened ?o); (taken ?o); (isat counter)
close-container	*?o - openable*	(opened ?o); (taken ?o); (isat counter)
open-cupboard	*?c - cupboards*	(opened ?c); (isat counter); (at plate ?c); (at glass ?c)
close-cupboard	*?c - cupboards*	(opened ?c); (isat counter); (at plate ?c); (at glass ?c)
sit-down	–	(seated); (isat table); (hungry); (thirsty)
get-up	–	(seated); (isat table); (hungry); (thirsty)
eat	*?e - eatable*	(seated); (cooked ?e); (taken spoon); (isin ?e plate); (has-eaten); (hungry)
drink	*?d - drinkable*	(seated); (taken glass); (isin ?d glass); (has-drunk); (thirsty)
wait	–	(isActive waiting)

of **L**ocations (L) – 58[6]. Due to the high degree of freedom, the model contains additional lock predicates that play the role of constraints limiting the possible actions that can be executed from a given state. This is the reason why the action templates contain complex descriptions with various mechanisms restricting the states from which an action could be executed.

3.3.2.2 Model analysis

Modelling the user actions: It was already mentioned that the model consists of many specialised actions. This is due to the attempt to reduce the state space, as more general action templates increase the number of parameters with which they can be parameterised, thus the number of grounded operators. More specialised actions, on the other hand, allow the pa-

[6]As the state space was never completely explored, the maximum branching factor was calculated given the observations. In other words, how many actions at most were available during inference given the specific type of observations.

Table 3.5: Cooking task model parameters.

Parameter	Value	Description
# operators	111	grounded actions after the model compilation
# predicates	129	grounded predicates after the model compilation
# object types	23	see Fig. 3.15
# persons	1	objects of type person
# locations	4	objects of type location
# objects	10	objects of type object (divided into several subtypes)
# hand locations	6	objects of type place
# activities	16	activities to be estimated; correspond to the activity object types
# states	$> 657\,000\,000$	state-space of the model
# valid plans	$> 687\,000\,000$	valid plans leading from the initial to the goal state
# hierarchy level	4	levels of the type hierarchy
# goal distance	approximately 80	minimum distance from the initial to the goal state
# max. branching factor	10 (OPL), 12 (OL), 24 (O), 32 (PL), 58 (L)	maximum number of possible actions at a given time

rameterisation only of specific types of parameters. Thus they reduce the number of grounded operators but increase the number of grounded predicates as new predicates are introduced to describe the more special actions.

Additionally, the model introduces several predicates that are used as lock mechanisms in order to reduce the number of possible plans that can be executed. These predicates indicate whether or not a person can go to a given location, whether an object can be manipulated or located at a given place.

```
(:action move
  :parameters (?from ?to - location)
  :duration (moveDuration)
  :precondition (and
    (isat ?from)
    (allowed-to-move ?from ?to)
    (not (= ?from ?to))
    (not (adjacent ?from ?to))
      (imply (= ?to table)(and (not (empty plate))(not (empty glass))))
    )
  :effect (and
    (isat ?to)
    (not (isat ?from))
    (forall (?p - fixed-place) (not (hand-at ?p)))
    (forall (?p - place)(and
      (when (adjacent ?from ?p)(not (reachable ?p)))
      (when (adjacent ?to ?p)(reachable ?p))
      (when (and (adjacent ?to ?p)(adjacent ?from ?p))(reachable ?p))
      )
    )
    (forall (?a - activity)
          (and (when (= ?a moving)(isActive moving))
              (when (not (= ?a moving))(not (isActive ?a)))))
    )
)
```

Figure 3.12: Move action template for the cooking task.

To further control the number of executable actions from a given state, even more general actions contain relatively complex description. For example, Fig. 3.12 shows the description of the action *move* where it can be seen that the predicate *(allowed-to-move)* takes care that the action is not executable during a certain stage of the meal preparation (e.g. the action *cut* sets the predicate to false for the sink as destination, making the action *move* impossible until the meal is prepared). Additionally, the predicate *(adjacent ?from ?to)* takes care that the person is not attempting to move between places that are adjacent to one another as they are reachable also without moving. On the other hand, the effect of the action takes care to set the predicate *reachable* to true for all places that are adjacent to the place where the person is located; while the predicate *isActive* sets to true the flag for the action being currently executed. With such complex descriptions the model copes with the high degree of freedom and limits the number of possible actions, making it possible at all to infer the action being executed.

Available context information: The cooking task model is a complex representation of the real world and can provide information about a number of elements. Table 3.4 shows the available action templates and their parameters. From it can be seen, that the model is able to provide knowledge about the location of the user, the objects being manipulated, their locations, the start point and destination of the user and the objects she is carrying with her. This can be seen in Fig. 3.13 that shows the probability distributions for different questions that the model can answer, such as *Where is the drink?*, *What is the meal?*, *Which objects have been washed?* etc. The questions regarding the user location and manipulated objects can be withdrawn directly from the sensor data when such is available, but in the case when one of these types of observations is not available, the model is still able to reason about the questions. Additionally, Fig. 3.14 shows the probability distribution for the actions that are being estimated.

The only action that does not provide any useful information is the action *wait* which is implemented to cope with situations where the user is just standing without moving hands or doing something. There are four more actions without any implicit parameters – *sit down*, *get up*, *turn on*, and *turn off*[7]. However, they are still able to provide information about the user location, and if objects were previously taken and are being held, the execution of these actions will not change or stop the flow of information about these objects. Additionally, the actions *take* and *put* are able to give information about the location of the user, the location from where an object was taken or where it will be left. The actions *wash*, *cut*, *open*, and *close* provide information about the object being manipulated and the place where this manipulation is taking place. They are also able to provide information about the user location. Finally, the action *fill* contains prior knowledge about the food or liquid being filled, the container from which it was willed, and the destination container. Furthermore, the actions' predicates allows to obtain such information as which places are reachable, where is the drink or the carrot that has been cooked, which objects have been washed, and when is the person hungry or not etc. (see Fig. 3.13). By using this encoded information, the model is able to provide much more than just the action being executed. Depending on the application using the activity recognition component, it gives user specific information that can later be used to adequately assist her.

Model parameters: The model complexity is reflected in its parameters. It can be seen that there are more than a hundred operators and predicates which is almost double the values for the multi-agent model in the meeting scenario. However, this does not double the state space but rather makes it more than a hundred thousand times bigger. Although the cooking task is a single user problem, it has more actions to be estimated (16 against 10 in the meeting) and more objects that are being manipulated (10 against 0 for the meeting) which results in a higher degree of freedom (branching factor of 10 when observing objects, locations and places, and increasing with removing any of these sensors[8]), and much more valid plans that

[7]They however have some explicit parameters that are encoded in the actions' descriptions and that provide additional context information (e.g. the place *stove* is explicitly encoded in the *turn on* and *turn off* actions).

[8]E.g. by removing the objects observations the branching factor increases to 32. When also the places are removed and the only observations are about the user locations, the branching factor increases to 58.

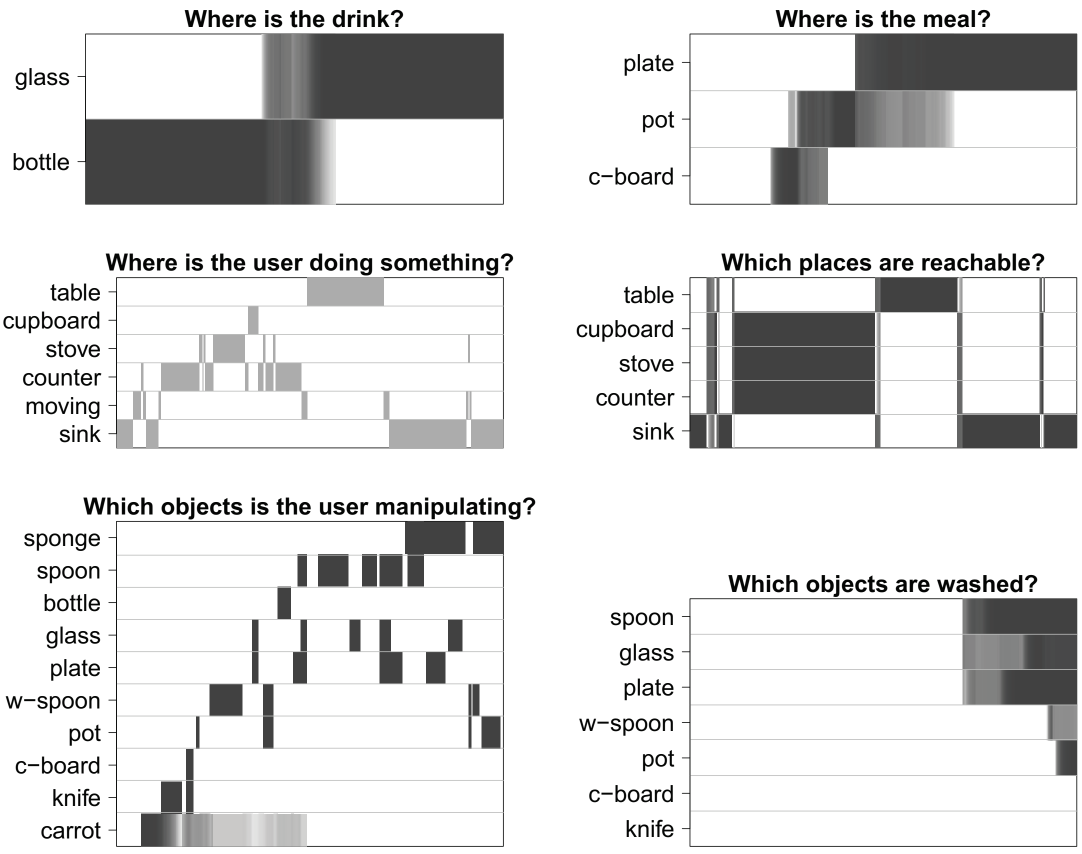


Figure 3.13: Context information that can be obtained from the model. In it, the darker the colour, the more probable the state. The left uppermost figure shows where the drink is located throughout the cooking task. The right uppermost figure shows what is the meal location throughout the experiemnt. The left figure in the middle row shows the places where the user is doing something. The right figure in the middle row shows which places are reachable with hands from the current user location. The left bottommost figure shows the objects that are being manipulated at a given time point. And the right bottommost figure shows which objects were actually washed and when.

lead form the initial to the goal state (more than 600 million). This shows that modelling a real world problem with causal models even for a single person, even with a limited number of locations and objects, leads to enormous state space. This in turn leads to the need of introducing new mechanisms for coping with the state space or for selecting the correct action. One such mechanism is the landmarks introduced by Richter [116, 115]. In it one first looks at the predicates that have to be true in the goal state and then backtracks to the initial state. Those predicates that are involved in reaching the initial state are called landmarks. Then when the model is running, it looks for these landmarks in order to decide which actions to select.

Another mechanism is a complex types system that is used to reduce the number of objects that can parameterise an action template, thus further reducing the possible model states. The structure of the model type hierarchy can be seen in Fig. 3.15 where there are 14 main types of parameters and several subtypes that inherit multiple main types. That way a given object can be used as a parameter for one action, but not for another because the two actions use different specialised subtypes (e.g. the actions *fill-from-container* and *fill-from-liquid-container*, where the objects that can be parameterised belong to the same main type *movable* but to different subtypes). Similarly, objects could be assigned multiple classes so that the given object can be used as a parameter in actions expecting different object types (e.g. the object *carrot* is

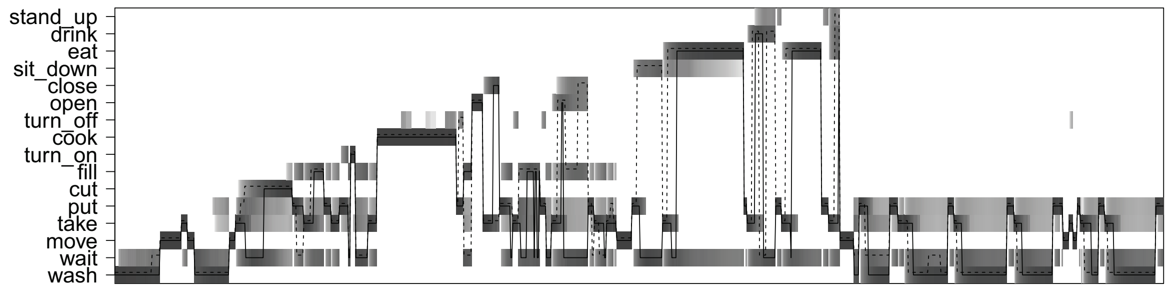


Figure 3.14: The probability distribution for the user actions. In it, the darker the colour, the more probable the action. Furthermore, the dashed line indicates the ground truth, whereas the solid line indicates the most probable action.

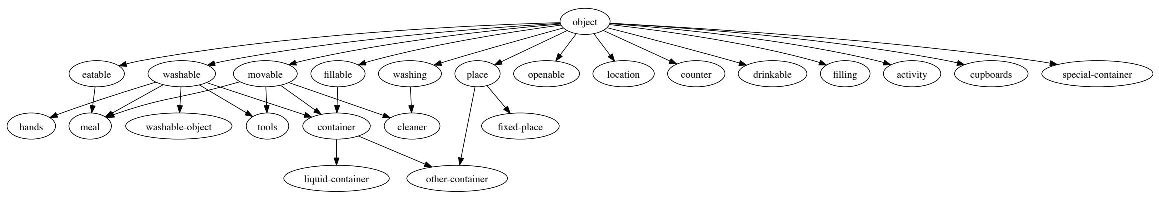


Figure 3.15: The type hierarchy for the cooking task model.

both *movable* and *eatable* allowing it to be used in *cook* and *take* although the actions require different parameter types).

3.3.2.3 Performance evaluation

To test the model, different combinations of the observations were tested. The first type of observation was the combination of objects, places and locations[9]. This observation is also equivalent to the combination of objects and places, as the places contain the information encoded in the locations observations plus some additional information. The second combination was that of objects and locations, and the third of places and locations. The latter is equivalent to observing just the places for the reasons described above. Then the model was tested with only the objects as observations, and finally, with only the locations as observations. The measurements used for evaluating the model were the same as those used in the meeting problem – accuracy, precision, and specificity. Fig. 3.16 shows the results from the performed activity recognition. It can be seen that the accuracy for the different datasets in the case of objects, places and locations being observed varies between 75% and 84%. This stands to show that in the presence of observations about the objects involved in different activities and the user fine-grained location, the model is able to reason about the user actions with a high accuracy. This is also reflected in the corresponding precision, recall and specificity values. This indicates that the model is able to correctly recognise the underlying behaviour, and to correctly identify most of the positive and negative action instances.

In the case of the objects and locations being observed, some observation information is lost compared to the first case, as the locations represent only the places which the user can reach by walking. In this situation, it can be seen that there is a small decrease in the model performance with an average accuracy and precision of about 70%. It could also be seen that

[9]As already explained, the difference between places and locations is that the locations are reachable only by walking while the places just by reaching with the hand.

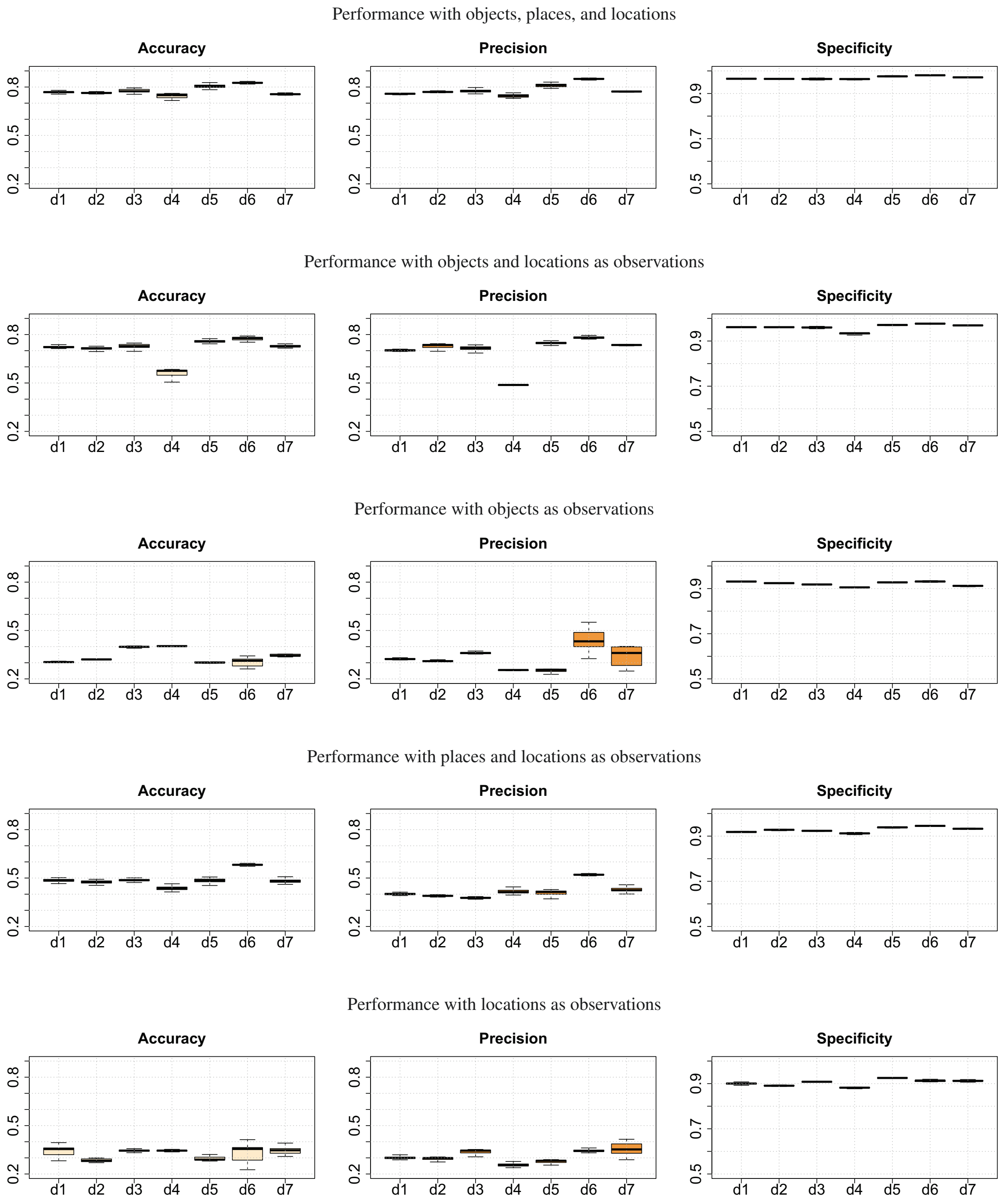


Figure 3.16: Model performance for the cooking task model. Here for each of the different observation combinations the accuracy, precision, and specificity are plotted. The x-axis represents the dataset and the y-axis the performance. For each observation type the results for 50 runs are plotted as boxes.

the lack of the additional information encoded in the places observations visibly reduced the performance of the fourth user. This could be explained by the fact that the order in which the fourth user executed the actions was atypical compared to the remaining 6 users for which the model was better suited. Thus in the case where the places were also observed, the model was able to discard the unlikely hypotheses, whereas in the latter case it found an explanation fitting better to the model but not to the actual user behaviour.

When also the locations are removed and only the objects are used as observations, it can be seen that the model performance suddenly drops to accuracy of between 30% and 40%. This is due to the fact that most of the actions can be applied on almost all of the available objects, so without an additional mechanism for improving the probability of the correct action, it is difficult for the model (combined with the particle filter inference mechanism) to find the actual execution sequence. Additionally, as the model is extremely huge, one cannot rely on the exact goal distance as a heuristic for finding the correct hypothesis. This is also evident in the obtained precision, which is similar to the results for accuracy. This indicates that the model is generally unable to represent the underlying behaviour. On the other hand the specificity still stays relatively high which indicates that the model combined with this kind of observations is still able to represent negative results.

The next case is where places and locations are used as observations. In this situation the objects are not observed and the inference engine has to reason based only on the user's current position. This significantly reduces the model performance as there are 16 possible actions and 7 of them can be executed at any place provided the preconditions for the action were met. This reflects in the model accuracy which now drops between 43% and 60% with most of the datasets performing with accuracy of about 50%. This indicates that the lack of objects being observed reduces the model performance with about 25%[10]. Still the results show that even with limited observations, the model is able to reason about the user state and to find explanation about what is being observed. One has also to consider the fact that this is by far the most complex example of user behaviour modelled with CCBM which, with its high degree of variance and number of objects involved, nears a real world example. Furthermore, in this case, similarly to the long meeting in the previous section, a dataset contains more than a thousand samples which compared to approaches like [114] represents a more realistic observation scenario.

The last combination is when only locations are observed. Not surprisingly, the model performance drops further as the information contained in the observations is even more scarce than in the previous case. This allows the model to follow hypotheses that otherwise will be pruned out because of being impossible. This in turn results in the model being able to find a causal explanation even to hypotheses that are not very probable from a human point of view. This is reflected in the low recognition rate, represented by the model accuracy – it is between 30% and 40% percent. Also the variance for the different runs is higher which indicates that in this combination the model is not robust to unwanted influences of the inference engine. The model specificity remains high but some fluctuations that were not present in the previous cases are now noticeable between the different runs.

Friedman test was performed to compare the variance of the model performance for the different datasets. The results can be seen in Table H.3 calculated separately for accuracy, precision, and specificity and in Table H.4 in Appendix H where the different performance metrics were all taken as an input for the test. The results for this model are labelled *intuitive*. It can be seen that the results are above the 0.05 p-value limit which indicates that the null hypothesis is accepted. In other words, the model performance does not vary significantly for the different datasets. This combined with the not overly high model performance points out toward the bias-variance trade-off where the ability to recognise huge set of execution sequences, results in decreasing the model performance. It can be further interpreted as the model being under-fitted thus being able to fit many hypotheses but with the disadvantage of increased bias.

[10]Compared to the first case of OPL.

3.3.3 Modelling the office scenario problem

The goal of the model is to describe the unsynchronised user behaviour in an office scenario, where one to three users act in the environment. Additionally, it aims at showing that the modelling formalism is able to reason about the users' identities and actions based only on observations about the occupancy of certain locations.

3.3.3.1 Model

The office scenario is both single- and multi-agent problem depending on the number of users present in the environment. The problem could be considered complex in terms of number of agents, but in terms of actions it is a relatively simple problem that consists only of five actions that take place. It is further simplified by the fact that agents act independently of each other and no additional action synchronisation is needed. Table 3.6 shows the actions modelled for this problem domain. They are based on the elements identified in Table 2.4 from Chapter 2. It can be seen that there are only five actions - *take* (divided into take refillable object, and

Table 3.6: Actions in the office scenario model. For each action, the predicates used in the preconditions and effects are given, as well as the parameters.

Action	**Parameters**	**Predicates**
start-walking	*?a - agent ?from ?to - position*	(at ?a ?from); (is-walking ?a ?to)
end-walking	*?a - agent?to - position*	(at ?a ?to); (is-walking ?a ?to)
start-printing	*?a - agent*	(at ?a printer); (is-there paper); (is-printing ?a)
end-printing	*?a - agent*	(at ?a printer); (is-there paper); (is-printing ?a)
take	*?a - agent ?r - refillable*	(at ?a paper-stack); (at ?a water-tab); (at ?a coffee-jar); (agent-has ?r); (holds ?r ?a)
refill	*?a - agent ?r - refillable*	(at ?a paper-stack); (at ?a water-tab); (at ?a coffee-jar); (is-there ?r); (agent-has ?r); (holds ?r ?a)
start-making-coffee	*?a - agent*	(at ?a coffee-machine); (is-there coffee); (is-there water); (is-cooking ?a)
end-making-coffee	*?a - agent*	(at ?a coffee-machine); (is-cooking ?a); (coffee-drink)
take-coffee-drink	*?a - agent*	(at ?a coffee-machine); (coffee-drink); (agent-has-coffee-drink ?a)

take coffee drink), *walk* (divided into begin-end pair), *print* (also divided into begin-end pair), and *make-coffee* (divided into begin-end pair).

Additionally, Table 3.7 shows the model parameters for the different number of users. It can be seen that there are 50 operators and 28 predicates in the single user case, and that they increase times the number of users. The goal distance also approximately doubles for each additional user as well as the branching factor. The state space and the number of valid plans also increase but they directly explode with increasing the number of agents. On the other hand the number of locations, objects and activities remain the same as the environment in which the users act does not have any new elements except for the varying number of users in it.

3.3.3.2 Model analysis

Modelling the user actions: In comparison with the cooking task, the office problem has much simpler definition of the actions. Fig. 3.17 shows the action *take-coffee-drink* where it can be seen that the action is influenced only by the available resources (whether there is a coffee-drink ready) and not by the rest of the agents. This is due to the fact that the problem contains only physical correlation and the only way actions of one user are influenced by actions of the other user is through the physical changes in the environment. Although the modelled actions are specialised, there are no constraints for synchronising based on a common goal like in the meeting problem or for limiting the state space with lock predicates like in the cooking task.

Table 3.7: Office model parameters.

Parameter	Value (1 user)	Value (2 users)	Value (3 users)	Description
# operators	50	100	150	grounded actions after the model compilation
# predicates	28	52	76	grounded predicates after the model compilation
# object types	3	3	3	see Fig. 3.20
# persons	1	2	3	objects of type agent
# locations	7	7	7	objects of type position
# objects	3	3	3	objects of type refillable
# activities	5	5	5	activities to be estimated
# states	21 504	9 633 792	> 880 000 000	state-space of the model
# valid plans	43329	48 373 761	>48 373 761	valid plans leading from the initial to the goal state
# hierarchy level	2	2	2	levels of the type hierarchy
# goal distance	14	33	> 33	minimum distance from the initial to the goal state
# max. branching factor	6	16	> 16	maximum number of possible actions at a given time

This makes the modelling of the actions straight forward and without further regards about the user influences on each other. On the other hand it increases the state space exponentially, as there are no mechanisms for reducing the predicates being grounded.

```
(:action take-coffee-drink
  :parameters (?a - agent)
  :saliency 1.2
  :agent ?a
  :duration (normal take-coffee) ; time to take a cup of coffee
  :precondition (and (at ?a coffee-machine)
             (coffee-drink)
             (not (agent-has-coffee-drink ?a)))
  :effect (and (not (coffee-drink))
        (agent-has-coffee-drink ?a)
  )
  :callbacks (setAction (action-id take))

)
```

Figure 3.17: Take coffee action template for the office scenario

Available context information: The model consists of 9 actions and 6 of them are combined into *begin-end* action pairs. Although the model is not as rich as the one for the cooking task, still it can provide additional information about the users' locations, and the objects they are manipulating. It is also able to assign an appropriate action to the user regardless of the fact that only the locations are being observed. The actions *print*, *make coffee*, and *take coffee* can give information about the user executing the action, her location and, of course, the action that is happening. The action *walk* is also able to provide information about the start point of the user and her destination, while the actions *take* and *refill* give information about the object being manipulated and the agent manipulating it. Furthermore, the model predicates can answer questions such as *Does user 3 have a coffee drink?* (obtained through the predicates *(agent-has-coffee-drink)*), *Are there supplies for printing or preparing coffee?* (obtained from the predicate *(is-there)*), *Does the user have her print?* (obtained from the predicate *(agent-has)*), etc. This can be seen in Fig. 3.18 that shows the probability distributions for the answers of some of these questions, as well as in Fig. 3.19 that shows the probability of available supplies.

Model parameters: The number of grounded operators and predicates depends on the model parameters and it can be seen in Table 3.7 that the only model parameter changing is

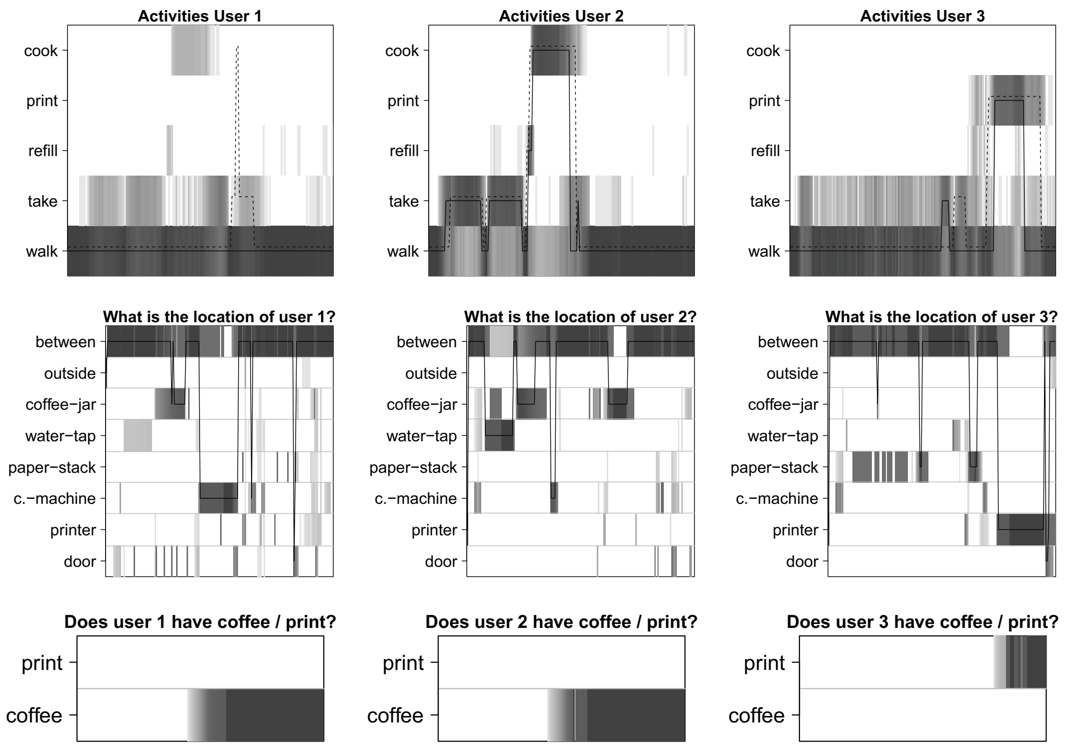


Figure 3.18: Context information that can be obtained from the model. In the figures, the darker the colour, the more probable is the state / action. The uppermost figures represent the activities that are assigned to a user at a given time. The figures in the middle row show the probability of a user location at a given time point. The bottommost figure shows what is the probability of a person having coffee or a print.

the number of users. Hence, it is not a surprise that the number of operators and predicates doubles with increasing the number of users. However, that is not the case with the model's state-space. The single-user case generates small state-space (compared to the kitchen task assessment problem). Yet, this changes when the problem becomes a multi-agent one where two or three agents are acting in parallel and where there are no synchronisation constraints to reduce the number of possible states. This can be seen in Table 3.7 where while the single-user model has only 21 504 states, for the two- and three-user cases it suddenly explodes (9 633 792 states for the two-users case and more than 880 million states for the three-users case). This is also reflected in the increasing number of valid plans, as well as the increasing goal distance, and increasing branching factor[11]. This indicates that there is need of adequate lock mechanisms for controlling the exponentially increasing state space with increasing the number of users (or at least, in the case where the goal distance is used as an action selection heuristic.)

[11]Unfortunately the computational power needed for calculating the 3-users case was not available and the analyser crashed after using the whole 120GB of RAM available without being able to fully explore the state space and to calculate the goal distance and the number of valid plans.

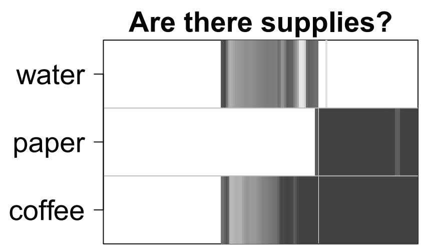


Figure 3.19: Probability of available supplies throughout the experiment.

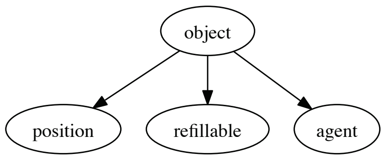


Figure 3.20: The type hierarchy for the office scenario model

Furthermore, it can be seen from Fig. 3.20 that the model has a simple type system with only three object types (position, refillable, and agent) which also does not contribute for the state space reduction. Of course, in difference with the cooking task, in this problem there are not many objects to be manipulated, so there is also no need of a complex hierarchy.

3.3.3.3 Performance evaluation

To evaluate the model, it was compiled into runtime model that used observations provided by the SensFloor to recognise the activities being executed and the agents associated with them. The observations had a binary value with 1 for person *sighting* at a given location and 0 for *no sighting* (for more information about the test datasets, see Chapter 2). Fig. 3.21 shows

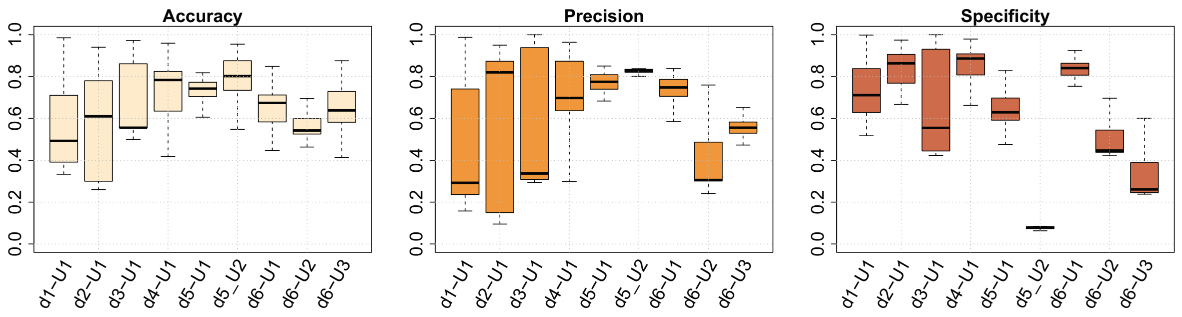


Figure 3.21: Model performance for the office scenario model. As there is a varying number of users, the performance of each user is plotted separately. The results for each user from 50 runs are plotted as boxes. The x-axis shows each user and the corresponding dataset, while the y-axis shows the performance.

the results from the evaluation. It can be seen that the first user for the first dataset performed surprisingly poorly and had an average accuracy of about 45% which is explained with incorrect action duration values which caused the model to be extremely dependent on the particle filter random seed. This resulted in the assignment of a higher probability to the action *walking*, when it should have been print.

On the other hand, the remaining datasets showed better recognition rate (with average above 60%), which is a surprise especially for the multi-agent cases in experiments 5 and 6. In these two cases the inference engine had to decide not only what action is executed but also who is executing it. It can also be seen that these two experiments showed a smaller variance as compared to the single user cases. However, this is explained by the fact that the multi-agent cases lasted longer and in more than 50% of the time the users were walking which the model showed to be good at recognising.

Additionally, the model's high degree of freedom made it dependent on the inference mechanism, thus causing deviation of about almost 90% in the precision values of the different runs. The same phenomenon is observed also for the specificity of the model. Furthermore, in dataset 5, the behaviour of the second user, although having relatively high accuracy rate, has a specificity rate of under 20% which indicates that above 80% of the negative instances were incorrectly assigned to an action class.

The results thus indicate that although the model is able to provide additional information about the given situation, it is still far from good. The high degree of freedom combined with incorrectly assigned actions' durations leads to the model dependence on the filter random seed. It also makes actions probable, that in reality would not be executed by the user, and that could be easily omitted by the introduction of some lock predicates.

Friedman test was applied in order to test the results variance for the different datasets. As the users per dataset vary and act unsynchronised, the results from each user were taken as different dataset. The results can be seen in Table H.4 in Appendix H. They show that the p-value is above the 0.05 threshold, which indicates that there is no significant difference in the model performance for the different users. This can be explained by the fact that the model is relatively general and there are no many constraints that reduce the behaviour variability. This results in increasing the model's ability to recognise variations in the behaviour but on the other hand, decreasing it ability to recognise the correct action which is reflected in the model performance. This once again points at the bias-variance trade-off where the decreasing variance results in increasing bias.

3.4 Discussion on modelling with Computational Causal Behaviour Models

In this chapter the models for the three problems were presented and discussed in terms of model parameters and model performance. However, the process of creating running and well performing models was one of trials and errors. Here we discuss some important issues about the experiments planning and executing, as well as such concerning the model implementation and evaluation. Furthermore, the need of structured modelling process is discussed as well as such of a collection of implementation templates.

3.4.1 Discussion on related work

Before looking at the problems during modelling, we shortly discuss how the implemented models compare to other state of the art work in the field of model-based activity recognition. In his dissertation Giersich introduces a model-based approach to activity recognition where he uses CTT model in order to model the behaviour of a 3-person meeting [57]. The model then is used to generate a DBN that deals with the unreliability or noise in the sensor data. The approach was tested on the 20 short meetings introduced in Chapter 2. The complexity of the model is comparable to that of the team model introduced in this chapter. The CTT consists of four task nodes that generate a DBN with 9 states. The actions that are recognised are *presentA*, *presentB*, *presentC*, *wander*, and *exit*. The accuracy that Giersich reports is between 67% and 91%. Giersich's approach is comparable to the CCBM approach, in so far that is uses symbolic description of the user behaviour to generate probabilistic model with which to infer the user actions. However, one disadvantage of task trees as a choice of formalism is that they are not able to generate models with high behaviour variability, because each instantiation of an action has to be modelled manually.

Another work that deals with the meeting problems is that of Krüger et al. where the authors replace the CTT model with a PDDL representation of the user behaviour. Furthermore, they also discuss the model performance in recognising not only the team behaviour but also the single user behaviour. The resulting model has 122 operators and 48 predicates. It then uses particle filter to estimate the users' activities. The authors report an accuracy between 67% and 78% depending on the observation model used. The model that was used is comparable to the team model presented in this chapter. It however has some problems with the actions' durations as from the figures presented in the work it can be seen that the model quite often attempted to choose another action while the current action was actually still running.

Ramirez et al. also use PDDL for modelling the underlying user behaviour which is then mapped onto **P**artially **O**bservable **M**arkov **D**ecision **P**rocess (POMDP) [114]. They then use a simulated data to predict the activities and goals in three different scenarios: an office problem, that is similar to the one presented in this chapter but consists of two separate rooms, a drawers problem where the agent is looking for items in a drawer, and a kitchen problem where the agent can cook several different dishes. The office problem produced a model with 2304 states, and 23 operators. There were also 3 different goals that the agent could follow. The number of observations that were used is 15. In the drawers problem, the model has 3072 states, 16 grounded actions, 16 observations, and 3 goals that the user can follow. In the kitchen problem there are much more states: 69 120, 39 grounded actions, 32 observations, and 5 possible goals. The model performance they report is between 84% and 100% depending on the scenario and the number of observations that were removed from the observation sequence. In differences to the models presented in this chapter, the ones that Ramirez et al. presented also discussed the recognition of different goals. Still it was already shown in [81] that CCBM models are also able to recognise different goals. Furthermore, the approach proposed by Ramirez et al. uses extremely short observations sequences and does not incorporate probabilistic duration or transitions. They however, point out that the latter could be solved by employing an HMM.

A different approach to model-based activity recognition is that proposed by Hiatt et al. where an ACT-R model is used to incorporate the user behaviour [66]. In it production rules are used to reason about the context encoded in the form of model chunks. Then in order to be able to follow different hypotheses, the model is assigned different weights to retrieving facts, different knowledge about the world (initial state), different acceptable subgoals, different parameterisation of the activation equations, different parameterisation of the utility functions. Then the probabilities are calculated based on the standard ACT-R heuristics for activating a chunk or for firing a production. The implemented model contained 134 chunks that contained the facts about the world, and 49 production rules. They also had an average of 4.25 hypothetical outcomes or goals to follow. One disadvantage to this approach is that it relies on variable-free predicates representation which makes it more difficult to encode the behaviour variability compared to CCBM.

The model-based activity recognition approaches that combine symbolic representation with probabilistic reasoning are a relatively new filed of research that shows promising results. In that respect CCBM shows competitive results as it is able to easily generate behaviour variability based only on a few action templates, in difference to ACT-R and CTT which need the designer to more or less manually incorporate this variability. Another advantage is its ability to cope with probabilistic durations. The approach proposed by Giersich and that by Krüger et al. also support such durations while ACT-R and the approach proposed by Ramirez et al. do not. This allows CCBM to cope with variations in the actions' durations and to be able to incorporate the information about the duration in a single action template instead of having to assign it to every action instantiation. Furthermore, CCBM is so far the approach that has managed to cope with the longest observation sequence (62 000 observations in the meeting model) compared to the maximum of about 3000 observations in the approaches proposed by Giersich and by Krüger et al. Furthermore, CCBM produced the model with the largest state-space reported so far that was still able to produce reasonable activity recognition results (with more than 600 million states in the kitchen model, and more than 900 million states in the office scenario). Of course, the process of creating the corresponding models was a challenge in itself that consisted of many trials and errors before achieving the reported results. For that reason in the sections below we discuss the problems during modelling, the solving of which is the topic of the remaining chapters in this work.

3.4.2 Challenges during modelling

The process of developing a model is a complex one where the experimental settings, the experiment planning, the correctness of the ground truth, the model implementation, and the inference mechanism all influence the final product. Even more, the experience showed that problems in one of these components could cause unexpected and unwanted model behaviour or evaluation results. For that reason, below some of the most important issues are discussed.

3.4.2.1 Problems with the experiment planning and settings

Planning an experiment might seem a straight forward task – decide on the scenario, decide on the participants and their agenda, decide on the sensors and provide the appropriate infrastructure. However, small details such as the step by step action execution or the constraints a participant should have during the experiment should also be carefully planned. For example, during the planning of the long meeting experiment, a discussion after each presentation was planned. However, it was not explicitly explained to the participants that the discussion can take place only after a presentation and not during the presentation. Thus when the meeting started where each of them presented a real topic, some of the participants asked their questions during the presentation making it impossible to distinguish between presentation and discussion, given the sensor infrastructure. Another example is the cooking task, where the participants knew what the task in hand is and what the different phases are, but each of them executed the actions in a given phase in completely different order. That made it impossible to introduce lock mechanisms that can considerably decrease the state-space.

One could argue that such variability is expected in real world scenarios, however when the goal of the experiment is to show that a given approach works at all, there should be carefully controlled experiment settings and tasks execution.

Additionally, one should always consider problems with the sensor infrastructure, failing sensors or missing records. Before the experiment, the damage of such problems could be decreased by checking whether the sensors and the whole infrastructure are working several times before the experiment, checking the sensors batteries, or having backup sensors. After the experiment, although the damage could not be possibly repaired, one should check the collected data for missing or faulty values, as such could cause unexpected model behaviour.

3.4.2.2 Problems with the ground truth

Conducting the experiment is actually the easier part, obtaining appropriate and correct annotation could turn into a real challenge. The annotation is the ground truth with which the activities estimated by the model are compared in order to evaluate how good the model is performing. It is obtained by defining natural language descriptions of the actions that take place during the experiment, and the exact time they started and the time they lasted. The annotation is created by human annotators and usually with the help of different annotation tools e.g. [84].

To evaluate the performance of causal models, the annotation itself has to be causally correct. While for machine learning approaches it would not be a big issue if the person suddenly teleports from one place to another, for causal models that is just impossible. Additionally, publicly available datasets like those in the CMU Multi-Modal Activity Database [145] contain gaps in the annotation, annotation not matching with the activities shown on the corresponding video etc. Depending on the length of the activities short incorrectly annotated activities will also cause small deviation in the accuracy computation. However, long actions that were incorrectly annotated could cause considerable drop in the estimated model performance.

Depending on the approach used for activity recognition, gaps in the annotation, or objects and persons that just appear from one place to another without being manipulated or have moved could cause serious problems. In causal approaches like CCBM the model then is not able to causally explain given annotation. This could lead, first to decreasing the model accuracy, and second to inability to generate useful observations from the annotation. For example, in the cooking task there were some gaps in the first version of the annotation as well as objects that appeared from place to place without being moved. After generating observations from the annotation, this led to the model being unable to explain them not because the model was wrong but because the annotation was causally incorrect.

3.4.2.3 Problems with the models

Although CCBM uses straight forward precondition-effect pairs for describing the action templates, it could sometimes be a challenge to create a template reflecting the needs of the problem. This is mainly because the incorrect use of a single predicate or of a logical expression causes the model to act unexpectedly. Such problem could easily be tracked in simple models like the team model for the meeting scenario, or the office scenario. However, in a more complex model like the cooking task, it is almost impossible to find the problem without some kind of backtracking mechanism for model implementations. The cooking task model led to the need of recording the changes and the reasons for them after each model change.

Furthermore, the models were initially implemented with the idea of creating a general domain description. Yet, the practice showed that it is extremely difficult to cope with such model as it has high degree of freedom and causes problems with some of the action selection heuristics (e.g. the goal distance, which can be calculated only after the whole state space is analysed).

Another issue was caused by the actions durations. Choosing appropriate duration and the corresponding probability distribution could be crucial for the correctness of the inferred action. Actions with too short or long durations tend either to cause the model to select another action although in reality the current action is still running, or to continue estimating the same action although it has already ended.

However the opposite – assigning too exact duration, or allowing too small behaviour variations, leads to model overfitting and its inability to cope with new data from the given scenario. For example, in the cooking task scenario, it was necessary to find the middle ground between the high state-space and the ability to explain new data. Another example showing overfitting, was the office dataset where for each action and user relatively exact duration was assigned. This would probably yield good results for more restricted model, however the high degree of freedom caused the wrong duration to be the one that is usually assigned.

A serious issue during the model development and evaluation was related to traceability issues. It was extremely difficult to discover the reasons behind past changes as there was no substantial documentation about these changes. Although the model changes were under version control, detailed information about the reasons behind these changes was missing. The same problem was observed during attempts to reproduce given results. This was due to the fact that the random seed generator uses the current time for generating values; due to different inference parameters that were not documented; due to undocumented changes in the model after the first results were obtained, etc. To avoid that, at a later point, a script was introduced generating a documentation file. The file then provides information about all parameters and scripts used for producing given results. The script also created copies of all scripts and files involved in the inference and evaluation process.

3.4.2.4 The influence of the inference mechanism on the model behaviour

As the particle filter provides an approximation of the estimated state, it is dependent on the random number (called random seed) used for sampling the state-space. In the particle filter at each time step the state space is approximated with a weighted set of samples (particles) where the weight of each particle is proportional to the particle's probability. This however leads to the problem that at some point many of the samples have very low weights and only a few of them will have significant weight. Or with other words, the number of effective particles drops significantly with time. To cope with this problem and increase the number of effective particles, resampling is performed. During resampling, a new set of the original size of particles is drawn with replacement from the discrete approximation of the samples distribution. In this new set all particles have the same weight. However, this still does not solve the problem that particles with larger weights will be drawn more often than such with low weights. This means that after resampling the diversity of the particles will decrease. To solve this new problem, random numbers are used in order to determine which particles to be drawn. For more details on the particle filter and resampling see e.g. [68].

It is then possible that the random seed influences the system in a different way. For example, it is possible that the correct hypothesis is not available because it was not sampled. It is also possible that the particle representing the correct state has too low weight thus it was never selected. This will result in the wrong state being estimated. Another possibility is that there were not enough particles to represent the density distribution. It is especially true in problems with large state-spaces like the cooking problem where the state-space is several hundred thousand states.

Additional problem could be caused by the action selection heuristics. For example, a model using the goal distance as a heuristic will have problems with a subject that is not acting in an obviously goal oriented manner. Each of the action selection heuristics, when inappropriately used, can lead to the wrong hypothesis being selected.

Furthermore, it is possible that the actions were assigned inappropriate probability distribution, or the chosen values did not represent the reality. This could result in either the action terminating sooner than in reality, or that the inference engine believes the action is still being executed even when in reality new action has started.

The above issues do not mean that the causal model is incorrect, they just indicate that the combination of causal modelling and probabilistic inference can lead to unwanted consequences. What can be done in this case on a causal level, is to introduce some artificial constraints that do not improve the system model, but provide a mechanism for coping with problems on the inference level. Another option would be more appropriate usage of the action selection heuristics, a better random number generator algorithm, and of course, careful decision on the probability distribution describing the action duration.

3.4.3 The need of structured modelling process

The above problems pointed at the need of a structured modelling process. Such process should be able to provide not only well performing models, but also documentation about them, about history of model changes and the reasons behind them. It should also give information about the model execution and evaluation that yields reproducible results. The experience showed that without such process, the task of modelling human behaviour could be difficult and result in unexplainable model dynamics. It also showed that especially in the case of complex models like the cooking task, the lack of such process and the corresponding documentation could lead to months of tracking and debugging implementation problems.

Apart from improving the model documentation, the main goal of such process would be to provide guidelines for a structured way in which an activity recognition model can be developed and evaluated. Such process in difference to the state of the art waterfall model [120] should include not only the model development itself but also the activity recognition lifecycle including also experimental setup, data collection, data analysis and model performance evaluation. These are steps that are typical for data analysis problems and empirical scientific experiments but not for software engineering problems.

The experience showed that it is easy to get lost during the model development. It is also easy to spend enormous amount of time in implementing a successful model, when it could be much faster were there some existing guidelines about the process. This could be considered as a drawback of any new and not well explored approach, but it is also a solid motivation for providing such development process.

3.4.4 Identifying Development Phases

From Chapter 2 and Chapter 3 several phases could be identified. The modelling and activity recognition process was intuitively divided into such phases to help the model designers in implementing appropriate models.

Problem analysis: This phase involved the identification of the problem; identification of important environment and situation elements that will later play crucial role in the model; discussion about possible solutions; and the supply of appropriate ground truth (annotation). In our experience, such phase is essential and one should spend enough time on the problem analysis. Although we instinctively had this preparation phase, the experience showed that not well identified problems and objectives could lead to difficulties during the model implementation and evaluation that is much more difficult to fix when the model is already implemented.

Model implementation: This phase concerns the incremental model implementation, debugging and validation. It consists of implementing the different aspects of the model: its causal structure, the action selection heuristics, and the action durations. This is the phase where a typical data analyst focuses her efforts and we made the experience that sometimes that leads to surfacing of problems caused by not carefully planning of the model solution in the earlier phase. Additionally, the lack of detailed documentation in the model changes, or the reasons for some solutions and changes made it hard to backtrack and fix problems.

Model evaluation: This phase is the most important result of the modelling process from an activity recognition point of view, or namely, how well does the model recognise the user actions. In difference with the various simulation and software engineering development processes, where a validation of the model correctness could be enough to make a statement, in the activity recognition case, often a correct model does not necessarily indicate high performance.

It can be seen that these phases actually reflect the waterfall model, where the development process is divided into analysis, design, implementation, and evaluation. Thus it could be concluded that a human behaviour modelling development process will be based on these typical software engineering phases. On the other hand, the process behind developing models for activity recognition is a data analysis process and the experience showed that it is difficult to fit typical data analysis procedures in a software development process. For that reason Chapter 5 investigates different existing development processes and together with the experiences gathered from this chapter proposes a structured development process for Computational Causal Behaviour Models.

3.5 Outlook

The chapter presented a detailed introduction to modelling with Computational Causal Behaviour Models, and showed that it is possible to model the three problems with this formalism. The resulting models were able to recognise the activities of the users; and in the case of a small enough model to generate an HMM, showed that a Hidden Markov Model generated from CCBM performs comparable to a hand crafted HMM. Along with the ability to recognise activities, the models were also able to provide additional context information about the users' whereabouts. The chapter also analysed the models in order to discover successful practices and pitfalls. The resulting analysis is later used as a basis for the modelling toolkit, presented in Chapter 4.

In general, the chapter presented the intuitive data analysis approach to solving model based activity recognition problems, and showed that the selected formalism is suitable for the modelling domain. Still, it was concluded that a more structured approach could potentially improve the modelling efforts and results.

Chapter 4

Modelling Toolkit for Computational Causal Behaviour Models

"Any intelligent fool can make things bigger, more complex, and more violent. It takes a touch of genius and a lot of courage to move in the opposite direction."
Ernst Friedrich Schumacher

Chapter Summary: *This chapter describes the CCBM modelling toolkit that contains modelling solutions (respectively patterns) for solving problems from the activities of daily living that are based on the modelling experiences from Chapter 3. Their aim is to provide mechanisms for reducing the model complexity in terms of state-space size, branching factor, number of operators, valid plans, as these model aspects influence the correct action selection. Furthermore, some relevant patterns are practically applied to the models from Chapter 3 and their performance is compared.*

Chapter Sources: *This chapter is partly based on the paper "Strategies for Modelling Human Behaviour for Activity Recognition with Precondition-Effect Rules" [161]*

Questions to be answered in the chapter:

Why is there need of modelling mechanisms for CCBM? (In Section 4.1)

What kinds of CCBM modelling mechanisms are presently available, how can they be used and why? (From Section 4.2 to Section 4.9)

Can the models from Section 3.3 be improved by applying some modelling patterns? (In Section 4.10)

4.1 Introduction and Motivation

The previous chapter already introduced the intuitive models for the three ADL use cases. During the models analysis and evaluation it became apparent that the successful modelling of the problems is not a trivial task. Even more, as CCBM is a probabilistic approach that relies on the goal distance for selecting the correct action, it was shown that models with high degree of freedom and huge state space tend to assign higher probability to incorrect actions. Even more, this makes the model more dependent on the underlying inference engine. For example, changing the random seed used in the different model runs creates larger deviation from the average performance value (as shown in the multi-agent model performance (Fig. 3.10), and in the office scenario model (Fig. 3.21)). This is an indication that the model is not overly robust and that its more probable execution paths do not match the actual activities.

One way of increasing the model robustness is the integration of modelling mechanisms that reduce the model size and its degree of freedom. Such mechanisms are handled by the different locks that allow a given action to be executed only if a certain constraint is met. The discovery of such mechanisms is done during the model implementation, when it is noticed that they are helpful for improving the model performance.

Another reason for the need of such modelling toolkit is that some modelling mechanisms are not straight forward to model in CCBM. For example, the idea of introducing lock predicates is clear, but on the other hand implementing a repeating action that can be executed only a given number of times could be tricky. That is due to the fact that CCBM does not provide an explicit mechanism for counting.

Furthermore, looking from the viewpoint of the requirements for human behaviour modelling from Chapter 2.4, some of them are straight forward to implement, but other such as the requirement for interleaving actions could be more complicated. This statement is reinforced by the conducted user study that among other things posed the question of *How feasible is a requirement?* (see Fig. C.5 in Appendix C.3). There it can be seen that requirements such as *interleaving* and *suspending* are scored as moderate to difficult to implement.

For the reasons above, this chapter extracts modelling mechanisms that were used throughout the modelling of the three problems, and discusses their effect on the model dimensions and performance in a more controlled manner. For each modelling mechanism, its structure as well as a sample implementation code are provided. Later, the consequences it has on the model parameters and performance are analysed by providing evaluating the consequences a pattern has on a simple model. The pattern influence on a general model are then discussed. Finally, the pattern applicability is discussed.

To evaluate the pattern consequences on the model, the model parameters as introduced in the previous chapter are analysed. These are

- the number of operators that tells us how many grounded actions are there in the model, or with other words how many different instantiations of the actions are available;
- the number of predicates that tells us how many grounded predicates are there in the model;
- the objects that are used for grounding these actions and predicates;
- the number of states in the model that gives information about how complex the model is. They are computed based on the number of different combinations the predicates in the model can provide;
- the number of valid plans that lead from the initial to the goal state, that indicates the model ability to explain different behaviour variations. They are computed based on the number of different combinations the actions in the model can provide;

- the shortest goal distance from the initial to the goal state. It is computed based on the minimum number of actions that have to be executed in order to achieve the goal. As all considered problems concern goal oriented behaviour, the goal distance provides a heuristic for choosing the next action based on its distance to the goal[1].
- the maximum branching factor that gives information about the maximum number of actions that can be executed from a given state;
- and finally, the final log likelihood is provided that gives the probability of the evaluated plan. It is calculated by the formula `log` $p(y_{1:t}|M)$ where M indicates the model being used and y the observation from time step one to t. Or in other words, the likelihood gives us the information how good our model fits to the observations. In that manner the system can follow different goals and just compare their likelihoods in order to discover the most probable one.

Based on these parameters, and in order to discuss their behaviour in a general case, here we formally define the notion of model.

Definition 27. ***(Model)*** *A model B is a tuple $B := (Pl_B, Op_B, Pr_B, Ob_B, S_B, G_B, Br_B)$, where Pl_B represents the number of valid plans the behaviour has; Op_B is the number of operators; Pr_B is the number of grounded predicates; Ob_B is the number of objects available; S_B is the number of states; G_B is the goal distance; and Br_B is the maximum branching factor.*

Furthermore, the model performance is evaluated based on the model's ability to correctly predict an action. For that, three metrics are considered:

- the model branching factor after each action execution;
- the model entropy after each action execution, that gives us the model variability at a given state;
- and the normalised action probability that gives the information about how probable is the action correct action.

4.2 Locks

Motivation: Before discussing the different modelling patterns, here we present the basic mechanism behind each pattern – namely the lock predicates that are used as constraints in the model. The fastest way of reducing the model complexity is by introducing additional locks that allow certain actions to be executed only after some conditions are met. These could be for example that the person doing the action has to be at a certain position or that the object being manipulated is located at a specific place. It could also be the case that an action can or respectively cannot be executed when another person is executing a given action. In that sense, locks are the base of all other more complex modelling mechanisms described in this chapter. They can be thought of as similar to the semaphores in operating systems that control the access of multiple processes to a shared resource [143, p. 78]. More precisely they are similar to the mutexes in operating systems which are a special kind of semaphores that can

[1]The goal distance is also applicable in situations where there is no common goal, but rather a set of competitive goals (e.g. the office scenario). In that case the agents still share the same state-space, thus the goal state will be the one in which the goals of all agents are achieved.

have only two states: locked and unlocked [143, p. 81]. In that sense the locks used here allow or block the access to a certain resource in the shared environment when more than one agents act in the environment, or when the execution of different actions has influence on the shared environment.

Structure and implementation: Locks are implemented in the action's preconditions and effects, where they are defined as any other predicate and later parameterised with the available objects. The lock takes care of whether the object being manipulated can be manipulated at the given state, or whether the user is allowed to be at the location she is, or even if she can execute the action or not. Fig. 4.1 shows an example structure of an action containing locks.

```
(:action A
  :precondition (and
    (lock 1)(lock 2)...(lock n)
    )
  :effect (and
    (not (lock 1))(not (lock 2))...(not (lock n))(lock m)
    )
)
```

Figure 4.1: Structure of an action with lock predicates. In this example in the precondition of the action A n lock predicates are expected to be true. Later in the effect these locks are set to false and another lock predicate m is set to true.

Examples of lock predicates can be seen in the sample code in Appendix E. For example, in Fig. E.2 on page 220, such predicate is *(counting ?i ?j)* that locks an action to be executed only a certain number of times. Another example is in Fig. E.1 on page 219, where the predicate *(allowed-repeat)* defines whether an action can be repeated or not. Such examples can be found in the implementation of all patterns as they are basically implemented and managed through various locks.

Consequences: The consequences that a lock has on the model is basically that it reduces the state space. This can be seen in Table 4.3 in the explicit case where lock predicates for the order of the atomic actions execution was introduced. The same applies for Table 4.6 on page 116 where special lock predicates were used to combine two objects and which also defined which objects can be combined together. The locks are essentially used in the phases definition and their influence on the state space can be seen in Table 4.7 on page 122. They are also able to reduce the branching factor, the number of valid plans and the number of operators. This is usually to the expense of an increasing number of predicates, as the additional constraints are introduced in the form of predicates.

Applicability: As already mentioned, locks are the main mechanism for building more complex modelling mechanisms for reducing the model complexity. For that reason they are applied to every pattern that is discussed below.

4.3 Abstract reusable actions

Motivation: Writing readable and error-free code often depends on the length of the code and its complexity [37]. It is much easier to write simple code and later it is also easier to discover problems in such code compared to some more complex implementation. Furthermore, complex code is difficult to reuse as it is specialised for a given problem and needs changes to be done before being applicable for different kinds of problems. In such cases abstract reusable actions can be used. They reduce the code needed for describing the different action variations and are easily reusable due to their general implementation. Such actions can be applicable when one type of action is applied to situations, such as moving from or to different places or manipulating different objects. Here we discuss how such actions are implemented and how they affect the model compared to specialised actions.

Structure and implementation: An abstract action is implemented by defining one action for all variations of a given situation with preconditions and effects that allow the action to be grounded with the parameters for all variations. This is equivalent to implementing several specialised actions, each of them representing just one situation and parameterisable with the elements for just this situation. Fig. 4.2 shows the structure of an abstract action that can be

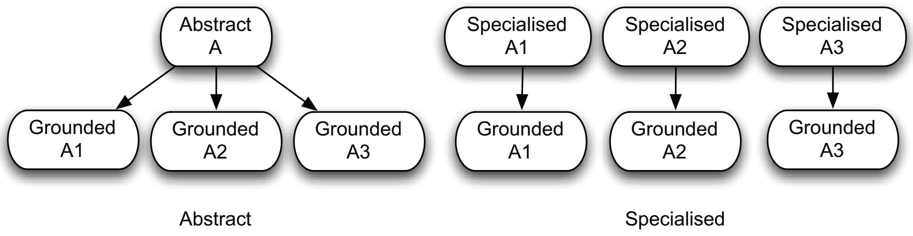


Figure 4.2: Structure of abstract and specialised action implementation. To the left the abstract action implementation is shown where one action template is grounded into three action instances. To the right, three specialised action templates are grounded in the same three instances as in the first case.

grounded into three specialised actions (to the left), and which is equivalent to three specialised actions' descriptions which after parameterisation result in the same grounded actions as the first case. The advantage in this case is that the implementation of the abstract description is usually shorter, as only one action is defined. This can be seen in Appendix E where Fig. E.3 on page 221 shows an example implementation of an abstract action, while Fig. E.4 (page 222) shows the implementation of the same problem with specialised actions. It can be seen that the specialised implementation requires the definition of three separate actions, while in the abstract implementation, only one action is needed. Additionally, the specialised actions need to make use of the type system. Otherwise each action description will be grounded with all available constants resulting in 15 grounded actions instead of only 5 in the current case.

Consequences: Although the two implementations of the problem have different structure,

Table 4.1: Parameters of abstract action implementation. The values for the abstract model are calculated based on the model implementation in Fig. E.3 on page 221 while, those for the specialised are based on the implementation of specialised actions in Fig. E.4 on page 222.

Parameter	Value (abstract)	Value (specialised)	Description
# operators	5	5	grounded actions after the model compilation
# predicates	5	5	grounded predicates after the model compilation
# objects	5	5	objects in the model
# states	32	32	state-space of the model
# valid plans	120	120	valid plans leading from the initial to the goal state
goal distance	5	5	distance from the initial to the goal state
max. branching factor	5	5	maximum number of possible actions from a state
final log likelihood	-4.78749	-4.78749	final log likelihood of the execution sequence

they have the same consequences to the model. Table 4.1 shows the parameters for the abstract and for the specialised implementations of the problem. It can be seen that regardless of the different structure, both implementations result in the same number of grounded operators (5), objects (5), states (32), valid plans (120), have the same branching factor and the same final likelihood. This indicates that the simpler and more abstract implementation can replace the specialised one without the danger of changing the resulting runtime model. In the normal case when the abstract action is shorter than a corresponding specialised one, it also makes it easier to discover problems in the implementation.

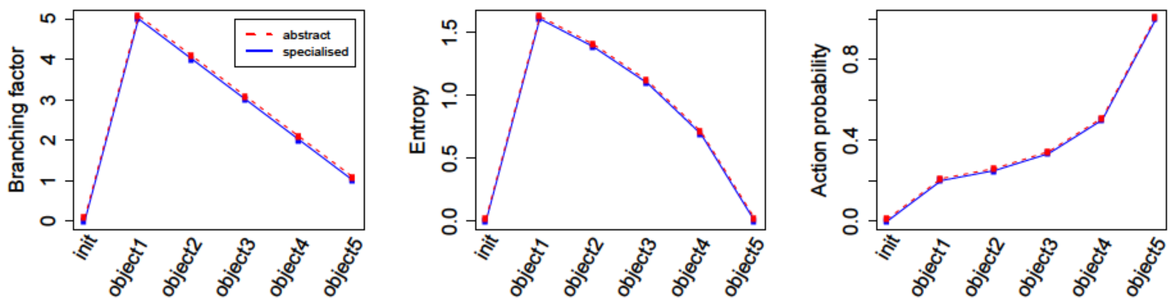


Figure 4.3: Model dynamics for abstract vs. specific actions. The red dashed line represents the behaviour of abstract actions, while the blue solid line – of specialised actions. The red line is slightly shifted for better visualisation.

This is also supported by the resulting model dynamics. Fig. 4.3 shows the abstract model behaviour (with blue line) versus the specialised (with red dashed line). To compare their behaviour a plan was created where the five available objects were manipulated sequentially. It can be seen that both implementations exhibit exactly the same behaviour. The branching factor decreases with each action as there are less objects available that were not manipulated[2]; the same applies for the entropy, while the actions probability increases with the decreasing entropy; and the probability of the normalised actions' sequence decreases with each executed action. This shows that there is no difference in the performance of the two implementations.

General case: Above we showed with a dummy example that the abstract implementation has the same model dynamics as one using several specialised actions. Here we assume a general case and show the conclusions are still valid.

Proposition: *A model that contains abstract action implementation will have the same dynamics as a model that uses several specialised actions that correspond to the grounded abstract actions from the first model.*

Assume a model B_1 with an abstract action a which takes one parameter m from which there are n instances. This means that the action will result in n grounded instances. If we assume the worst case scenario in which each instance of action a can be executed sequentially in any order without repetition of an instance, then the number of valid execution combinations will be $n!$.

$$Pl_{B_1} = n!, \tag{4.1}$$

where the number of operators Op_{B_1} corresponds to the number of grounded actions and is

$$Op_{B_1} = n. \tag{4.2}$$

If we assume that the abstract action needs at least one predicate with which it can be represented, then when grounding the action, it will result in n grounded predicates.

$$Pr_{B_1} = n \tag{4.3}$$

[2]Here we assume that an object can be involved in an action execution only once.

If we assume that in the best case there is one object per action instance, namely the object with which the instance is grounded, then the number of objects will be n[3].

$$Ob_{B_1} = n. \tag{4.4}$$

The length of the goal distance will be equal to the length of the shortest execution plan, namely

$$G_{B_1} = n. \tag{4.5}$$

The state-space will then be computed based on the number of predicates. In this case, if we have Pr_{B_1} with n predicates, then the number of all subsets of Pr_{B_1} will give us the number of possible states , or with other words

$$S_{B_1} = 2^n. \tag{4.6}$$

The branching factor then in the worst possible case will be equal to the number of available actions at a time,

$$Br_{B_1} = n. \tag{4.7}$$

Now assume there is a second model B_2 with actions a_m, where $m := \{1, ..., n\}$ and where to each action a different object is assigned as a parameter. Assuming once again that the actions can be executed sequentially in any order without repetition of any action, then the number of plans will be $n!$ as shown in Equation 4.1; the number of operators will be represented by the number of available actions, namely n (Equation 4.2). If we assume that in the best possible case, the same predicate but with a different parameter will be used in all actions, this will result in n grounded predicates (Equation 4.3). In the case when one different object is assigned to each action, then the number of objects will also be n (Equation 4.4), as well as the goal distance and the maximum branching factor. Assuming all actions have to be executed, the state-space will once again contain all subset of the set of predicates, namely 2^n (Equation 4.6). Comparing the parameters of B_1 and B_2 it can be seen that they have the same values. From this it could be concluded, that if the models have the assumptions described above, they will also exhibit the same model behaviour in both cases. The models' behaviour for increasing number of actions is visualised in Fig. 4.4.

Applicability: The abstract reusable actions can be used interchangeably with the specialised actions with the additional advantage that they take less time for implementing and produce less code. For that reason they are often preferred to the specialised actions especially in cases where there are no other modelling mechanisms that can be influenced by the abstract actions. However, in more complex models like the cooking problem, specialised actions could be preferable as they allow the control of only a given subset of the more general action in a given phase of the model. On the other hand, in situations like the meeting scenario and the office scenario the abstract actions come in handy as they reduce the model implementation effort. For that reason the implementation of the remaining patterns in this chapter is done with abstract action templates instead with specific actions, as this reduces the number of action templates that have to be modelled. This results in the need of introduction of objects that have to ground the templates in order to produce the needed number of variable-free actions.

[3]For the sake of fairness we have to admit that in the case n specialised action templates are implemented, we can achieve the same effect when we introduce n variable-free predicates, each corresponding to one specialised action. However, that means the model designer has to introduce each of the n variable-free predicates, which could be omitted in the other case, when just one predicate with variable is introduced, and which later is grounded with the corresponding object assigned to the specialised action.

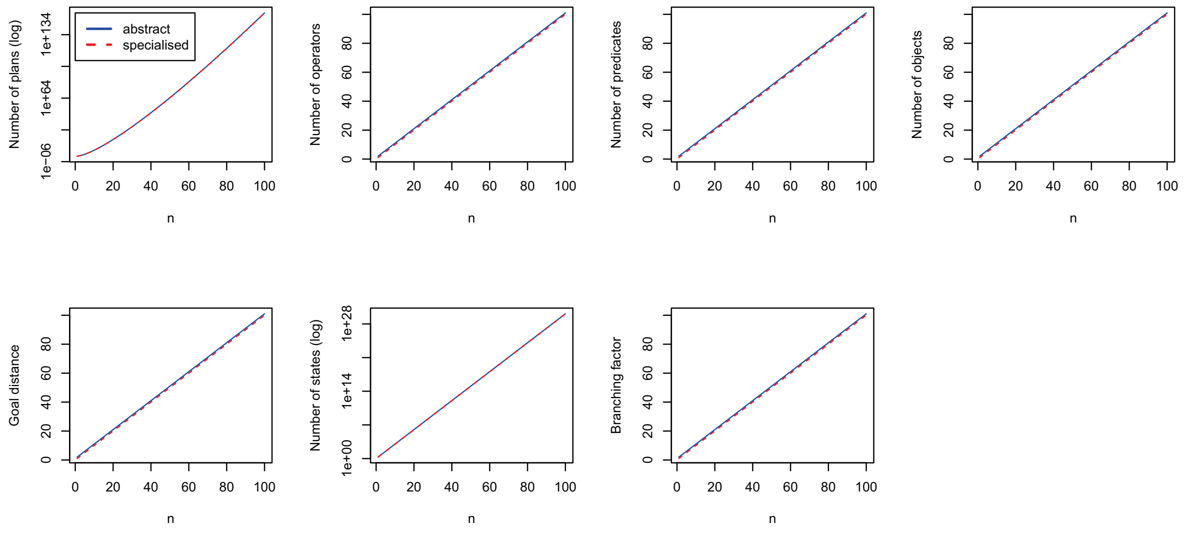


Figure 4.4: General model dynamics for abstract actions. The dashed red line represents the behaviour of specialised actions, while the blue solid line – of an abstract one. The model complexity for increasing number of actions was plotted starting with one action and increasing until 100 actions are reached. The number of states and number of plans are plotted on logarithmic scale.

4.4 Repeating behaviour

Motivation: Repeating a given action or a set of actions is something that often happens in our everyday's life. For example, repeatedly eating until we are no longer hungry, or washing the dishes several times until we are sure they are clean. Such kind of behaviour can be sometimes complicated to model with causal models as they do not have a direct mechanism for counting repeating actions. This means that either such counting mechanism has to be introduced by the designer in the model itself or that the repeating action has to be introduced by the formalism's heuristics, or more concretely – the actions' revisiting factor[4]. Here we discuss both options – using a counter and increasing the revisiting factor; and the effect both implementations have on the model. The general structure of the two options is shown in Fig. 4.5.

Structure and implementation: The first option – of using a counter – is implemented by introducing *counter* type[5] that is used for creating the needed number of constants (respectively the number of times the action will be executed). Later when implementing the repeating action, in its precondition a predicate is used that expects one of the constants to be true. If that is the case, in the effect the counter with this concrete constant is set to false and the next one is set to true. That way the model is counting how many times the action was executed until the expected number of repeats is reached. The sample code with the implementation of this approach can be found in Fig. E.2 in Appendix E (page 220). It can be seen that the action can be executed 5 times as the counter is set to count to 5.

The second option is to introduce two actions, the first having preconditions allowing the action to be repeated indefinitely, while the second contains the effects that have to take place after the last instance of the action is executed. This implementation is simpler than the first one,

[4]More information about the revisiting factor is provided in Chapter 3 in the discussion of Formula 3.1 (page 55).

[5]In reality the name of the type is irrelevant. Here it is called *counter* just for simplicity.

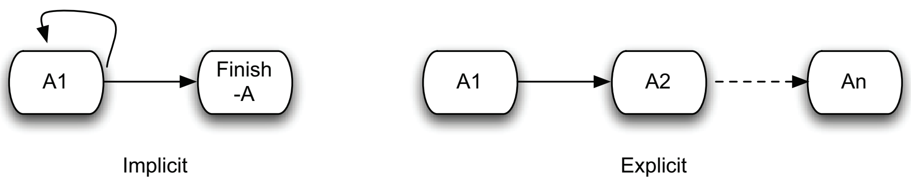


Figure 4.5: Structure of repeating actions implementation. To the left – an implicit implementation where the same action is repeated until the action *finish* is executed. To the right, the explicit implementation is shown, where each instantiation of the action is explicitly modelled.

as the need of explicitly encoding the number of times the action can be repeated is omitted. A sample code with the implementation of this approach can be found in Fig. E.1 in Appendix E (page 219). Here in difference with the explicit implementation, there is no counter calculating how many times the action can take place. It can continue indefinitely, or until the action *repeat-finish* has taken place. The action *repeat* has as precondition *allowed-repeat* which is set to true until the action *repeat-finish* has taken place. This mechanism however, is working only with the revisiting factor that allows exploring states that were already visited.

Consequences: The two different implementations of the repeating behaviour have different consequences to the model. After compiling, they have different number of grounded operators and predicates and they influence the model dynamics in a different way. Table 4.2

Table 4.2: Parameters of repeating behaviour implementation. The values for the explicit repetition are based on the implementation provided in E.2 on page 220, while those for the implicit are based on the implementation in Fig. E.1 on page 219.

Parameter	Value (explicit)	Value (implicit)	Description
# operators	5	2	grounded actions after the model compilation
# predicates	6	3	grounded predicates after the model compilation
# objects	3	0	objects in the model
# states	6	3	state-space of the model
# valid plans	1	1	valid plans leading from the initial to the goal state
goal distance	5	2	distance from the initial to the goal state
max. branching factor	1	2	maximum number of possible actions from a state
final log likelihood	0	-2.77259	final log likelihood of the execution sequence

shows the parameters for the two implementations. The first column with values contains those for the explicit implementation with counters, whereas the second column – the parameters for the non-explicit model. It can be seen that the first one has more grounded operators. This is due to the fact that the actions are grounded against all instances of the predicate *(counter ?i ?j)* that are allowed in the precondition. On the other hand the implicit implementation does not have this mechanism for counting the instances, but rather allows the action execution as long as another action has not taken place and set the precondition of the repeating action to false. The first implementation also uses objects while the second avoids that. The number of operators and predicates for the first implementation also increases the state-space compared to the second one (6 states in the first against 3 in the second), and for that reason also increases the goal distance [6]. That however, does not mean that the second implementation is better as it

[6]The values given here are for the shortest possible execution path given the model definition. This means that the implicit model has 2 states and 1 valid plan, given the model ends after the repeating action is executed only once. If we however calculate the values for five repetitions (as explicitly modelled in the first model), then the results look different: 26 states and 24 valid plans which is much more than for the explicit model.

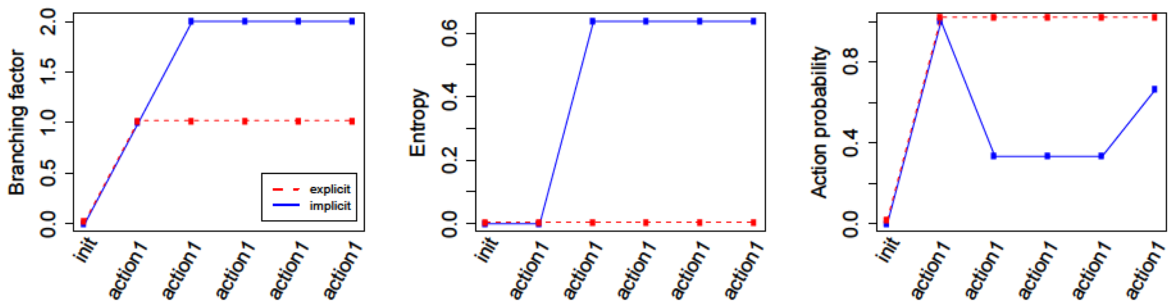


Figure 4.6: Model dynamics for repeating behaviour. The dashed red line represents the behaviour of explicitly modelled repeating action, while the blue solid line – of an implicitly modelled action.

has the danger of entering infinite loop where the action is repeated indefinitely. The implicit implementation, on the other hand, increases the branching factor. In a real model that would mean there is always one additional action that can be executed.

Looking at the parameters of the two implementations it is obvious that the second is easier (in terms of need of counting the actions) and generates smaller model. So one could ask why ever use the explicit implementation? The answer is that during the inference phase, the two implementations also affect the model dynamics in a different way. To visualise that, a plan was created that consists of 5 times repeatedly executing the action repeat and the results from the validation of the two implementations is showed in Fig. 4.6. Here the first action is the *INITIALIZE* action, while the remaining 5 are the repeated action. It can be seen that for the explicit implementation, all factors stay constant — there is a branching factor of 1 indicating that there is always only one action that can be chosen from a given state. Because of that, the entropy is zero, and the current action's probability as well as that of the execution sequence are all one. This indicates that using such action implementation will assign higher probability to the actions during the inference. On the other hand, the implicit implementation increases the branching factor as now after the first action is executed, there are two possible actions – *repeat* and *repeat-finish*. This increases the entropy of the model and decreases the actions' probability and the probability of the execution sequence. It can be seen that the more the action is repeated, the lower the probability of the execution sequence (the final log likelihood in Table 4.2).

General case: Above we showed in a dummy example that the explicitly modelled repeating actions have smaller branching factor compared to the implicitly modelled repetitions. On the other hand the implicit actions require less parameters to be modelled and have shorter minimum goal distance, less operators and predicates. To prove that this behaviour will be also valid for a general model we make the following proposition.

Proposition: *A model B_1 that contains explicit implementation of repeating actions will have predicates, operators, objects, states and goal distance that increase linearly with increasing the number of actions. The branching factor and number of plans of B_1 will stay constant. An implicit model B_2, on the other hand will have linearly increasing number of plans and states, while its predicates, operators, objects, minimal goal distance and branching factor will stay constant.*

Assume a model B_1 with explicitly repeating action a_m where $m := \{1, .., n\}$ is the number of times the action a can be executed. We assume that each instantiation of the action a can be executed only once, and that an action a_j with $j := \{2, ..., n\}$ can be executed only and exactly after the action a_{j-1} was executed. Then the number of valid execution combinations will be 1. From the above follows that the number of valid plans Pl_{B_1} such behaviour will have is

$$Pl_{B_1} = 1, \tag{4.8}$$

where the number of operators Op_{B_1} corresponds to the number of grounded actions and is

$$Op_{B_1} = n. \tag{4.9}$$

Furthermore, if we assume that we have n operators which can be executed only once each, then the minimal number of grounded predicates will be $n+1$. This is the minimal number needed to represent the increasing counter.

$$Pr_{B_1} = n+1. \tag{4.10}$$

If we assume that in the best case there is one object per action instance (namely the counter object), then the number of objects will be

$$Ob_{B_1} = n, \tag{4.11}$$

and the length of the goal distance will be equal to the length of the shortest execution plan, namely

$$G_{B_1} = n, \tag{4.12}$$

Furthermore, assuming that at least n actions are needed to reach the goal, then the states in an execution path will be $n+1$. In this case they reflect the number of predicates, as only one predicate can be true in a given state.

$$S_{B_1} = n+1. \tag{4.13}$$

The branching factor then in the worst possible case will be equal to the number of available actions at a time, namely

$$Br_{B_1} = 1. \tag{4.14}$$

Now assume there is second model that uses implicit implementation of repetition B_2 with possible actions $\{a^*, b\}$, where $*$ indicates that the action a can be repeated indefinitely and b is the action that can terminate a. Assuming there is no mechanism to indicate when the repetition of the action has to terminate, and that the action a has to be executed at least once before b takes place, then the number of valid plans Pl_{B_2} will increase linearly with each new iteration of a.

$$Pl_{B_2} = 1, ..., \infty, \tag{4.15}$$

where 1 valid plan will represent the best possible case when the model terminates after one execution of a and ∞ the worst possible case where the model will execute a indefinitely.

The number of operators Op_{B_2} is then

$$Op_{B_2} = 2, \tag{4.16}$$

as the sequence is represented by only two actions – a and b.

The number of grounded predicates will be equal to 3 represented by one predicate that allows the execution of the repeating action, one that indicates the action is executed, and one that blocks the further execution of this action[7].

$$Pr_{B_2} = 3. \tag{4.17}$$

As in this case we do not need to track the number of action executions, the minimum number of necessary objects will be 0. This means that the non-grounded predicates will be already variable-free[8].

$$Ob_{B_2} = 0, \tag{4.18}$$

and the length of the goal distance will be equal to the length of the shortest execution plan, namely the two available actions.

$$G_{B_2} = Op_{B_2} = 2 \tag{4.19}$$

The number of states will then be equal to 3 represented by the initial state, the goal state, and the state between the two actions that have to be executed, as no matter how many times a is repeated, it will always end in the same state.

$$S_{B_2} = 3 \tag{4.20}$$

The branching factor then in the worst possible case will be equal to the number of available actions, namely,

$$Br_{B_2} = 2. \tag{4.21}$$

From Equation 4.8 and Equation 4.15 it follows that while B_1 has a constant number of plans, B_2 has infinitely many plans. From Equation 4.9 and Equation 4.16 it follows that B_1 has n operators that however cannot be repeated, while B_2 has only two operators that can however produce an execution sequence that can have as few as 2 transitions or as many as ∞. From Equations 4.10 and 4.17 it follows that B_1 will have $n+1$ grounded predicates, while B_2 will have only three. From Equation 4.11 and Equation 4.18 it follows that in B_1 there are at least n objects, each corresponding to the index assigned to the available actions. On the other hand, B_2 does not make use of the counting mechanism, thus in the best case scenario it does not have any objects. From Equation 4.12 and Equation 4.19 it follows that while B_1 has a goal distance of n, in the case of B_2 a plan can have a goal distance from 2 to infinity depending on the repetition of action a. From Equation 4.13 and Equation 4.20 it can be seen that while B_1 has $n+1$ states, the state-space of B_2 stays constant. Finally, the branching factor of B_1 is equal to 1 (Equation 4.14), while in the case of B_2 it is equal to 2 (Equation 4.21). The above is reflected in Fig. 4.7 which visualises the dynamics of explicit and implicit repetitions for increasing number of repetitions.

Applicability: Based on both implementations and their influence on the model dynamics, each of them can be used in different situations. The explicit implementation is more time consuming as one has to implement the counting mechanism. On the other hand, it ensures

[7]From the viewpoint of scheduling we need only two predicates – one for allowing the action execution and one for blocking it. However from the perspective of the causal structure we need another predicate indicating the action is executed as one execution could be needed to unlock another action execution and still allow the repeating action to be executable.

[8]In difference with the abstract and specialised actions, here there is no need of an object as there is only one grounded action, so there is no need to implement the predicate with variable, as there is only one instantiation. If there are more than one instantiations of the whole pattern, then one can consider introducing objects and predicates with variables. However, in the case where there is only one instance of the pattern, the necessary objects are 0.

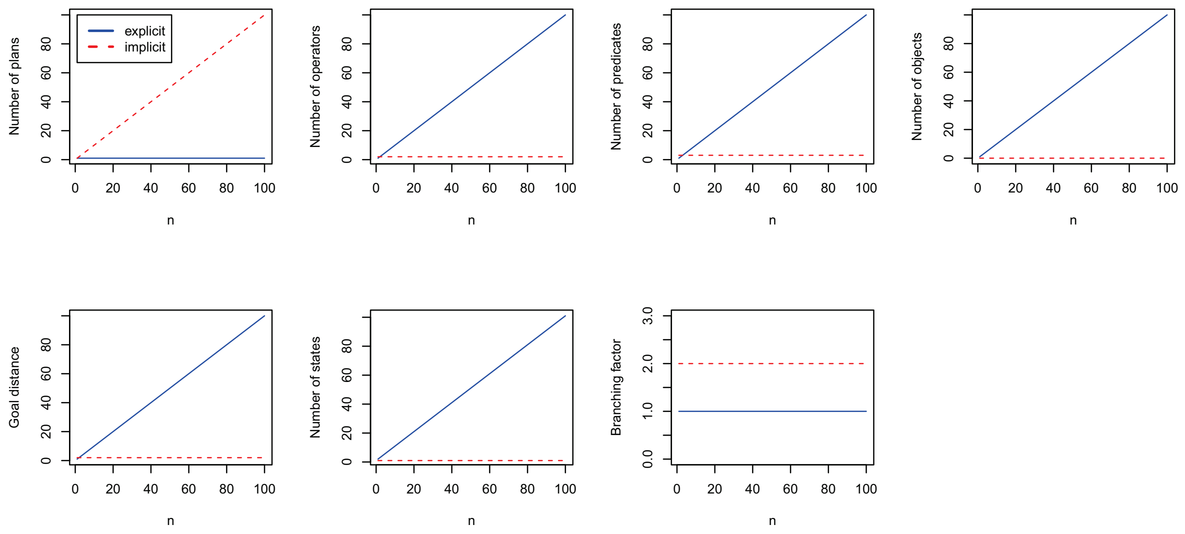


Figure 4.7: General model dynamics for repeating behaviour. The dashed red line represents the behaviour of explicitly modelled repeating action, while the blue solid line – of an implicitly modelled action. The model complexity for increasing number of repetition was plotted starting with one repetition and increasing until 100 repetitions are reached.

that there is only one action of the given type that is executable from a given state and that the action will be repeated exactly the needed number of times. Such implementation is preferable in situations where there is prior knowledge about the number of times the action is executed.

On the other hand, the implicit implementation is faster to implement and allows more flexibility in the model. It also has a constant number of operators, predicates, objects, states, goal distance and branching factor compared to the explicit implementation where these factors are linearly increasing with the number of actions. However, it could cause unexpected problems in the model, like for example executing the *finish* action too early, thus blocking any further execution of the action[9]. One option for coping with this problem, although not optimal, is to remove the *finish* action and allow the action to be indefinitely repeatable. Later, the model should rely on the observations in order to infer how many times the action is executed. This, of course, is not an optimal solution as the repeating action is always executable, even when it should no longer be executed. A possible solution to that (in the case where there is no information about the number of iterations) is using an explicit implementation and making an educated guess about the maximum number of times the action can be repeated. Another option is just to rely on the sensors that they will provide accurate information about the *finish action*.

4.5 Macro structures

Motivation: Macro structures are composed actions that consist of *begin-end* action pair and that could encompass other actions in between. The need of such modelling mechanism comes from situations where interleaving actions are required or where a composite action

[9]One can argue that given the appropriate preconditions the point at which the finish action is executed can be controlled. However, this can be achieved by introducing a mechanism for tracking the number of times the action was executed, thus reducing it to the explicit implementation of repetition.

is expressed. Like with the repeating actions, macro structures can be expressed either in explicit or implicit way. That means, given more than two actions involved, one could either explicitly specify the order in which the actions can take place, or one could leave it to the inference engine and the observations to decide what is the action sequence. Here both cases are presented and their influence on the model is discussed.

Structure and implementation: Macro structure is implemented by creating a *begin-end* action pair that due to its preconditions and effects is able to enclose in between the begin and end part of a set of other actions. This is shown in Fig. 4.8 where it can be seen that three macro actions are executed sequentially. The first action is a macro structure that encloses the remaining two macro actions, while the second and the third are interleaving in between the first action. There are two ways of implementing such behaviour – either explicitly defining

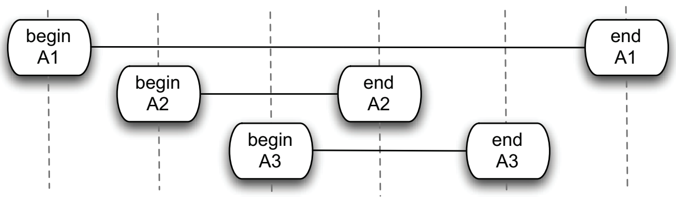


Figure 4.8: Structure of macro actions implementation. In this example there are 3 macro actions where the first is built of two other actions, and each of these two actions have parts interleaving with each other.

the preconditions and effects that influence the actions' order or by leaving the order to the inference engine and the received observations. The sample code showing the different implementations can be seen in Appendix E in Fig. E.5 on page 223 for the explicit implementation, and in Fig. E.6 on page 224 for the implicit implementation. It can be seen that both cases are implemented by an abstract *begin-end* pair that is later parameterised by the action number. However, while in the implicit case the actions' order is not specified, in the explicit case the order is given by constraints encoded in the additional predicates. In the second case, the first action can start only before the second and the third *begin* parts have been executed, and the second can start only after the first and before the third. The order for the third is specified in exactly the same manner.

Consequences: The two implementations also create different runtime models and have different consequences for the model dynamics. Table 4.3 shows the model parameters for the

Table 4.3: Parameters of macro structures implementation. The values for the explicit macro structures are based on the implementation in Fig. E.5 on page 223, while those for the implicit are based on the implementation in Fig. E.6 on page 224.

Parameter	Value (explicit)	Value (implicit)	Description
# operators	6	6	grounded actions after the model compilation
# predicates	6	6	grounded predicates after the model compilation
# objects	3	3	objects in the model
# states	7	27	state-space of the model
# valid plans	1	90	valid plans leading from the initial to the goal state
goal distance	6	6	distance from the initial to the goal state
max. branching factor	1	3	maximum number of possible actions from a state
final log likelihood	0	-5.0876	final log likelihood of the execution sequence

two implementations. Although both have the same number of operators (6), predicates (6), and objects (3), it can be seen that the implicit implementation has bigger state-space (27 states against 7 for the explicit). This is caused by the more complex preconditions in the explicit macro structure where it is specified in what order the actions can be executed. This also increases the number of valid plans for the implicit implementation, as in the explicit there is only one possible execution sequence. This is also reflected in the increased branching factor for the implicit implementation and the lower log likelihood.

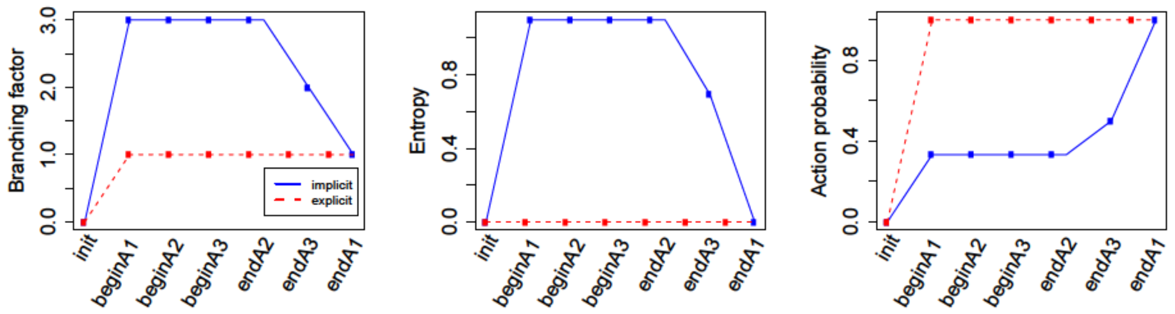


Figure 4.9: Model dynamics for macro structures. The behaviour of the explicit implementation is presented with a dashed red line, while that of the implicit – with a solid blue line.

The same model dynamics can be seen in Fig. 4.9 where throughout the model execution, the explicit implementation always has a branching factor of 1 while the number of possible actions for the implicit implementation increases to 3. It decreases only at the end of the execution when most of the possible actions are already executed. The entropy also reflects the increasing number of choices in the implicit implementation, while at the same time the actions' probability decreases and is increased only during the last two actions. On the other hand the values for the explicit implementation always stay constant due to the lack of choices in a given state. This reflects the final plan log likelihood: for the implicit model it decreases with each executed action, while it stays 0 for the explicit (the final log likelihood in Table 4.3).

General case: Above we showed with a dummy model that explicitly modelled macro structures reduce the model complexity compared with such where the actions sequence was implicitly modelled. Here we look at the general case and show that the behaviour dynamics will stay the same. We also show that both explicitly and implicitly modelled macros will have smaller number of states and branching factor compared to a plain model that does not use macros.

Proposition: *A model that does not use macro structures will be more complex in terms of number of plans, number of objects, number of states and a higher branching factor, compared to a model that uses macro structures. Furthermore, a model that uses implicit macro structures will be more complex that a model that uses explicit macro structures in terms of number of plans, number of states, and higher branching factor.*

Assume we have a model B_1 that does not use any macro structures and that has n actions which can be executed sequentially in any order without repetition and without any further restrictions. Such model will exhibit behaviour and has parameters as those in Equation 4.1 to Equation 4.7.

Now assume there is a model B_2 that uses implicit macro structures and that has n actions where each two actions form a begin-end action pair. In such action pair the first action has to be executed before the second. On the other hand, it is possible that other actions are executed in between each action pair. Assuming that all actions have to be executed in order to reach the goal state, then the number of valid plans Pl_{B_2} can be calculated by calculating the number of permutations one can do with some identical elements. In our case the identical elements will be the beginning and end actions of the macro structures because for calculating the different combinations it does not matter which part of the pair we call *begin* and which *end*. In n actions we will then have $n/2$ pairs of identical elements. Then the number of plans can be calculated by applying the formula for permutations of multisets (see Formula F.9 in appendix F). In our case we have $n/2$ times two identical elements which will transform the denominator in the formula to $2!^{n/2}$.

$$Pl_{B_2} = \frac{n!}{2!2!2!...2!} = \frac{n!}{2!^{n/2}} = \frac{n!}{2^{n/2}}. \tag{4.22}$$

The number of operators Op_{B_2} corresponds to the available actions and is

$$Op_{B_2} = n. \tag{4.23}$$

If we assume that in the best case we need at least two predicates per action pair to manage which actions were already executed (one for indicating that the action is in progress and another for indicating the end of the action), then we will need at least n grounded predicates.

$$Pr_{B_2} = n. \tag{4.24}$$

If we have $\frac{n}{2}$ action pairs then we will need at least one object per action pair in order to manage the executed actions, namely

$$Ob_{B_2} = \frac{n}{2}. \tag{4.25}$$

The length of the goal distance will be equal to the length of the shortest execution plan, namely

$$G_{B_2} = n. \tag{4.26}$$

Furthermore, assuming that at least n predicates are needed to represent the actions, and that at most half of them can be true in a given state[10], then we will have at most $m = n/2$ predicates from which to create new states. When we have all m predicates set to true, namely the predicates indicating an action is being executed, then we can have only one state, because all the remaining predicates that represent the action has ended can only be false in this case. On the other hand if we have less than m predicates set, then we can generate $k < m$ times 2^k different combinations of predicates. For example, if we have $m-1$ predicates indicating the action is being executed set to true, then we have still one of these predicates set to false. This could mean two things – either that the predicate indicating the end of the action is true (the action has ended) or that the action has never started. Then for each $k \leq m$ we can compute the number of k-combinations of sets given m elements (see Formula F.7 in Appendix F). This will produce the number of combinations for the predicates indicating the action is being executed. Then to compute the corresponding number of combinations for the predicates indicating the end of the action, given the number of k *being executed* predicates that are set to false, we just multiply the result from Formula F.7 with the number of subsets k elements can produce (see Formula F.8). In other words, our state space will be represented by the following formula.

$$S_{B_2} = \sum_{k=0}^{m} \binom{m}{k} 2^k, \tag{4.27}$$

[10]That is because the first predicate in the predicates pair indicates the action is being executed while the second indicates that the action has ended, thus both cannot be true at the same time.

with $m = n/2$.

The branching factor then in the worst possible case will be equal to the half the number of the available actions as each two actions have a specific order,

$$Br_{B_2} = \frac{n}{2}. \tag{4.28}$$

Now assume, there is a model B_3 that uses explicit macro structures. It has n actions and each two actions form a begin-end action pair. The model has the same properties as B_2, but in addition the actions can be explicitly executed in only one way. Assuming that all actions have to be executed in order to reach the goal state, then the number of valid plans Pl_{B_3} will be

$$Pl_{B_3} = 1, \tag{4.29}$$

where the number of operators Op_{B_2} corresponds to the available actions and is

$$Op_{B_3} = n. \tag{4.30}$$

Like in the case with B_2, if we have n actions each two of which form an action pair, we will need at most 2 predicates per action pair to manage the start and end effects of the action, thus the number of predicates will be equal to

$$Pr_{B_3} = 2\frac{n}{2} = n. \tag{4.31}$$

If we assume that in the best case we need objects only for managing the sequence of executed actions, then we will need at least $n/2$ objects as we have $n/2$ distinct action pairs.

$$Ob_{B_3} = \frac{n}{2} \tag{4.32}$$

The length of the goal distance will be equal to the length of the shortest execution plan, namely

$$G_{B_3} = n. \tag{4.33}$$

Furthermore, assuming that at least n actions are needed to reach the goal and that there is only one execution sequence, then the number of states in the model will be equal to

$$S_{B_3} = n + 1. \tag{4.34}$$

The branching factor will always be 1 as there is only one possible action that can be executed from a given state,

$$Br_{B_3} = 1. \tag{4.35}$$

Comparing the parameters of the three models it can be seen that B_1, which does not make use of macro structures or any additional constraints has the largest number of valid plans – $n!$ (Equation 4.8) – compared to the $\frac{n!}{2^{n/2}}$ plans of B_2 (Equation 4.22). On the other hand, the third model B_3 that in addition to the macro structure, also contains explicit order in which the actions can be executed, has only one valid plan. In terms of operators, all three models have the same number of actions, thus the same number of operators. The number of predicates for B_1 is n (Equation 4.3), as is also the case for B_2 and B_3. The number of objects in B_1 is n (Equation 4.4), while in the case of B_2 and B_3 they are $n/2$ as each two actions create an action pair and less objects are needed for managing the action sequence. The length of the plans is also the same for all three models as all actions in the model have to be executed to reach the goal state. They however differ in their number of states. While B_1 has 2^n states, B_2 will have $\sum_{k=0}^{m}\binom{m}{k}2^k$ states. In the case of B_3, the state space will be much smaller as there is only one execution sequence, thus there are only $(n+1)$ states. This is also reflected in the models' branching factor – while B_1 has maximum of n actions that can be executed from a state, B_2 has $\frac{n}{2}$, whereas B_3 has only 1. This proves that using implicit macro structures decreases the model complexity compared to models that do not use any macro structures. Additionally, using explicit macro structures further reduces the model complexity. The above can also be

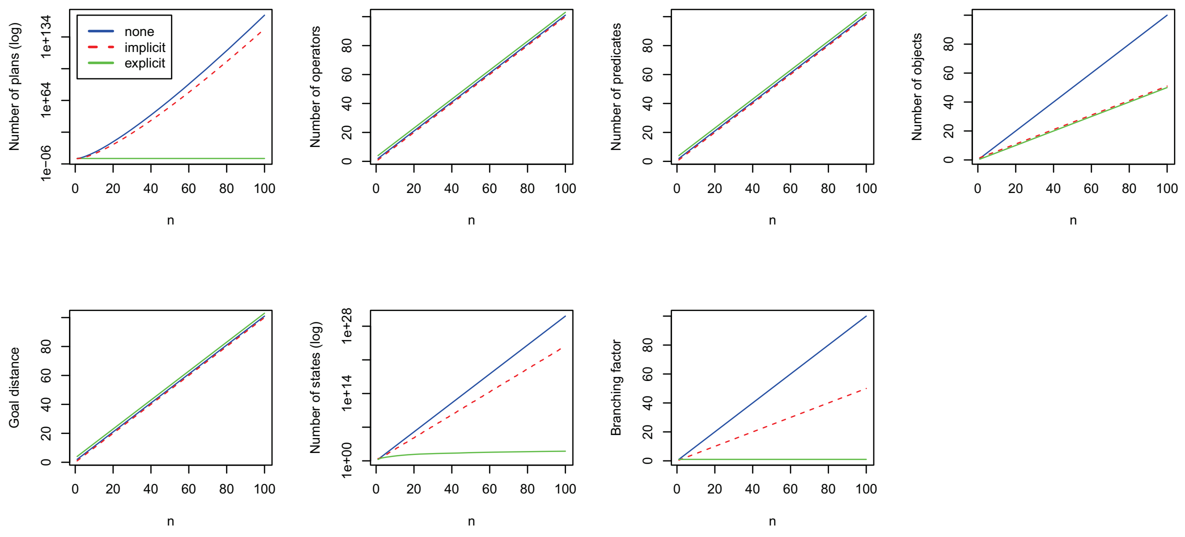


Figure 4.10: General model dynamics for macro actions. The solid blue line represents the behaviour of actions that do not use macro structures, the dashed red line shows the behaviour of implicitly modelled macro structures, while the blue solid line of explicitly modelled. The model complexity for increasing number of repetitions was plotted starting with one repetition and increasing until 100 repetitions are reached. The number of states and number of plans are plotted on a logarithmic scale.

seen in Fig. 4.10 where the model complexity for an increasing number of actions was plotted.

Applicability: The macro structures are useful in situations where the user's actions are not forced by an external factor like in the case with the team meeting model, where the user behaviour was forced by the team behaviour. On the other hand, the multi-agent behaviour was implemented exactly with *begin-end* action pairs. They ensured that the user action will not be blocked by another action, but rather will end by itself evoking its own effects.

Another application for the macro structures is defining composite actions, where the *begin-end* pair via its preconditions and effects enforces a certain set of actions to be executed in between.

Macro structures are also used for expressing interleaving actions where one action starts, then it is interrupted by another and then continues again. Such is the case with filling something in an object in the cooking task, or eating then stopping in order to drink water, then continuing to eat. These actions could also be expressed by repeating actions and the choice is left to the prior knowledge available when implementing the model and whether the action is independent or influenced by another action.

Like in the case with explicit and implicit repeating actions, it depends on the context information, whether to use explicit or implicit macro structures. When the order in which the actions are executed is already known, it is better to use explicit definition, as it reduces the number of possible actions and increases the action's probability. However, if such information is not available beforehand, then the implicit implementation is the only choice, as it can cope with the behaviour variability.

4.6 Actions synchronisation

Motivation: In problems where multi-user behaviour is expressed and where the separate users' actions have to be coordinated for achieving a common goal, action synchronisation is needed. For example, in the meeting scenario, the users' actions had to be synchronised in order to achieve the team goal. This was done in two ways – the first was that the team behaviour enforced the actions synchronisation on the separate users' level, whereas the second option was to let the users for themselves decide when a synchronised action can take place. As it was already shown in the meeting problem, both options have their advantages and can be used depending on the model purpose. Here we explain in detail how the actions' synchronisation was implemented and discuss both cases.

Structure and implementation: The idea behind synchronising actions is that when multiple users are acting in an environment, at a given point a mechanism is triggered that creates constraints synchronising the actions of the users with each other. For example, in the meeting scenario the users enter the room and can do whatever they want within the limits of the room, but when one of them prepares for presentation, their actions are synchronised and they have to finish whatever they are currently doing and prepare for the common goal of conducting a presentation. Fig. 4.11 shows the structure of synchronising the actions of two users, where the synchronisation itself can be enforced by the team or can be made as a conscious decision by each of the users. The implementation of the two options can be found in Appendix E in

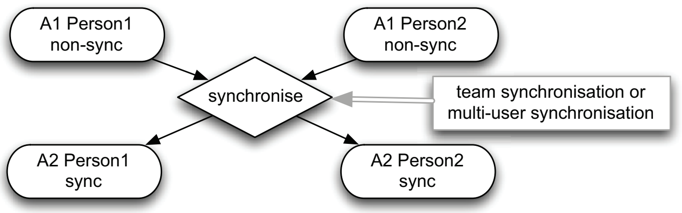


Figure 4.11: Structure of implementation for synchronising actions. In it the actions of two agents are executed in an unsynchronised manner until either the team forces them to coordinate their next actions, or they make the synchronisation based on their own free choice.

Fig. E.7 on page 225 for the synchronisation enforced by the team, and in Fig. E.8 on page 226 for the multi-agent synchronisation. It can be seen that in the first case (Fig. E.7) each of the users have to execute an action. However, the moment all users have started executing the action, and one has finished it, she can start executing the synchronisation action thus enforcing it to all remaining agents in the environment. This is done in the action's effect, where the *forall* clause enforces the agents to quit their actions and to be able to execute only the synchronised action. In that manner time information about the action is lost as it is possible it was never executed until the end. On the other hand, the implementation of the second option, where each agent has to make her own decision, it can be seen that a macro structure is used, ensuring that the agents will not be influenced by external factors while executing their actions. Only after the actions are completed, the synchronised actions can take place (see Fig. E.8). The difference between the two implementations is that while in the team case, the expected behaviour is enforced by the effects of the synchronising action, in the multi-agent case, the required conditions for executing the synchronised actions are specified in the preconditions.

Consequences: The two options for implementing synchronisation, differ not only in the details about the actions' durations but also in the runtime models' parameters and their dynamics during execution. Table 4.4 show the parameters for the two implementations. It can be

Table 4.4: Parameters of synchronisation implementation. The values for the team synchronisation are based on the implementation in Fig. E.7 on page 225, while those for the multi-agent synchronisation are based on the implementation in Fig. E.8 on page 226.

Parameter	Value (team)	Value (multi)	Description
# operators	4	6	grounded actions after the model compilation
# predicates	7	8	grounded predicates after the model compilation
# objects	2	2	objects in the model
# states	7	12	state-space of the model
# valid plans	4	12	valid plans leading from the initial to the goal state
goal distance	4	6	distance from the initial to the goal state
max. branching factor	4	2	maximum number of possible actions from a state
final log likelihood	-4.85203	-2.77259	final log likelihood of the execution sequence

seen that the synchronisation enforced by the team has less operators and less predicates, thus generating smaller state-space. That is due to the usage of macro structure in the multi-user synchronisation for ensuring that the users make the decision of finishing the action independently. On the other hand, the fact that the first synchronised action in the multi-user case can take place only after all other actions are executed, decreases the model's branching factor (2 against 4 in the team enforced synchronisation) and the number of valid plans (12 in the multi-user case against 4 in the team case). The lack of macro structures in the team implementation decreases the goal distance and the shortest path to the goal is 4 states instead of 6 in the multi-user implementation. Because of the higher branching factor in the team implementation the final log likelihood of the model is smaller than for the multi-agent implementation, making the chosen plan less probable than for the multi-agent implementation.

This is also reflected in Fig. 4.12 that shows the model dynamics for a plan where the non-synchronised action of the first user is executed twice, then the non-synchronised action of the second user is executed twice, followed by the first user executing the synchronised action then the second. While in the multi-user behaviour, the model is just waiting until both users have executed their actions, in the team behaviour, the synchronisation is enforced by the first execution of the synchronised action. This can be seen in the increasing branching factor for the team enforced implementation, where after the first execution of the second user's non-synchronised action, the preconditions for executing the first user, non-synchronised action, as well as those for executing the synchronised actions are met, increasing the branching factor from 2 to 4. On the other hand, the branching factor for the multi-agent implementation decreases directly before the end part of the second user action is executed as it is the only possible action left. Then the branching factor once again increases to 2 as the synchronised actions for the two users are now possible. The same behaviour dynamics are also reflected in the model entropy, where for the team implementation the entropy increases at the synchronisation step, while for the multi-user implementation it decreases. On the other hand the dynamics for the actions probabilities is exactly the opposite with the model entropy and the probability increases to 1 at the synchronisation step for the multi-user implementation while it decreases for the team implementation.

General case: Above we showed in a dummy example of the consequences of two different ways for users actions' synchronisation. Below we discuss the model complexity in a more

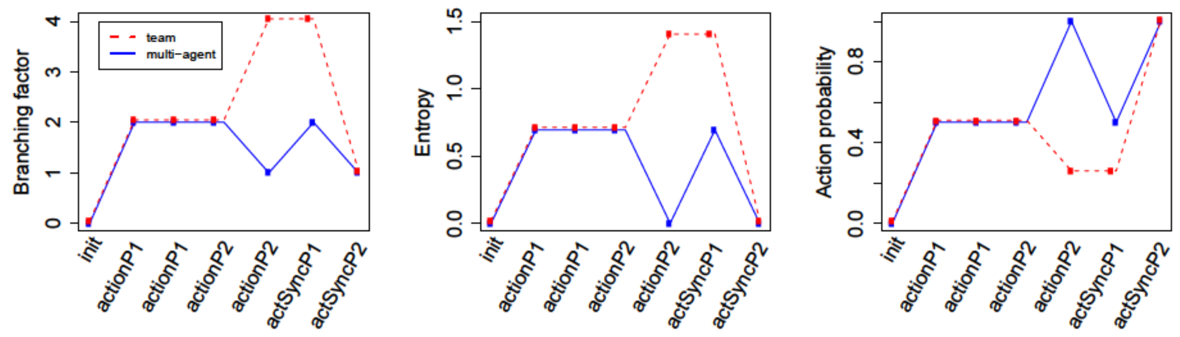


Figure 4.12: Model dynamics for synchronised actions. In the figures the team behaviour is represented with a dashed red line while the multi-agent – with a solid blue line.

general case.

Proposition: *A model using action synchronisation has smaller number of plans, states and objects than a plain model. Furthermore, a model that uses a team synchronisation has a smaller number of operators, predicates, and states than one using multi-agent synchronisation. On the other hand the multi-agent synchronisation reduces the model's branching factor.*

Assume we have a non-synchronised model B_1 that has n actions which can be executed sequentially in any order without repetition and without any further restrictions. Additionally, the actions are executed by at least two agents. Such model will exhibit behaviour and has parameters as those in Equation 4.1 to Equation 4.7.

Now assume there is a second model B_2 that uses team synchronisation and that has n grounded actions which are divided in two sets, the first containing l actions, the second – m; and where $m+l=n$. Moreover all l actions from the first set have to be executed before the m actions from the second set can be executed. Assuming that all actions have to be executed in order to reach the goal state, then the number of valid plans Pl_{B_2} can be calculated by calculating the number of all combinations of l elements for the unsynchronised actions, then multiplying by all combinations of m elements that represent the synchronised actions. This results in

$$Pl_{B_2} = l! * m!, \tag{4.36}$$

where the number of operators Op_{B_2} corresponds to the available actions and is

$$Op_{B_2} = l + m = n. \tag{4.37}$$

If we assume that in the best case we need at least one predicate indicating that an action can be executed, one indicating that the unsynchronised action is being executed, one indicating that the synchronised action is being executed, and one predicate without parameter indicating the synchronisation has taken place, then depending on the number of users in the environment we have:

$$Pr_{B_2} = 3 * Ob_{B_2} + 1, \tag{4.38}$$

where Ob_{B_2} is the number of objects with which the predicates are grounded.

If we assume that in the best case there are only two objects used for the representation of two users, then the number of objects will be 2. In a more general case where p users are interacting in the environment, the number of obligatory objects will be equal the number of users. This number will also be equal to the number of synchronised or unsynchronised actions, assuming there is only one synchronised or unsynchronised action per user that can be executed.

$$Ob_{B_2} = p = l = m = n/2 \tag{4.39}$$

and the length of the goal distance will be equal to the length of the shortest plan, namely

$$G_{B_2} = n. \tag{4.40}$$

Furthermore, assuming that in the best possible case only two users interact in the environment, then the minimal state space of the model S_{B_2} can be computed by the number of predicate combinations when the users can execute unsynchronised actions, namely 2^2. Here these are the two predicates – one indicating the user can do something, and one indicating the user is doing something unsynchronised – times the number of users. Then the number of combinations for the synchronised actions $(2^2 - 1)$ is added. Here these predicates are that the user is doing something synchronised times the number of users. The reason that there is one combination less than with the unsynchronised actions is due to the fact that in the case no unsynchronised actions can be executed, at least one of the synchronised actions should be true. In a more general case where we have p users, the state space can be computed in the following way:

$$S_{B_2} = 2^p + 2^p - 1 = 2^{p+1} - 1 = 2^{l+1} - 1 = 2^{(n/2)+1} - 1. \tag{4.41}$$

The branching factor then in the worst possible case will be equal either to l until all non-synchronised actions are executed, and m during the execution of the synchronised actions. The only exception is at step $l+1$ as at that point the preconditions for all synchronised and also for all non-synchronised actions will hold. In that case, the maximum branching factor will be

$$Br_{B_2} = l + m = n. \tag{4.42}$$

Assume also, there is a third model B_3 that uses multi-agent synchronisation. The actions in this model are also divided into two sets – unsynchronised and synchronised. However in difference with the actions in B_2 here each unsynchronised action is expressed with a begin-end action pair that ensures the action will be completed successfully. This means that we no longer have $l + m = n$ actions but rather $2l + m$ where $l + m$ equals the number of operators n from B_2. Like in model B_2, the actions from the first set have to be executed before those of the second can be executed. In addition the actions from 1 to $2l$ that are combined in a begin-end macro pairs have the properties described in Equation 4.22 to Equation 4.28. Assuming that all actions have to be executed in order to reach the goal state, then the number of valid plans Pl_{B_3} will be calculated by computing the number of plans for the unsynchronised actions according to Formula 4.22, then multiplying them by the number of combinations for the synchronised actions.

$$Pl_{B_3} = \frac{2l!}{2^{2l/2}}! * m! = \frac{2l!}{2^l} * m! \tag{4.43}$$

The number of operators Op_{B_3} corresponds to the available actions and is

$$Op_{B_3} = 2l + m. \tag{4.44}$$

The minimal number of predicates that are to be grounded will be 4: one predicate that indicates whether the user is doing something, one indicating that she did something, one indicating that she is allowed to do something, and finally, one predicate for the synchronised action. To calculate the minimal number of grounded predicates one has to then multiply these 4 predicates by the number of users in the environment.

$$Pr_{B_3} = 4 * Ob_{B_3}, \tag{4.45}$$

If we assume that in the best case there are two objects used in the actions (for expressing the two users), then the number of objects will be 2. In the more general case and assuming each user has to execute one synchronised and one unsynchronised actions, the number of users will still be the same as in B_2, namely

$$Ob_{B_3} = p = l = m = n/2, \tag{4.46}$$

and the length of the goal distance will be equal to the length of the shortest execution sequence, namely

$$G_{B_3} = 2l + m. \tag{4.47}$$

Furthermore, assuming that at least $2l+m$ actions are needed to reach the goal, and assuming that the initial and the goal state are always the same in each execution sequence, the state space S_{B_3} can be calculated partially with the help of Formula 4.27 where in this case $m = p = l = n/2$. This is the behaviour of the two predicates *doing something* and *did something* which express a macro structure. To the number of combinations these two predicates generate, additional combination has to be added for the predicate *can do something* as it can also be either true or false when both *doing something* and *did something* are false. In the remaining cases it is true only when *doing something* is true. Finally, to add the remaining combinations from the synchronised actions one has to compute $Ob_{B_3}! = p!$ which is the number of combinations of the grounded *doing something synchronised* predicate. This results in the following formula.

$$S_{B_3} = \sum_{k=0}^{p} \binom{p}{k} 2^k + 1 + p! = \sum_{k=0}^{n/2} \binom{n/2}{k} 2^k + 1 + (n/2)! \tag{4.48}$$

The branching factor then will be either l or m depending on which of the two is greater,

$$Br_{B_3} = \begin{cases} l & \text{if } l > m \\ m & \text{otherwise.} \end{cases} \tag{4.49}$$

Comparing the parameters of the three models, it can be seen that B_1 which does not make use of action synchronisation has larger number of valid plans – $n!$ (Equation 4.8) – compared to the $(l! + m!)$ plans of B_2 (Equation 4.36) that uses team behaviour synchronisation. On the other hand, the third model B_3 that allows multi-agent behaviour synchronisation has $\frac{2l!}{2^l} + m!$ plans (Equation 4.43). This means that B_1, has more plans than B_2, and B_3 has more than B_2 and B_1. This is caused by the macro structures in the third model that increase the number of operators compared to B_2 and B_1. In terms of operators, the first two models have the same number of actions, thus the same number of operators, while the third has $2l+m$ because of the begin-end action pairs with which the user behaviour is expressed. The number of predicates for B_1 is n which equals the number of actions, while for B_2 it is $3*p+1$ where p is the number of users. For B_3 the number of predicates is slightly higher than for B_2 – $4*p$ – which is due to the begin-end action pairs. The number of objects for the first model equals the number of actions. For the two models using action synchronisation the number of objects is constant, and in the best case the models will need two objects for expressing the minimum of two users. In a more general case it will be p objects corresponding to the number of users. The length of the plans is also the same for the first two models, while for the third it is $2l+m$. The models differ in their number of states. While B_1 has 2^n states, B_2 will have $2^{p+1}-1$ states. In the case of B_3, the state space will be $\sum_{k=0}^{p} \binom{p}{k} 2^k + 1 + p!$ which makes it increase faster than that of B_1 and B_2 for increasing number of actions. The models' branching factor for

B_1 and B_2 has maximum of n actions that can be executed from a state, while B_3 has l or m, which is smaller than that of B_1 and B_2. This shows that the multi-agent synchronisation reduces the branching factor as the synchronisation is not enforced by another action but rather is a result from the agents own decision to synchronise their actions. The models' dynamics for increasing number of actions can be seen in Fig. 4.13. It can be seen that the number of

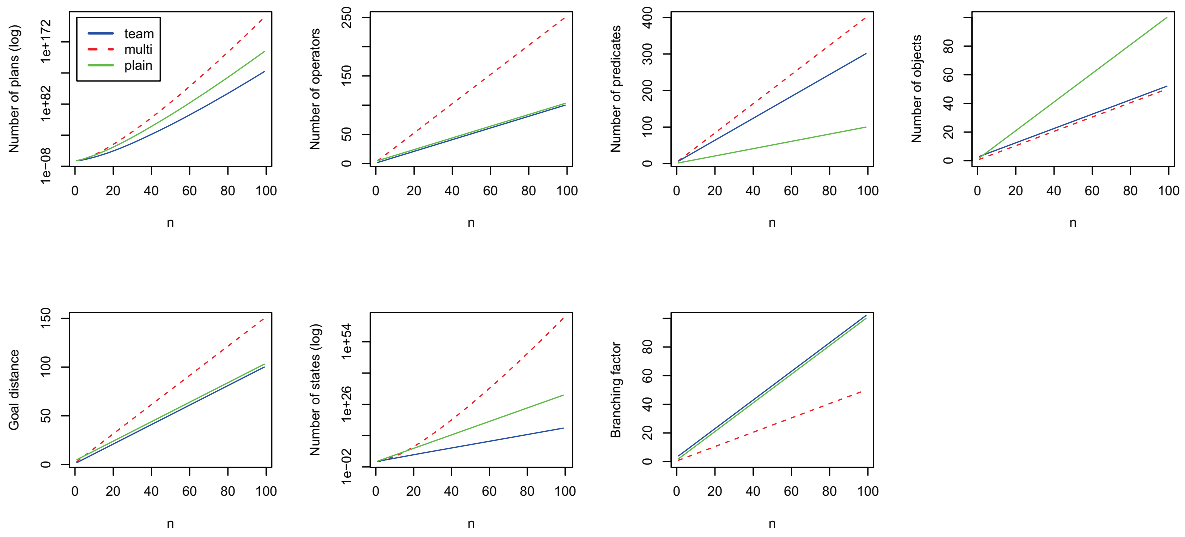


Figure 4.13: General model dynamics for synchronised actions compared to plain model. The solid green line represents the behaviour of actions that do not use synchronisation structures, the dashed red line shows the behaviour of multi-agent modelled macro structures, while the blue solid line of team synchronisation. The model complexity for increasing number of actions was plotted (1 to 100). The number of states and number of plans have been plotted on a logarithmic scale.

plans for a team synchronisation model increases more slowly than that of the plain model. The multi-agent model on the other hand increases its number of plans faster than the other two models due to its macro structures. The number of operators increases linearly with that of multi-agent synchronisation being higher due to the two operators needed for expressing an action. The same applies for the goal distance. The number of objects, on the other hand, is smaller for the synchronised models as for n actions one needs at most $n/2$ users to execute them. The state space is smaller for the team synchronised model than for the other two models. The number of states in the multi-agent increase faster than the other two models due to the higher number of operators needed for expressing an action. Finally, while the branching factor for a plain and a team synchronised model equals the number of operators, in the case of multi-agent synchronisation, it is smaller because one can execute either the synchronised or the non-synchronised actions.

Applicability: As already mentioned, the synchronisation mechanisms can be applied in situations where multiple users are acting in the same environment and there is some common goal that they have to follow. The team enforced synchronisation can be applied in cases where only the team behaviour is of importance and where the model does not need to explain the separate single user actions (like in the team model for the three person meeting). The multi-user synchronisation on the other hand, allows the users to make their own choices to follow the common goal. Both models have their advantages, as the team model needs less operators, predicates, plans, and a smaller state space. On the other hand, the multi-agent synchronisation has a smaller branching factor ensuring that irrelevant actions do not influence the actions probability. Thus, depending on the application one could use both of them.

4.7 Type hierarchy

Motivation: Using types in the human behaviour model is essential in situations where a given action template needs to be parameterised with just a subset of the available objects. This is important for reducing the number of available operators and grounded predicates, thus also reducing the number of possible actions from a given state. On the other hand, it also allows specific objects to inherit multiple types making them applicable for more than one parameter type in an action template. Such system allows flexible work with the objects in the environment and makes the model dynamics easier to control. It also allows the existence of only those operators that are possible for the given problem. Here we show example of how the type hierarchy influences the model compared to such without type hierarchy.

Structure and implementation: CCBM allows a simple way of creating type hierarchy. There is a general default type and every other type that is defined is a subtype of the default type. One can then create subtypes of the subtypes and also to allow a given type to have more than one parent type. Fig. 4.14 shows the structure of the type system where it can be seen that *specialised type 2 on level* n is a subtype of two parent types. Examples of the type system were also shown in Chapter 3.3 where Figures 3.7, 3.15, and 3.20 show the type system for the three modelling problems. The usage of the type system is also relatively easy.

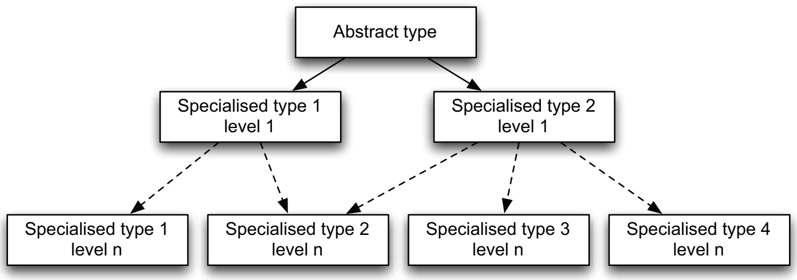

Abstract type
Specialised type 1 level 1
Specialised type 2 level 1
Specialised type 1 level n
Specialised type 2 level n
Specialised type 3 level n
Specialised type 4 level n

Figure 4.14: Structure of implementation for type hierarchy. It can be seen that there is one abstract parent type from which all specialised subtypes emerge. Each subtype in turn can have its own children subtypes.

When defining an action template one has to just choose the appropriate type for the action's parameters thus allowing only specific objects to be used for the action parameterisation. That way combination of actions and objects that do not appear in the problem are avoided. An example of a type hierarchy implementation can be found in Appendix E in Fig. E.9 (page 227) where there are three subtypes defined, and 8 objects belonging to these subtypes. Later when defining the action *manipulate* its parameter is of *type1* thus parameterising the template only with the objects of this type. Were there no type system, the action would be parameterised with all 8 available objects, instead with the 5 objects from *type1*.

Consequences: The usage of a type system in a problem changes the runtime model parameters and the model dynamics during execution. Table 4.5 shows the parameters for the same model, where in the first case a type hierarchy was used and in the second it was without hierarchy. It can be seen that the usage of the type system reduces the number of operators and predicates from 8 in the plain case to 5 in the hierarchical as the action is parameterised only with the relevant objects. This also reflects in the model state space decreasing it from 256 states in the plain case to only 32 when using the type system. It also creates much less

Table 4.5: Parameters of hierarchy implementation. The values for the hierarchical object usage are based on the implementation in Fig. E.9 on page 227. The values for the plain model are based on the same implementation with the difference that the parameters that the action can take are from the general object type (or with other words any object is applicable to the action).

Parameter	Value (hierarchical)	Value (plain)	Description
# operators	5	8	grounded actions after the model compilation
# predicates	5	8	grounded predicates after the model compilation
# objects	8	8	objects in the model
# states	32	256	state-space of the model
# valid plans	120	13 944	valid plans leading from the initial to the goal state
goal distance	5	5	distance from the initial to the goal state
max. branching factor	5	8	maximum number of possible actions from a state
final log likelihood	-4.78749	-8.81284	final likelihood of the execution sequence

valid plans and narrows the model diversity. This, in a case where the model heuristics are not very good, could play an essential role in recognising the correct action. That is reflected in the branching factor where in the plain case each operator can be executed from a given state, while in the hierarchical, only those operators parameterised with the desired subtype are possible. It also changes the final log likelihood making it almost twice higher for the hierarchical case compared to the plain model.

Concerning the model dynamics – it can be seen in Fig. 4.15 that the branching factor is always about 0.6 times higher in the plain case, which also influences the model entropy. It can be seen that while at the last action execution the entropy for the hierarchical case is decreasing to 0, for the plain it still stays high, as there are 3 more actions that are executable. This is also reflected in the actions probabilities, where in the hierarchical case it is decreasing with each executed action while in the plain case, it stays relatively low as there are still three more actions available.

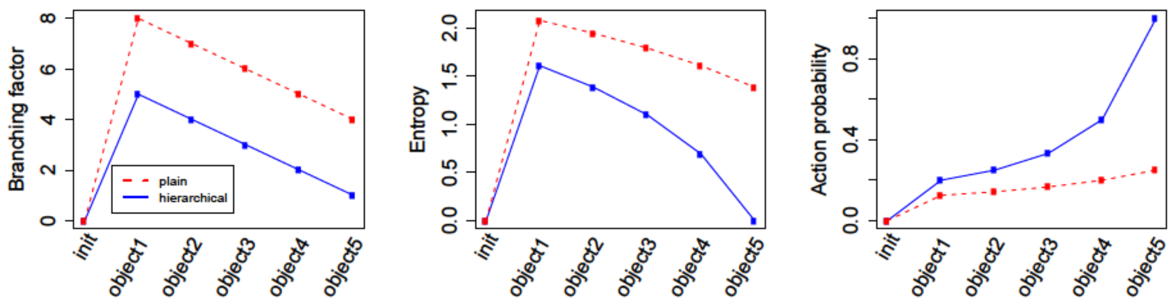


Figure 4.15: Model dynamics for type hierarchy. The plain model that does not use any hierarchy is represented by dashed red line, while the one using hierarchy is shown as a solid blue line.

General case: Above we showed the behaviour of plain versus hierarchical models in a dummy example. Here we consider the general case for any model that contains type hierarchy.

Proposition: *A model that makes use of type hierarchy reduces the number of operators, states, plans, as well as the branching factor compared to a plain model.*

Assume we have a model B_1 with actions $A := \{a_1, a_2, ..., a_m\}$ where $m \in M$ is the number of objects available. Then the model has m actions that can be executed sequentially in any order without repetition and without any further restrictions. If we assume that in order to reach the goal, n actions have to be executed with $n < m$, then the number of valid plans Pl_{B_1} can be calculated by calculating the combinations of all plans of length n containing the n actions, then summing them up with all plans of length $n+1$ containing all n actions needed to reach the goal, then all plans with length $n+2$ and so on. Or with other words,

$$Pl_{B_1} = \sum_{i=n}^{m} i!, \tag{4.50}$$

where the number of operators Op_{B_1} corresponds to the available actions and is

$$Op_{B_1} = m. \tag{4.51}$$

If we assume that to express an action we need at least one predicate per action, then the minimal number of predicates will be equal to the number of actions:

$$Pr_{B_1} = m. \tag{4.52}$$

If we assume that in the best case there are m objects used in the actions, then the number of objects will be

$$Ob_{B_1} = m, \tag{4.53}$$

and the length of the goal distance will be equal to the length of the shortest plan, namely

$$G_{B_1} = n. \tag{4.54}$$

Furthermore the minimal state space of the model S_{B_1} is calculated by combining all possible predicate combinations, and given we have m predicates and no further constrains, then the state space will be

$$S_{B_1} = 2^m. \tag{4.55}$$

The branching factor then in the worst possible case will be equal to the number of available operators, namely

$$Br_{B_1} = m \tag{4.56}$$

Assume also there is a model B_2 that uses a type hierarchy. This model has $p := \{1, 2, ..., m\}$ objects with $p \in M$ where m is the number of objects but due to the type system used in the action templates, only a subset of them can be used for parameterising the actions. With other words we have $q := \{1, 2, ..., n\}$ where $q \in N$ objects and $N \subset M$. If we assume that an action template is parameterised by a single object, then the number of available actions we have will be n. Further assuming that in order to reach the goal, one needs to execute these n actions regardless of their order, then the number of valid plans Pl_{B_2} can be calculated by

$$Pl_{B_2} = n!, \tag{4.57}$$

where the number of operators Op_{B_2} corresponds to the available actions and is

$$Op_{B_2} = n. \tag{4.58}$$

If we assume that to express an action we need at least one predicate per action, then the minimal number of predicates will be equal to the number of actions:

$$Pr_{B_2} = n. \tag{4.59}$$

The number of objects will be m as all of the initial objects still exist in the environment.

$$Ob_{B_2} = m, \tag{4.60}$$

and the length of the goal distance will be equal to the length of the shortest plan, namely

$$G_{B_2} = n. \tag{4.61}$$

Furthermore, given that n actions are needed to reach the goal and that only n predicates exist, then the minimal state space of the model S_{B_2} is

$$S_{B_2} = 2^n. \tag{4.62}$$

The branching factor then in the worst possible case will be equal to the number of available operators, namely

$$Br_{B_2} = n. \tag{4.63}$$

As model B_2 has n actions while B_1 has m with $m > n$, the number of plans in B_2 will be less than those in B_1. The same applies for the number of operators, predicates, states and the branching factor. This is due to the fact that the action templates in the second model are always parameterised with a subset of all available objects while in B_1 each action template is grounded with all available objects. On the other hand, the number of overall objects in both models is the same, as well as the minimal goal distance as in both cases there are n actions that have to be executed to reach the goal. This stands to show that using type hierarchy reduces the model complexity in terms of valid plans and model states. The model dynamics can be seen in Fig. 4.16 that shows the model characteristics for increasing number of actions (from 1 to 100).

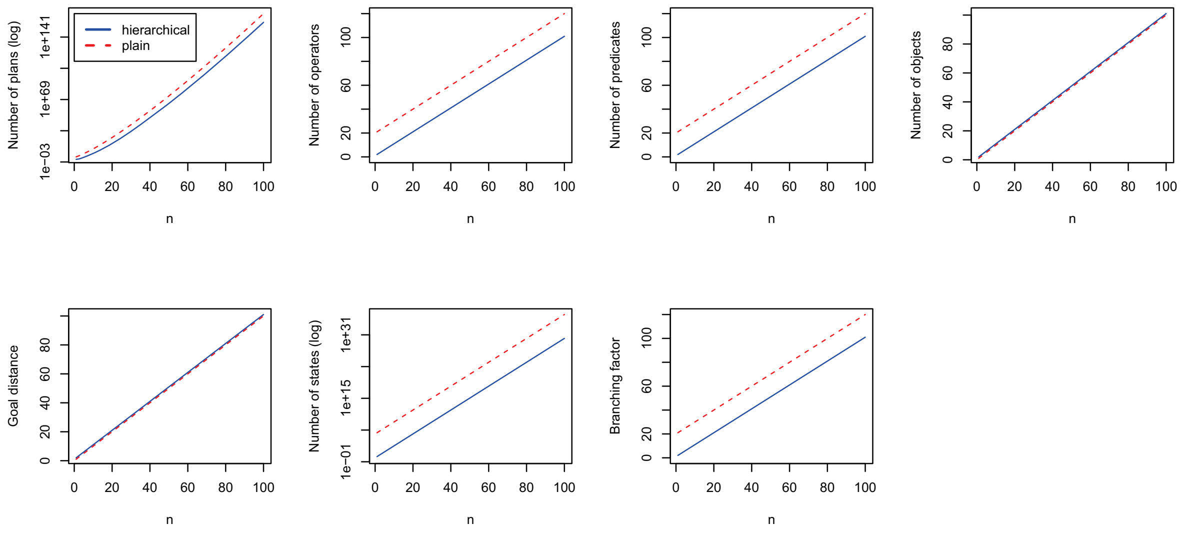


Figure 4.16: General model dynamics for hierarchical actions compared to plain model. The solid blue line represents the behaviour of actions that use type hierarchy, the dashed red line shows the behaviour of a plain model. The model complexity for increasing number of actions was plotted (1 to 100). The number of operators for the plain model was set to $n+20$ where n is the number of operators in the model using type hierarchy.

Applicability: The type hierarchy is applicable in any case where more than one objects exist that should not be applied to the same action. Additionally the more complex the problem, the more important it is to use such types system. For example, in the cooking problem it was essential to use a complex type system as the lack of such resulted in too many possible actions from a given state, which usually led to the wrong action being selected. In more simple problems such as the meeting scenario, it is not necessary to use overly complex type hierarchy, still the division of the different types of locations proved to be a useful tool for reducing the number of possible *move* actions from a given place.

4.8 Combining objects

Motivation: Combined objects are used in situations where there are many objects in the environment and an operator is manipulating more than one of them. In that case using the objects separately increases the model complexity due to the higher number of combinations possible. This leads to the existence of repeating combinations that only decrease the probability of the correct action. To avoid that, a mechanism can be applied that either makes use of a single combined object which in itself contains several objects, or that uses lock predicates to define which objects can be combined together. Here we discuss the way in which such mechanisms can be implemented and their consequences on the model parameters and dynamics.

Structure and implementation: To create combined objects one needs to define an object type and the corresponding constants for a combined object. Additionally, predicates are being introduced that allow assigning a single object to a predicate with combined object. The structure of such modelling mechanism is shown in Fig. 4.17 where it can be seen that two single objects are represented by a combined object. To get the single object id, two different predicates with the same combined object are used. This can also be seen in Appendix E in Fig.

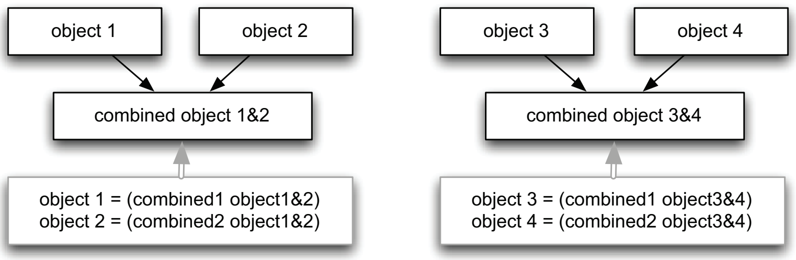

object 1
object 2
combined object 1&2
object 1 = (combined1 object1&2)
object 2 = (combined2 object1&2)
object 3
object 4
combined object 3&4
object 3 = (combined1 object3&4)
object 4 = (combined2 object3&4)

Figure 4.17: Structure of implementation for combined objects. The figure shows four objects each of which is combined in a pair of two. Later to retrieve each element in the combined object, a special predicate is used that provides the mapping from the combined object to the separate objects.

E.10 (page 228) where there are ten single objects and five combined, each of the combined representing two single objects. Later, the action *manipulate-two-objects* has as parameter a combined object and with the help of the predicates *combined-object1* and *combined-object2* sets the predicate indicating that a single object has been manipulated to true. The assigning of a predicate with a combined object to a single object is then done in the problem description, where each of the predicates with a combined object is assigned to one of the single objects. That way the model is able to represent the manipulation of all objects without creating redundant combinations.

Alternatively, the same can be achieved by using lock predicates that define which objects can be used together. Then the action that makes use of these objects is parameterised only with the objects that are allowed to be together and ignores all remaining combinations. An example implementation of this mechanism can be seen in Appendix E in Fig. E.11 (page 229). In it, the predicate *can-be-together* takes care of the object combinations that are allowed ignoring all the remaining combinations.

Consequences: Both types of model implementation have the same influence on the runtime model parameters and dynamics. Table 4.6 shows the parameters of the runtime model with combined objects, compared to those with lock predicates, compared to those of a model that uses operator that simultaneously manipulates two single objects without any lock mechanisms. As the compiled model generates all possible combinations of two objects and as there are 10 single objects it can be seen that the single objects model has 100 operators. However, most of these operators are redundant as the place of the objects (which of the two objects is first) does not matter. Furthermore, if we assume that only two specific objects can be manipulated together, the number of relevant operators will be reduced to 5. This is the case in the model using combined objects or lock predicates where the objects that can be combined together are already defined. Of course, the combined model has more objects as it also has to define the combined objects. On the other hand, using lock predicates does not increase the number of objects, yet all other model parameters are comparable with the first model. This significantly decreases the model state-space (compared to a model without lock mechanisms) and it can be seen that whereas the model using single objects has 1024 states, the combined both have only 32. This also generates much less valid plans which means that a given plan (or execution sequence) would have higher probability than in the single objects case. This is reflected by the models' final log likelihood where the combined objects models have 4 times higher likelihood than the one using single objects.

Table 4.6: Parameters of combined objects implementation. The values for the combined actions is based on the implementation in Fig. E.11 (page 229), those of the lock predicates is based on the implementation in Fig. E.11 (page 229), and the plain model is the same as the implementation with the lock predicates, with the difference that the predicate *can-be-together* is missing, thus allowing any possible combination.

Parameter	Value (combined)	Value (lock)	Value (plain)	Description
# operators	5	5	100	grounded actions after the model compilation
# predicates	10	10	10	grounded predicates after the model compilation
# objects	15	10	10	objects in the model
# states	32	32	1024	state-space of the model
# valid plans	120	120	11 551 680	valid plans leading from the initial to the goal state
goal distance	5	5	5	distance from the initial to the goal state
max. branching factor	5	5	100	maximum number of possible actions from a state
final log likelihood	-4.78749	-4.78749	-16.5065	final log likelihood of the execution sequence

This can also be seen in the model dynamics shown in Fig. 4.18 where a plan is executed that included the manipulation of the 10 single objects with five actions. It can be seen that while the branching factor for the combined objects (both when using one combined object or lock predicates) is always small, the branching factor for the single objects explodes at the beginning and slowly decreases with each executed action. This also makes the model entropy for the single objects case higher than for the models using lock mechanisms and in the same time assigns smaller probabilities to the correct actions. This makes it more probable that the inference engine will choose an incorrect action than in the case with the combined objects.

General case: Above we showed that using combined objects or lock predicates to combine objects reduces the model complexity compared to plain models. We also showed that the

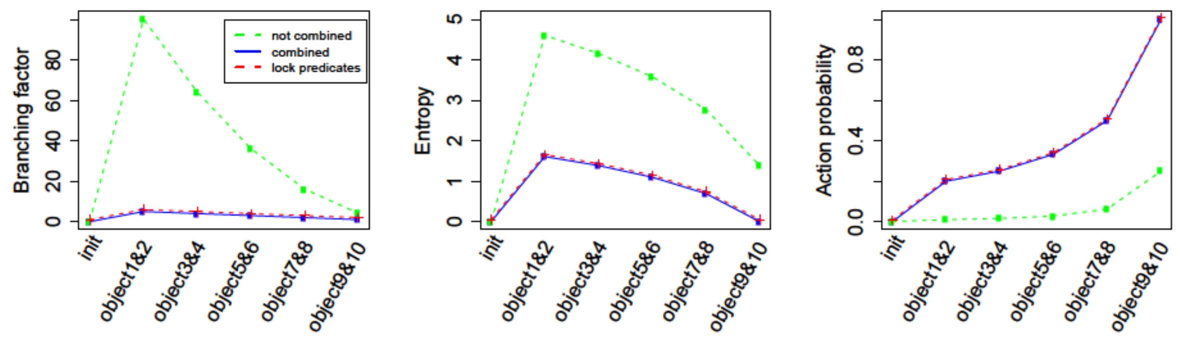


Figure 4.18: Model dynamics for combined objects. A model that does not use any objects combination is shown with a dashed green line. A model that uses lock predicates to combined objects is shown with a dashed red line. Finally, a model that uses combined objects is shown with a sold blue line. The red line is slightly shifted so that it does not overlap with the blue line.

only difference between using lock predicates and combined objects, is the number of objects available. Below we consider the influence of these mechanisms in a general case.

Proposition: *Using mechanisms for combining objects reduces the number of plans, states, operators, as well as the branching factor compared to a plain model. Using lock predicates produces the same model dimensions as using combined objects with the exception of the number of objects. In that case, the objects in a model using combined objects is higher than that of using lock predicates.*

Assume we have a model B_1 with actions $A := \{a_{ij}\}$ with $i := \{1,...,n\}$ and $j := \{1,...,n\}$ where n is the number of objects available. This means that an action can be parameterised with any combination of two available objects, also with repeating objects. Then the model has n^2 actions. Let assume that in order to reach the goal, all the available objects have to be involved in an action execution once and without any object appearing twice. This means that we can have a plan with length at most n elements in the case where an action contains the combination of two equal elements e.g. a_{11}[11]. This will result in $n!$ plans. However, all objects can appear in the plan also when it is less than n actions long. This is in the case when some of the involved actions contain two different objects. For example, the plan can have the length of $(n-1)$ when two of the different objects are combined in one action a_{ij} with $i \neq j$, and the remaining $(n-2)$ objects appear as two identical objects i in an action a_{ii}. In that case we will have $n!/(n-r)!$ combinations of actions with identical objects where r represents the $(n-2)$ actions with repeating objects. Each of these combinations then has to be multiplied by 2 as the remaining action can have either the order a_{ij} or a_{ji}. The same procedure can be applied for a plan that has more than two different objects combined together. In that case the number of combinations with identical objects will be equal to $(n-4), (n-6), ...(n/2)$ and each of these combinations have to be multiplied by 2 for the last pair of two different objects.

The procedure is repeated until the length of a plan is either $n/2$ when the number of objects is even, or $(n+1)/n$ when the number of objects is odd. In the case of odd number of objects, the procedure of calculating the number of possible plans is completed, as the shortest possible plan will have one action with identical elements and $(n-1)/2$ actions with pairs of

[11]That is a result from the lack of constraints forbidding the same object to appear twice in the same action. This results in the preconditions for executing such action being satisfied, as the object has not been involved in an action before.

two different elements.

In the case of even number of objects, the shortest plan will however contain $N/2$ actions with pairs of only different objects. To compute the combinations of the shortest plan then the number of objects n is multiplied by the available number of objects $(n-2)$ when the first pair of objects is fixed, times the number of objects $(n-4)$ when also the second pair of objects is fixed, and so on, until the last not fixed pair of objects is reached which produces two more possible combinations.

Finally, a special case arises when the number of objects equals 2. In that case there is no plan that contains action with identical objects and action with different objects. Then we have only 2! combinations in the case only identical objects are used, and additional 2! combinations when the objects in the action differ.

The procedure of calculating the number of plans can be then represented by the three cases described above and summarised in Formula 4.64.

$$Pl_{B_1} = \begin{cases} n! + \sum_{i=2}^{(n+1)/2} \frac{n!}{i!} 2 & \text{if odd and } n > 2 \\ n! + \sum_{i=2}^{n/2} \frac{n!}{i!} 2 + \prod_{i=1}^{n/2} 2i & \text{if odd and } n > 2 \text{ ,} \\ n! + \prod_{i=1}^{n/2} 2i & \text{if } n < 2. \end{cases} \tag{4.64}$$

Here i is the length of the plan. It is the result of simplifying the denominator of $n!/(n-(n-i))$ which represents choosing $r = (n-i)$ elements from a set of n elements (see Formula F.5 in Appendix F for more details about permutation of elements of a set).

The number of operators Op_{B_1} corresponds to the available actions and is

$$Op_{B_1} = n^2. \tag{4.65}$$

Assuming we need at most one predicate per object, then the number of predicates will be equal the number of objects.

$$Pr_{B_1} = Ob_{B_1}. \tag{4.66}$$

The number of objects will be

$$Ob_{B_1} = n, \tag{4.67}$$

and the length of the goal distance will be equal to the length of the shortest plan. In this case, as each action can be parameterised with two objects, it will be half the number of objects in the case we have even number of objects and half the number of objects plus one, in the case of odd number of objects.

$$G_{B_1} = \begin{cases} n/2 & \text{if even} \\ (n+1)/2 & \text{if odd.} \end{cases} \tag{4.68}$$

Furthermore, given that there are n grounded predicates and no additional constraints, then the number of states in the model will be equal to all subsets of these predicates. With other words,

$$S_{B_1} = 2^n. \tag{4.69}$$

The branching factor then in the worst possible case will be equal to the number of available operators, namely

$$Br_{B_1} = n^2. \tag{4.70}$$

Assume also there is a model B_2 that makes use of lock predicates, with $A := \{a_1, ..., a_{n/2}\}$ where n is the number of objects available. Each action takes as parameters two objects and there is only one combination possible for each two objects. Then the number of valid plans Pl_{B_2} can be calculated by

$$Pl_{B_2} = \begin{cases} (n/2)! & \text{if even} \\ \left(\frac{(n+1)}{2}\right)! & \text{if odd.} \end{cases} \tag{4.71}$$

The number of operators Op_{B_2} corresponds to the available actions and is

$$Op_{B_2} = \begin{cases} n/2 & \text{if even} \\ (n+1)/2 & \text{if odd.} \end{cases} \tag{4.72}$$

The number of predicates then will be equal to the number of available objects, namely

$$Pr_{B_2} = n. \tag{4.73}$$

The number of objects will be

$$Ob_{B_2} = n, \tag{4.74}$$

and the length of the goal distance will be equal to the length of the shortest plan, namely

$$G_{B_2} = \begin{cases} n/2 & \text{if even} \\ (n+1)/2 & \text{if odd.} \end{cases} \tag{4.75}$$

Furthermore, assuming that there are no repeating combinations of objects, then after the execution of an action two predicates will have the same value (true or false). This will reduce the possible predicates combinations from 2^n to $2^{n/2}$.

$$S_{B_2} = \begin{cases} 2^{n/2} & \text{if even} \\ 2^{(n+1)/2} & \text{if odd.} \end{cases} \tag{4.76}$$

The branching factor then in the worst possible case will be equal to the number of available operators,

$$Br_{B_2} = \begin{cases} n/2 & \text{if even} \\ (n+1)/2 & \text{if odd.} \end{cases} \tag{4.77}$$

Assume also there is a model B_3 that uses combined objects, with $A := \{a_i\}$ with $i := \{1,...,n/2\}$ where n is the number of objects available, and where there are $\frac{n}{2}$ additional combined objects with each combined object representing a unique action pair. Then the number of valid plans Pl_{B_3} can be calculated by

$$Pl_{B_3} = \begin{cases} (n/2)! & \text{if even} \\ \left(\frac{(n+1)}{2}\right)! & \text{if odd} \end{cases} \tag{4.78}$$

where the number of operators Op_{B_3} corresponds to the available actions and is

$$Op_{B_3} = \begin{cases} n/2 & \text{if even} \\ (n+1)/2 & \text{if odd.} \end{cases} \tag{4.79}$$

The number of predicates then will be equal to the number of single objects, as each complex object is just a mapping to two single objects.

$$Pr_{B_3} = n. \tag{4.80}$$

The number of objects in this case will be the n single objects plus the additional $n/2$ combined objects,

$$Ob_{B_3} = n + \begin{cases} n/2 & \text{if even} \\ (n+1)/2 & \text{if odd} \end{cases} \tag{4.81}$$

and the length of the goal distance will be equal to the length of the shortest plan, namely

$$G_{B_3} = \begin{cases} n/2 & \text{if even} \\ (n+1)/2 & \text{if odd.} \end{cases} \tag{4.82}$$

Like in the case of B_2 the minimal state space of the model S_{B_3} is

$$S_{B_3} = \begin{cases} 2^{n/2} & \text{if even} \\ 2^{(n+1)/2} & \text{if odd.} \end{cases} \tag{4.83}$$

The branching factor then in the worst possible case will be equal to the number of available operators,

$$Br_{B_3} = \begin{cases} n/2 & \text{if even} \\ (n+1)/2 & \text{if odd.} \end{cases} \tag{4.84}$$

Comparing the three models, it can be seen that B_1 which does not use combined objects or

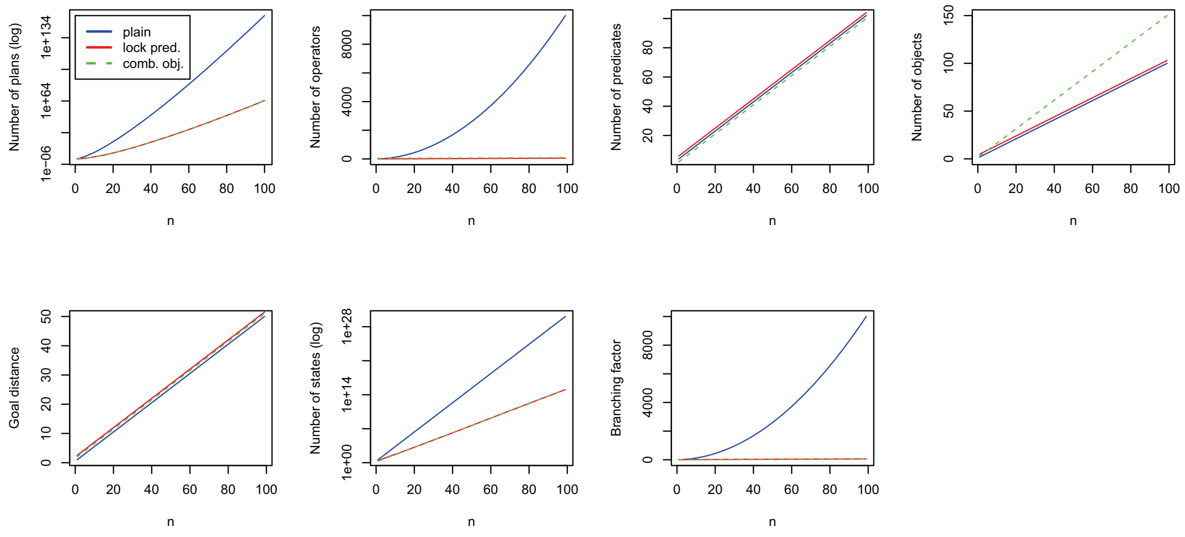


Figure 4.19: General model dynamics for combined actions compared to plain model. The solid blue line represents the behaviour of plain model, the solid red line shows the behaviour of a model that uses lock mechanisms to combine objects, and the dashed green line uses combined objects. The model complexity for increasing number of actions was plotted (1 to 100). The number of states and number of plans are plotted on a logarithmic scale.

lock predicates has the largest number of plans which in any of the three cases is higher than $n!$. Using lock predicates, the number of plans is reduced to $(n/2)!$ in B_2, and is equivalent to B_3. The same proportions are reflected in the state space and number of operators, as well as the branching factor. On the other hand, the usage of combined objects increases the number of objects compared to the remaining two models, as it introduces additional combined objects. Finally, the goal distance for the three models stays the same. This indicates that using lock predicates or combined objects reduces the model complexity as it removes redundant object combinations. The usage of combined objects compared to lock predicates shows the same model complexity. However, in the case of B_3 there is the disadvantage of the additional combined objects that are needed. The models' dynamics for increasing number of actions can be seen in Fig. 4.19.

Applicability: As already mentioned, the implementation of combined objects or lock predicates for combining objects can be used in situations where a lot of objects are available. These objects make the model state-space to explode due to the high number of possible combinations between objects. When the objects that can be combined together are already known, it can decrease the model complexity as only the relevant combinations are considered at all. The difference between using combined objects and using lock predicates is that in the first case only one parameter is passed to the action template, whereas in the second case there are two parameters. On the other hand, the first model artificially increases the number of available objects which is avoided in the case with the lock predicates. For that reason, it is preferable to use the lock predicates, as they require less implementation effort and do not introduce any additional elements to the environment.

4.9 Phases

Motivation: In situations where the model is too complex and creates huge state-space that often leads to decreasing the actions probabilities and inferring the incorrect action, this could be avoided to an extend by defining phases. Such phases force the valid model sequences to pass through a single state at some point during the model execution making all plans that do not visit this state invalid. Especially, in cases where it is known that the model always visits a certain state, phases can be extremely useful tools. Here we discuss how such modelling mechanisms can be implemented and what consequences they have on the model.

Structure and implementation: Phases are nothing more than actions that contain certain constraints in their preconditions and effects. After compiling, they force the model to always pass through a given state at a certain point of the model execution. Fig. 4.20 shows the idea

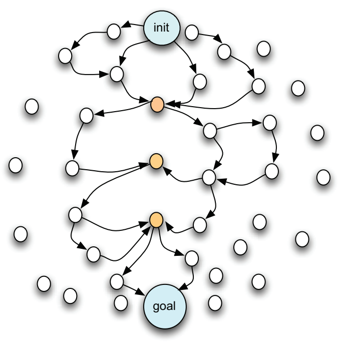


Figure 4.20: Structure of phases. In it the states representing the phases are coloured in orange. All execution paths that pass through all of the orange states are valid, while the rest are discarded.

behind the phases. It can be seen that states leading to the phases are connected with arrows – these are the paths that are valid and lead to the goal state. All states that do not lead to the phases are no longer valid thus the compiler removes them.

The implementation of phases can be seen in Appendix E in Fig. E.12 (page 230). It contains the action *do-something* that generates five operators and that after grounding defines two phases in the model (before and after action 3). It can also be seen that the abstract definition of the action defines the boundary between the two phases after the execution of action number 3. This is done by the predicate *can-do-something* as for the first three actions it is set to false while it is set to true for the remaining actions that were previously not allowed. The same effect can be achieved by defining the phase boundary as a separate action, however it will result in more operators as there will be more action templates.

Consequences: The usage of phases influences the model parameters and its dynamics as it reduces the number of available states. To illustrate that, a model using phases was compared to one that did not use such. Fig. 4.7 shows the model parameters in the case when phases were

Table 4.7: Parameters of phases implementation. The values for the phases are based on the implementation on Fig. E.12 (page 230). Those for the plain model are based on the same implementation with the difference that the *when* clause in the action's precondition is missing.

Parameter	Value (phases)	Value (no phases)	Description
# operators	5	5	grounded actions after the model compilation
# predicates	10	10	grounded predicates after the model compilation
# objects	5	5	objects in the model
# states	12	32	state-space of the model
# valid plans	4	120	valid plans leading from the initial to the goal state
goal distance	5	5	distance from the initial to the goal state
max. branching factor	3	5	maximum number of possible actions from a state
final log likelihood	-2.48491	-4.78749	final log likelihood of the execution sequence

used and then the same model without phases. It can be seen that introducing a phase boundary in the action itself did not increase the number of operators, or the objects needed. However, the fact that the phase restricts the model to just a fraction of the available states, reduced the state-space from 32 to 12 states. Furthermore, it greatly reduced the number of valid plans (form 120 to only 4) which directly results in increasing the correct action's probability. The goal distance was not affected as the phase boundary is integrated in the actions that are anyway needed for executing the given plan, while the branching factor became twice smaller than without phases. This also affected the final log likelihood making the execution sequence more probable than in the case without phases.

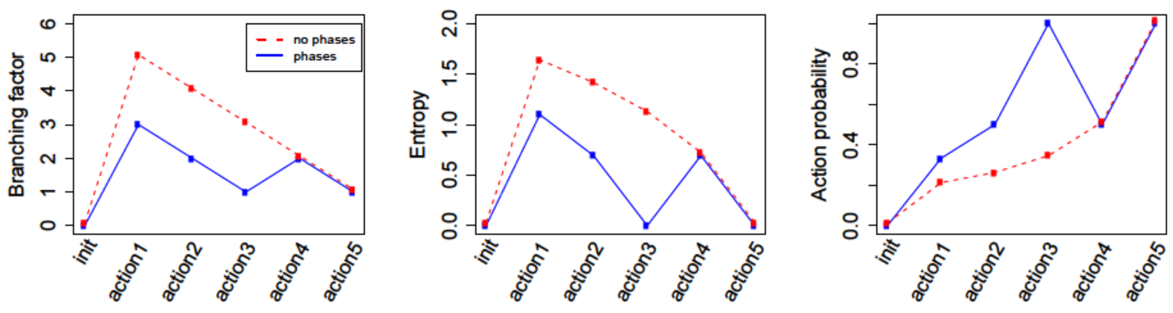


Figure 4.21: Model dynamics for phases. The model that uses phases is represented with a solid blue line. The one without phases is shown with a dashed red line.

This is also reflected in the model dynamics shown in Fig. 4.21 where the influence of the boundary between the two phases can easily be seen at action 3. There the model branching factor drops to only one available action, reducing the entropy to 0 and increasing the action's probability to 1. It can also be seen that the constraints introduced in the phase control the number of executable actions throughout the model. Their number is always smaller than the branching factor of the model not using phases. The same is reflected in the model entropy where the phases take care of lower entropy throughout the model execution. This also makes the executed plan more probable than in the case without phases, thus increasing the probability of selecting the correct action.

General case: Above we showed with a dummy example that the usage of phases in a model decreases the model size in terms of state space and number of valid plans. Below we discuss the influence of phases in a general model.

Proposition: *By using phases the model complexity compared to that of a plain model decreases in terms of number of plans, number of states, and branching factor.*

Assume we have a model B_1 with actions $\{a_1,...,a_n\}$ where n is the number of objects available. If we assume the worst case scenario in which each instance of action a can be executed sequentially in any order without repetition of an instance, then the number of valid execution combinations will be $n!$.

$$Pl_{B_1} = n!, \tag{4.85}$$

where the number of operators Op_{B_1} corresponds to the number of grounded actions and is

$$Op_{B_1} = n. \tag{4.86}$$

If we assume that the abstract action needs at least two predicates with which it can be represented (one for allowing the action execution and another for indicating the execution), then when grounding the action, it will result in $2n$ grounded predicates.

$$Pr_{B_1} = 2n \tag{4.87}$$

If we assume that in the best case there is one object per action instance, namely the object with which the instance is grounded, then the number of objects will be

$$Ob_{B_1} = n, \tag{4.88}$$

and the length of the goal distance will be equal to the length of the shortest execution plan, namely

$$G_{B_1} = n, \tag{4.89}$$

The state-space will then be computed based on the number of predicates. In this case, we have Pr_{B_1} with $2n$ predicates, and we know that whenever the predicate allowing action execution is true, the predicate indicating the action is being executed is false. This means that there will not be 2^{2n} states but rather 2^n as each two predicates are dependent on each other.

$$S_{B_1} = 2^n. \tag{4.90}$$

The branching factor then in the worst possible case will be equal to the number of available actions at a time, namely

$$Br_{B_1} = n. \tag{4.91}$$

Assume also there is a model B_2 with actions $A := \{a_1,...,a_l,b_1,...,b_q,...,m_1,...,m_p\}$ where $l+q+...+p = n$ is the number of available actions and where $a \in L$, $b \in Q$, $m \in P$ and $L \subset A$, $Q \subset A$, $P \subset A$ and $L \cup Q \cup ... \cup P = A$. Additionally, actions a_1 to a_{l-1} have to be executed before a_l, and, all actions b can be executed only after all the a actions are executed with b_1 to b_{q-1} executed before b_q and so on. The actions a_l and b_q are then the boundaries between phases in the model. Then the number of valid plans Pl_{B_2} can be calculated by calculating the number of possible combinations for actions a_1 to a_{l-1}, then multiplying the results by the number of combinations for b_1 to b_{q1}, and finally, by the number of all combinations for m. The reason for calculating only $l-1$ and $q-1$ combinations is that the execution order of the actions a_l and b_q is fixed. On the other hand, the last subset of actions does not contain a phase boundary action thus actions m_1 to m_p can be executed in any order.

$$Pl_{B_2} = (l-1)!(q-1)!...p!.$$

If I is an index set that indexes the subsets of A P, namely all subsets except for the last subset P, then we can write the number of plans in the following manner:

$$Pl_{B_2} = p!\prod_{i\in I}(|X_i|-1)!, \tag{4.92}$$

where $|X_i|$ is the number of elements for each subset in A.

The number of operators Op_{B_2} corresponds to the available actions and is

$$Op_{B_2} = l+q+...+p = n. \tag{4.93}$$

Like in the case of B_1, here we also make use of two predicates per action.

$$Pr_{B_1} = 2n \tag{4.94}$$

The number of objects will correspond to the number of grounded actions, namely

$$Ob_{B_2} = n, \tag{4.95}$$

and the length of the goal distance will be equal to the length of the shortest plan:

$$G_{B_2} = n. \tag{4.96}$$

Furthermore, assuming that at least n actions are needed to reach the goal, and that the model has always to pass through certain actions (a_l, b_q), then the number of states in the model can be computed by calculating the number of states in the different phases and then adding them up.

$$S_{B_2} = 2^l + 2^q + ... + 2^p = \sum_{i\in I} 2^{|X_i|}, \tag{4.97}$$

where $|X_i|$ is the number of elements in each subset of A. The branching factor then in the worst possible case will be equal to the number of available operators in the given phase. That means that if the number of operators in a phase is denoted by $Op(y)$, where y is the phase, then

$$Br_{B_2} = \arg\max_{y_{max}} Op(y_{max}) := \{y_{max}|\forall y : Op(y) \leq Op(y_{max})\} \tag{4.98}$$

Whereas model B_1 has $n!$ plans, B_2 reduces them to $(l-1)!(q-1)!...p!$ with $l+q+...+p = n$. The number of operators in both cases is n as is the goal distance, and the number of objects. The minimal state space is also reduced – while B_1 has 2^n states, B_2 has $2^l+2^q+...+2^p$. In the case of B_2 also the branching factor is smaller as in the worst case it is always equal to a subset of the n available actions, while in B_1, the maximum branching factor is n. The above equations indicate that using phases reduces the model complexity, as the model can execute

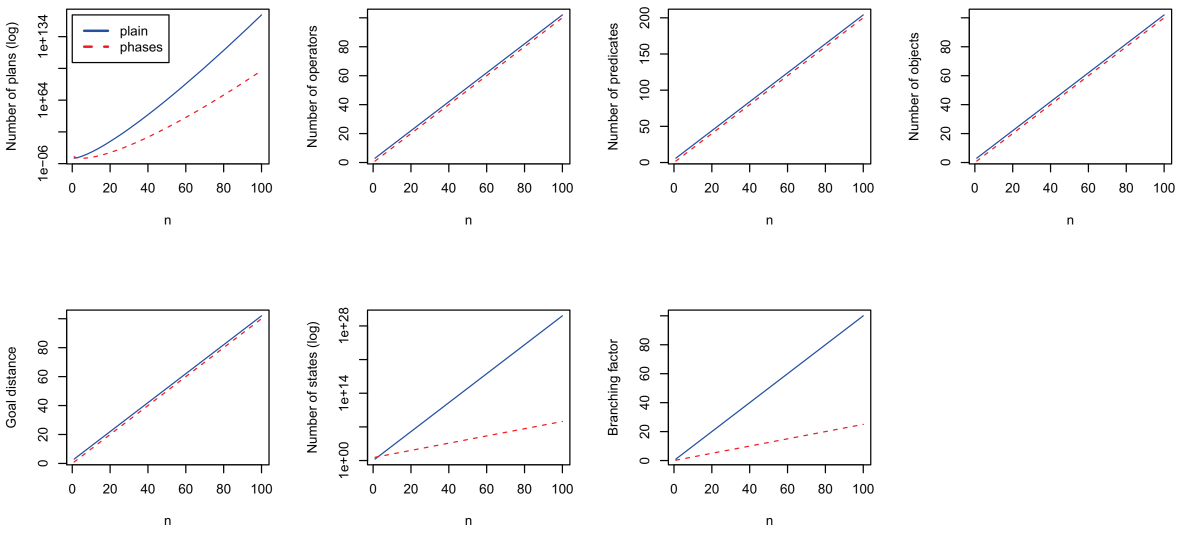


Figure 4.22: General model dynamics for a model using phases compared to a plain model. The solid blue line represents the behaviour of the plain model, while the dashed red line shows the behaviour of a model that uses phases. The model complexity for increasing number of actions was plotted (1 to 100). The number of phases here is fixed to 4 phases with equal number of actions in each of them.

certain actions only in different subgraph of the model. This can also be seen in Fig. 4.22 which shows the model parameters for increasing number of actions.

Applicability: Phases are applicable when modelling complex problems that generate huge state-spaces. This often leads to decreasing the probability of the correct actions during inference. Phases, for example, can be introduced in the cooking problem. Due to the high behaviour variability, it will not be possible to introduce real phases that make the model pass through only one state. However, semi-phases can be used that force the model to pass through just a few states during the phase, thus increasing the probability that the correct action will be inferred.

4.10 Modelling the three problems – revisited

Chapter 3 introduced the intuitive CCBM solutions of the three modelling problems. Here the models are revisited and improved based on the model analysis performed in the previous chapter. This is done by incorporating relevant patterns from the CCBM modelling toolkit.

4.10.1 3-person meeting

The 3-person meeting model already contained abstract actions, implicit macro structures, multi-agent action synchronisation, and a simple type hierarchy.

In this section it was improved by introducing an agenda that describes the order in which the presentations can be executed. That is nothing more than an implementation of the phases modelling mechanism described in the previous section. The mechanism was introduced based on the results from the intuitive model which showed that the model sometimes confuses the user presently executing a presentation action. Fig. 4.8 shows the differences in the two model parameters. The first is the intuitive model presented in Chapter 3.3, while the second is the improved version that contains phases. It can be seen that the phases reduced the number of

Table 4.8: Comparison of model parameters for the intuitive and the improved models.

Parameter	**Intuitive model**	**Improved model**
# operators	88	72
# predicates	72	97
# object types	4	4
# persons	3	3
# locations	7	7
# activities	6 + 10 single-user	6 team + 10 single-user
# states	5568	5110
# valid plans	3515	2714
# hierarchy level	3	3
# goal distance	48	48
# max. branching factor	9	9

grounded actions (from 88 in the intuitive model to 72 in the improved) to the expense of increasing the number of predicates (from 72 to 97). That is due to the additional predicates needed for expressing the constraints of the phases. The type system remained the same as well as the involved persons, locations, objects and activities to be recognised. The state space however was reduced from 5565 in the intuitive model to 5110 in the model with phases. The additional constraints also reduced the number of valid plans thus removing execution sequences that are not plausible for the scenario. Finally, the shortest goal distance and the maximum branching factor both remained the same.

From the values above it can be seen that the additional constraints managed to decrease the model size. The hypothesis then was that decreasing the model size can improve its performance. Fig. 4.23 shows the comparison between the performance of the two models. The results of the intuitive model are shown in red while those of the improved model – in blue. It can be seen that there is slight improvement in the latter's performance. It is most visible in the case of user one and on a team level. This can be explained by the fact that in the model without agenda the inference engine sometimes chose the wrong user to begin the first presentation. In the second case, this artefact is not present as the phases allowed just the right user to start first. Additionally, there is a noticeable improvement in the performance of the model on the long meeting (the last dot in the plots) where the accuracy of the first two users was increased from about 95% to about 98%. It can also be seen that the model precision was improved on a team level, namely the model ability to correctly recognise positive instances. On the other hand, the model ability to correctly recognise negative instances (represented by its specificity) didn't improve. A noticeable improvement is present only in the case of the long meeting (the last dot in the specificity plots) and especially regarding the second user, where it increased with about 1%. This is a noticeable difference for specificity as this measurement unit usually stays very high.

To test the significance of the differences between the two models, also a Wilcoxon test was performed[12] [154]. The results from the test can be seen in Appendix I. Assuming significance confidence of 95% or more, the *p-value* of the Wilcoxon test should be less or equal to 0.05. From Table I.1 (page 244) it can be seen that the difference in the models results is significant for all three users and the team. This indicates that the introduction of patterns in the model improved the model accuracy and that one can conclude with 95% confidence or more that the obtained values were not just due to chance. On the other hand, the results show that the model precision was insignificant and that there is significant difference only in the case of the team behaviour. Additionally, the difference in the specificity was insignificant in the case of

[12]For more information on Wilcoxon test, see Appendix I.

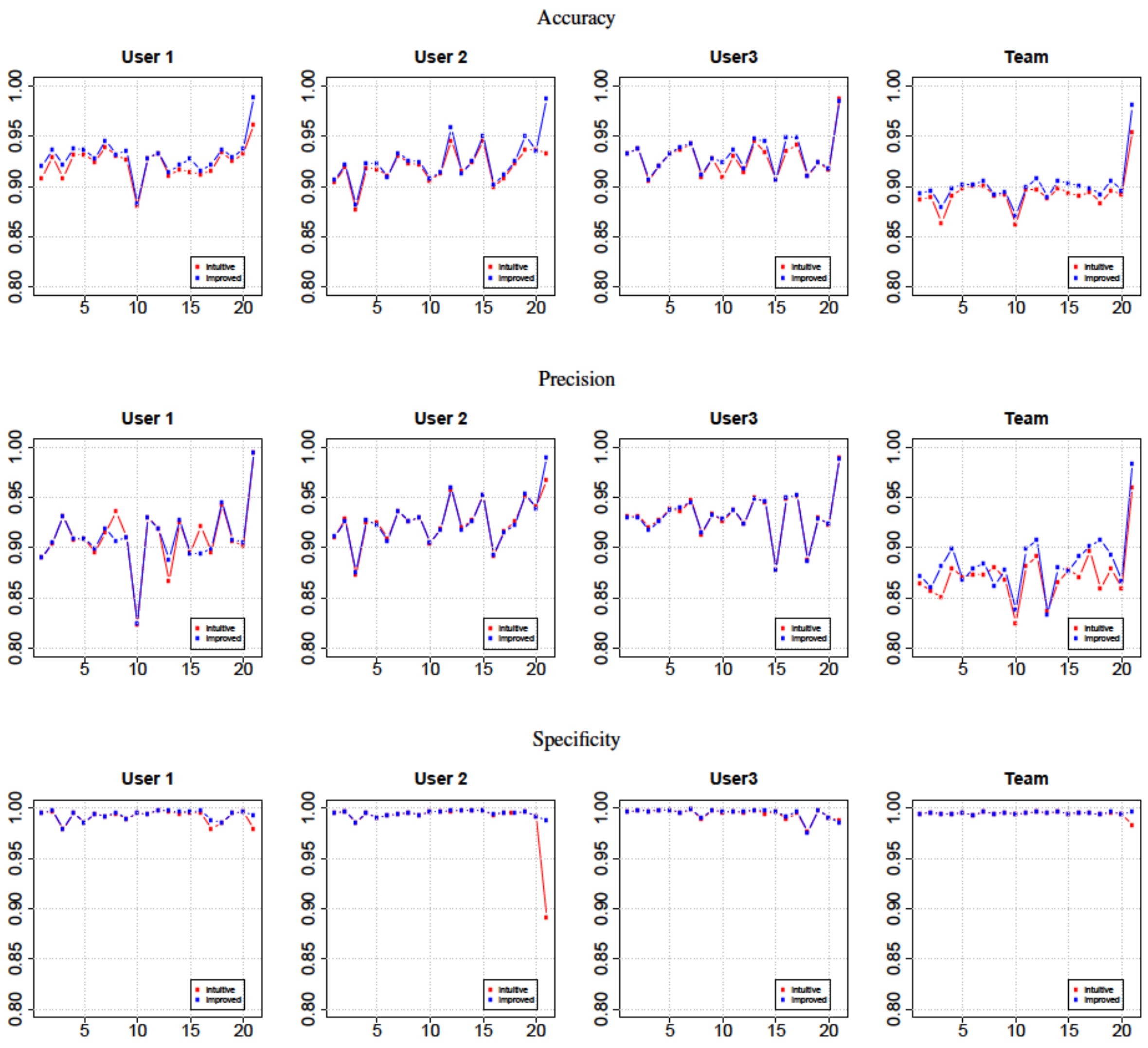


Figure 4.23: Performance comparison of the two meeting models. The results of the intuitive model from Chapter 3 are given in red, while that of the improved in blue. For each of the three users and the team the accuracy, precision, and specificity are plotted. For each of the 21 meetings the average of 50 runs is taken.

the third user, while for the remaining two users and the team, the ability to recognise negative instances were improved.

Furthermore, like in Chapter 3 Friedman test was applied to the results from the second model that contains patterns. This was done in order to test whether the model performance significantly differs given the different datasets. The results from the Friedman test can be found in Table H.2 (page 241) where the accuracy, precision, and specificity were tested separately; and in Table H.4 (page 242) where all performance metrics were taken together as an input for the test. It can be seen that difference in the model performance was significant when all the metrics were taken together as in input, and that it increased compared to the intuitive models from Chapter 3 (see Table H.4). This stands to show that the bias-variance dilemma is also valid here, and that with increasing model performance, the model becomes more sensitive to changes in the data. This is, of course, no surprise as this phenomena is common for any problem that deals with data. On the other hand, when looking at the separate performance metrics, one sees that the differences in accuracy were insignificant. This stands to show that the introduced patterns in this case reduced the results variations. This is to no surprise as the introduced agenda reduces the model ability to make errors thus producing similar recognition results for the different datasets. This however, point once again toward the presence of model overfitting.

4.10.2 Cooking task

The cooking task model already contained abstract actions, implicit macro structures, implicit repetition, a complex type hierarchy, and combined objects.

In this section it was improved by introducing five phases that limited the number of actions which can be manipulated at a given location, or the phase at which they can be manipulated. This was introduced based on the conclusion that most of the actions can be executed with all available objects, so it is difficult to distinguish an action based just on the objects being observed[13]. Additionally, as the repeating actions were modelled to allow infinite repeating[14], a limit of repetitions was introduced based on an educated guess about how many times an action can be executed. The resulting model parameters compared to those of the intuitive

Table 4.9: Cooking task model parameters.

Parameter	Intuitive model	Improved model
# operators	111	110
# predicates	129	159
# object types	23	23
# persons	1	1
# locations	4	4
# objects	10	10
# hand locations	6	6
# activities	16	16
# states	$>$ 657 000 000	$>$ 657 000 000
# valid plans	$>$ 687 343 670	$>$ 687 000 000
# hierarchy level	4	4
# goal distance	approximately 80	approximately 80
# max. branching factor	10 (OPL), 12 (OL), 24 (O), 32 (PL), 60 (L)	10 (OPL), 11 (OL), 18 (O), 30 (PL), 58 (L)

model can be seen in Table 4.9. It can be seen that the phases actually increase the number of predicates while there is almost no difference in the number of operators. The state space also remains huge and it was impossible to calculate the exact number of states. Thus it is also impossible to see if there is difference in the state-space size. Due to the fact that the state space was not calculated, the exact goal distance was also not calculated as that can be done only after the whole state space is analysed[15]. However, there is some difference in the number of possible actions from a given state. In the case when all observations are involved there is no difference, while when the places are removed the branching factor decreases with one, and in the case when places and locations are used, or just locations it decreases with 2. This is

[13]This was observed in the case where only the objects were used as observations (see Fig. 3.16) in Chapter 3.

[14]Due to the fact that it was not known beforehand how many times the action was repeated.

[15]However, even in this case the goal distance is used as a heuristic for the action selection. This is done by approximating the distance to the goal based on landmarks in the model. These landmarks are identified based on the predicates that define the goal state. Then backward search is done for all preconditions and effects that lead from the goal to the initial state and the predicates involved are called landmarks. A more detailed description of landmarks can be found in [116, 115].

not much difference but it is indication that the patterns have influence on the state space. The situation changes in the case when only objects are used. In that case the maximum number of possible actions decreases from 24 to 18. This can be explained by the fact that the phases contain information that is otherwise incorporated in the places and locations observations. Thus the phases will have noticeable influence on the model size only when the observations containing this information are removed.

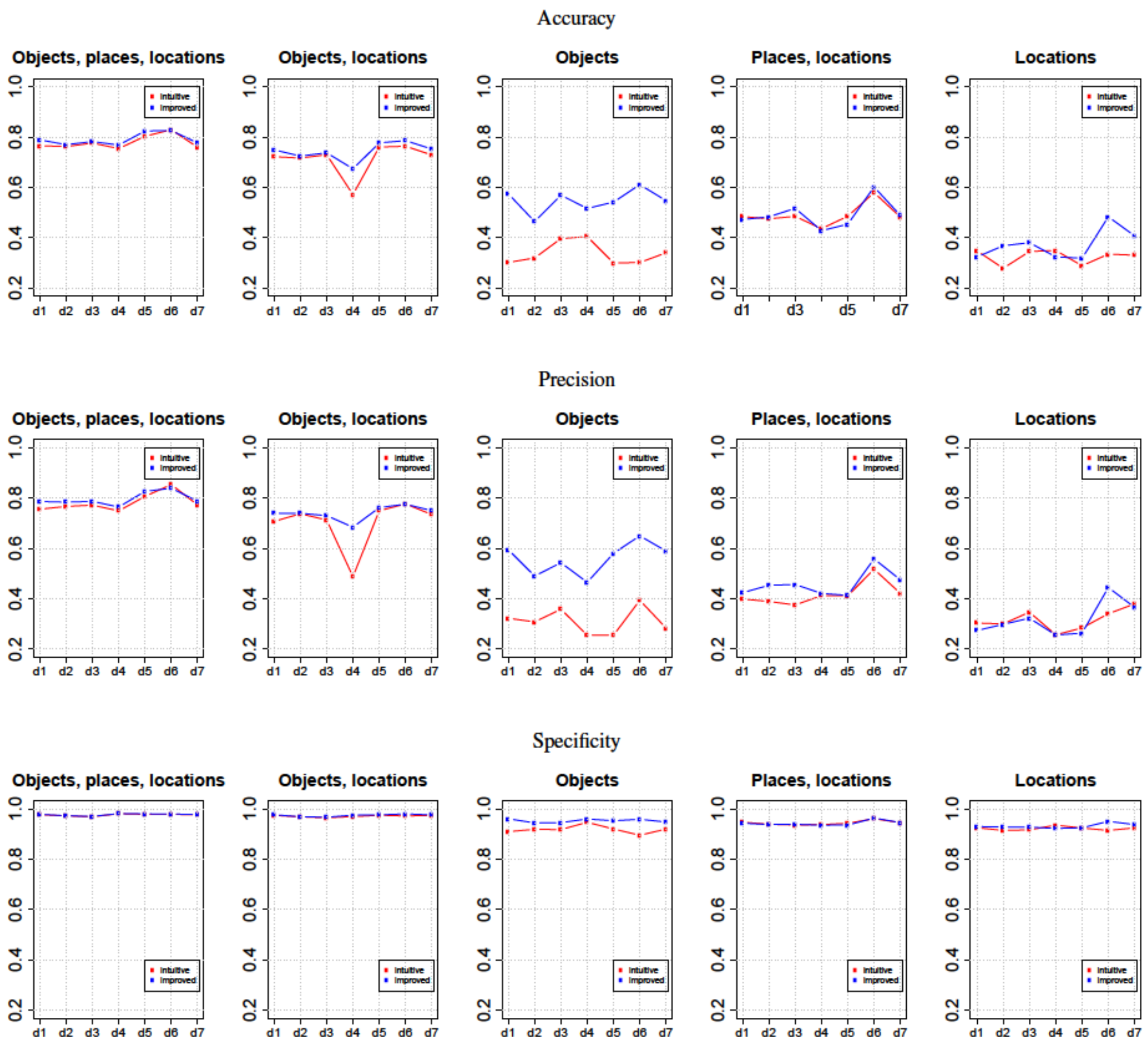


Figure 4.24: Performance comparison of the two cooking task models. The results of the intuitive model from Chapter 3 are given in red, while that of the improved in blue. For each of the different observation combinations the accuracy, precision, and specificity are plotted. For each of the 7 experiments the average of 50 runs is taken.

The difference in the models' performance is reflected in Fig. 4.24 where the accuracy, precision and specificity for the different observations are shown. It can be seen that in the case of having objects, places and locations as observations, the performance for the two models is comparable. This is explained by the fact that the observations are so detailed that the additional constraints of the improved model do not bring anything to its performance. In the case where objects and locations are used as observations, there is a noticeable improvement of the accuracy and precision for user 4. This means that the introduced constraints provide a better explanation for the user behaviour. On the other hand the specificity, namely the model ability to detect negative instances was not improved.

Noticeable difference in the models' performance can be seen in the third case – when only objects are used as observations. While the intuitive model performed with 30%-40% accuracy, the introduction of phases improved it to 50%-60%. This shows that this modelling mechanism is indeed able to reduce the number of possible actions in such a way that the inference engine is able to find a much plausible explanation of the user behaviour. This is reflected not only in the model accuracy but also in its precision and specificity, which indicates that the applied patterns are able to improve the overall model performance.

The situation somehow changes in the case of not so strong observations. When places and locations are used as observations, it can be seen that the intuitive model performed slightly better in three of the datasets (in the case of model accuracy). This can be interpreted as there is no influence of the patterns on situations where the objects are not involved as observations. It can be explained by the fact that the phases are connected to the objects the model is reasoning about and in a loose way can be looked upon as a mechanism for incorporating observation model information into the system model. In the last case where only the locations are taken as observation, the improved model is generally showing better recognition rate than the intuitive model. This once again can be explained by the information encoded in the phases as they limit certain actions to specific places which reduces the number of possible places accessible at a given location. On the other hand the model ability to recognise positive instances that are correctly recognised (its precision) is generally better for the intuitive model.

The results show that the phases are preforming better in situations where the additional information encoded in them is not present in the sensor infrastructure. The significance of the results is presented in Table I.2 (page 244). Assuming the 95% confidence, it can be seen that the difference in model performance for the objects, places and locations was significant. Thus we can assume that any changes in the model behaviour were due to the patterns and not just to chance. The same could be concluded for the objects and locations, and for the objects as observations. This indicates that all cases in which objects were involved as observations had significant improvement in the performance. It is however not the case for the places and locations, and only for the locations as observations. For the places and locations there is significant difference in the models precision but not in its accuracy and specificity. On the contrary, the p-values are so high that we can easily conclude that any difference in performance was due to chance alone. In the case of locations it can be concluded that the model accuracy changed based on the introduced patterns but its precision and specificity were insignificant.

Furthermore, like in Chapter 3 Friedman test was applied to the results from the second model that contains patterns. This was done in order to test whether the model performance significantly differs given the different datasets. The results from the Friedman test can be found in Table H.3 (page 241) where the accuracy, precision, and specificity were tested separately; and in Table H.4 (page 242) where all performance metrics were taken together as an input for the test. It can be seen that difference in the model performance was significant for almost all the calculated metrics, and that it increased compared to the intuitive models from Chapter 3. This stands to show that the bias-variance dilemma is also valid here, and that with increasing model performance, the model becomes more sensitive to changes in the data. This is, of course, no surprise as this phenomena is common for any problem that deals with data. It however, points out at the fact that one has to find the middle ground between bias and variance, or with other words between the ability to produce accurate results, and that to predict new data. The difference in the model performance however, was insignificant in the case of specificity which can be explained with the fact that the increase in the model performance, also increases the model ability to recognise negative instances, thus making the specificity more robust to variations in the data.

4.10.3 Office scenario

The office scenario model already contained abstract actions and implicit macro structures. In this section it was enhanced by introducing lock mechanisms for the users' agenda. This was due to the fact that the users act in an unsynchronised manner, so synchronisation patterns will not be applicable to the problem. On the other hand, it was shown that the model has incredibly huge state-space for such small problem. In attempt to reduce the state-space, the users' agenda listed the activities the user plans to execute. The agenda here can be looked upon as a single phase that indicates the constraints related to the user. The parameters of the resulting model can be seen in Table 4.10. It shows that this modelling mechanism seriously

Table 4.10: Office model parameters

	Intuitive model			Improved model		
Parameter	1 user	2 users	3 users	1 user	2 users	3 users
# operators	50	100	150	43	50	50
# predicates	28	52	76	19	28	28
# object types	3	3	3	3	3	3
# persons	1	2	3	1	2	3
# locations	7	7	7	7	7	7
# objects	3	3	3	3	3	3
# activities	5	5	5	5	5	5
# states	21 504	9 633 792	$>$ 880 000 000	106	3064	962 880
# valid plans	43329	48 373 761	$>$ 48 373 761	183	3217	4 590 145
# hierarchy level	2	2	2	2	2	2
# goal distance	14	33	$>$ 33	14	15	29
# max. branching factor	6	16	$>$ 16	7	8	21

reduces the number of operators and predicates. While in the intuitive model the number of operators increases exponentially with each new user, in the improved model it almost does not change as the model already knows which activities to ignore regarding a given user. The number of elements that are modelled, as well as the object hierarchy stay the same. However, there is a significant improvement in the state-space. While in the intuitive model there are 21504 states for the single user problem, in the improved there are only 106. And although the states increase significantly with increasing the number of users, it is still possible to compute the number of states for 3 users (in comparison to the more than 880 million states in the intuitive model). The same is reflected in the number of valid plans as well as the visibly smaller branching factor and the smaller goal distance (in the case of two and three users). This indicates that including simple non-synchronised and non-ordered agenda is able to significantly reduce the model size.

This also improves the model performance as it can be seen from Fig. 4.25. It shows that in overall the performance was noticeably better in the improved model. The same is reflected in the model's ability to represent the underlying behaviour (namely its precision) where in some of the cases the precision increases from about 30% to about 90%. Similar tendency can be seen in the model specificity with the exception of user 6. This indicates that the improved model, as well as the intuitive were both unable to capture negative results.

The results significance is presented in Table I.3. It shows that all three measurements had values under 0.05 which indicates that the changes in the performance were due to the

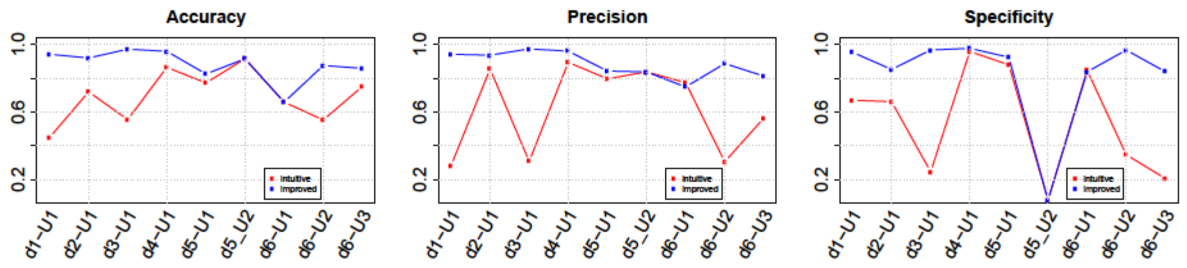


Figure 4.25: Performance comparison of the two office scenario models. The results of the intuitive model from Chapter 3 are given in red, while that of the improved in blue. For each of the users the accuracy, precision, and specificity are plotted. The average of 50 runs is taken.

introduced modelling mechanisms rather than to chance. From this, it can be concluded that even simple usage of lock predicates can have significant influence not only on the model size, but also on its performance. It also shows that the size of the model and the number of possible actions has influence on the model performance.

Finally, to test the performance variability across different datasets, Friedman test was performed. The results can be seen in Table H.4 in Appendix H (page 242). It shows that the performance significantly differed given the different datasets. It stands to show that the increased model performance resulted into higher results variance. This once again shows the problem of the bias-variance tradeoff. Namely, if we want to have high recognition rate, we are forced to reduce the model degrees of freedom, thus increasing the model variance, and vice versa. This points to the fact that one should be aware of this effect and carefully consider what exactly one wants from the model being developed – very accurate activity recognition, or ability to recognise large range of user behaviour, or maybe somewhere in between.

4.11 Discussion

Chapter 3 introduced the problem of the model's inability to recognise the correct actions in the case of large state spaces. This resulted in the modelling toolkit that provides different modelling patterns that could reduce the model complexity thus increase the model performance. One problem with reducing the complexity is the model's ability to recognise new variations of the behaviour. This is a result of introducing constraints that allow just subset of the logically valid actions to be applicable. This problem can be considered as a special case of the bias-variance dilemma [36, p. 466]. It states that there is always a trade-off between variance and bias: by increasing the model's flexibility to adapt to future data, its performance tends to decrease. On the other hand, when the model is able to correctly predict the true value, it tends to be more sensitive to changes in the data. This problem can also be noticed here as the behaviour variability decreases with introducing the modelling patterns but the model sensitivity to data variations increases. On the other hand it was shown that the models perform better when the complexity is reduced. It is obvious that the middle ground between these two phenomena has to be found. So the question arises: *When do we stop to reduce the model complexity?*

Having in mind that the ultimate reason for activity recognition is to assist the user in a certain way, the application has to be able to recognise the user actions to such degree that it is later able to accurately assist her. This means that the model should be able to provide high recognition rate. Meanwhile it should also be able to cope with the behaviour variability. The inability of the model to recognise behaviour just because it is executed in some other order, renders the model exactly as useless as when it is unable to recognise the activities with high accuracy. From this is obvious that we need the model to have high performance and still to be able to cope with differences in the user behaviour. One answer to this problem is that the patterns although reducing the model variability, still do not reduce it to the extent that it is able to recognise only the available datasets. For example, the meeting model after some patterns are applied, still contains almost 3000 different plans for executing the user actions which is much more than a system designer could model by hand. Especially having in mind the actions' granularity (basically the actions recognised are atomic actions) it is impossible to introduce constraints that will render the behaviour fully deterministic. In that sense, we can conclude that although the patterns reduce the model complexity, they still preserve some of the behaviour variability, thus finding the middle ground between bias and variance. Furthermore, the available amount of context information is not hindered by the reduction of the model complexity. This means that all the information one could reason about in a general model is still there in the reduced model with the added advantage that it is more accurate as it is restricted to more specialised cases.

Another aspect that deserves discussion is the patterns applicability to other rule-based formalisms. The patterns discussed in this chapter are, of course, presented in the concrete context of CCBM. However, the applied mechanism – namely the rules that guide these patterns – are universal for any rule-based formalism. They would yield similar complexity behaviour regardless if they are implemented e.g. in CCBM, PDDL, LISP[16], STRIPS etc. In that sense one could consider the CCBM patterns as an example of how they will affect a model in a more general case. This was also shown in the *General case* sections. The way in which these constraints are then implemented depends on the concrete formalism and its syntax. The important thing here is not how exactly or in which language they are implemented, but rather how they affect the model behaviour and why and where we can apply them in order to decrease the model complexity and produce desirable results.

4.12 Outlook

The chapter introduced the CCBM modelling toolkit. It is based on successful modelling practices employed in the models from Chapter 3 and on needs discovered during the modelling process. Its purpose is to improve the model performance by reducing the size of the model parameters.

The influence of the patterns on the model parameters and performance was experimentally shown by comparison of dummy examples. Later, the influence of the patterns was shown more formally by calculating the parameters for a general model containing the mechanisms in question. Furthermore, the models from Chapter 3 were enhanced with relevant patterns and parameters, and their performance was compared with that of the intuitive models from Chapter 3. The results showed that for the given use cases and models the mechanisms reduced the size of some model parameters and increased the model performance.

A future work that was left aside here, can be the separate performance evaluation for each pattern included in the model. Another aspect that deserves looking into is the structured incorporation of observation information into the system model. The effects of such incorporation were observed in the cooking task problem (with objects as observations). However, as the experiment was not intended to investigate the effects of such incorporation, it was not further pursued here.

[16]**LIS**t **P**rocessing Language (LISP) [137].

Chapter 5

Development Process for Symbolic Human Behaviour Models for Activity Recognition

"It may well be that intuition or artistic skill is largely the product of imitation and practice, yet this process of development must have a beginning."
William T. Morris

Chapter Summary: *This chapter looks at the state of the art in model engineering and discusses advantages and drawbacks to existing approaches and their (possible) application in the field of activity recognition. Furthermore, it introduces a structured modelling process for developing human behaviour models for activity recognition based on the state of the art and the experiences made in Chapter 3. The chapter aims at providing a structured workflow for developing Computational Causal Behaviour Models for activity recognition. It should allow easier problem backtracking, better model documentation, and results reproducibility – aspects that are generally under-researched in the field of activity recognition.*

Chapter Sources: *This chapter contains previously unpublished work.*

Questions to be answered in the chapter:

What development processes are there in the different fields of computer science? (In Section 5.2)

What is the development process for CCBM proposed in this work? (In Section 5.3)

5.1 Introduction

The previous chapters showed that modelling human behaviour with CCBM is not a trivial task. It is rather a complex process where different model components interact and influence each other to provide good activity recognition results. It was also shown that there are modelling patterns that can reduce the model complexity and improve the actions' probabilities.

However, so far there is no structured process that provides step by step guidelines about developing CCBM models in a successful manner (or for any approaches that combines symbolic and probabilistic techniques). As already shown in Chapter 3, intuitive modelling can lead to a variety of problems some of which are visible only after the model is implemented and inference is performed. Such problems could potentially be avoided by applying a structured model development and by being aware of the effects a given modelling mechanism could have on the model. Yet, as discussed in the thesis introduction, usually the field of activity recognition concentrates more on the performance of the runtime model rather on the analysis and design of such models. For that reason, this chapter focuses on discussing model development processes from different fields of computer science and their suitability in the case of causal models for activity recognition. Based on them and the experiences made in Chapter 3, a development process for CCBM models is introduced. The chapter is structured as follows. Section 5.2 introduces existing development processes and discusses their (possible) application in the field of activity recognition, as well as the drawbacks they have for applying in this field. Later, Section 5.3 introduces the development process for Computational Causal Behaviour Models, where the process phases are discussed in detail.

5.2 Development processes for model engineering

Structured development processes are something common in the field of software engineering where it is a well established practice to divide the production and maintenance of a given software product in different stages. This allows better problem understanding, improves the quality of the final product and provides structured documentation. On the other hand, the field of activity recognition and data analysis concentrates on the implementation and performance of runtime models, usually omitting extended problem analysis and design phases. However, during the modelling of the three problems it became apparent that one could benefit from a more structured development process for human behaviour models for activity recognition. Thus, here we discuss the concept of a development process in different fields of computer science and try to find the bridge between data analysis and engineering.

5.2.1 Software engineering development processes

In the field of software engineering, there are different well established models of the software development process [135, p.7]. They all aim at producing more reliable, efficient, and easily reproducible software. However, it is possible that the developed product is actually a model itself and not a software. Schreuders argues that engineering paradigms should be considered not only in the field of software creation, but also in the cases when the product is a model for problem solving [127]. Thus, software engineering processes can also be applied to model development. For that reason, here we look into different software development processes.

5.2.1.1 The waterfall model [120]

The waterfall model was introduced by Royce in his work *Managing the development of large software systems: concepts and techniques* [120]. The process was later called *waterfall* because of the cascade structure from one phase to another. Fig. 5.1 shows the structure of the waterfall model. In it the first phase is the *requirements definition* where the system services, constraints and goals are identified, followed by the *system design* where the components

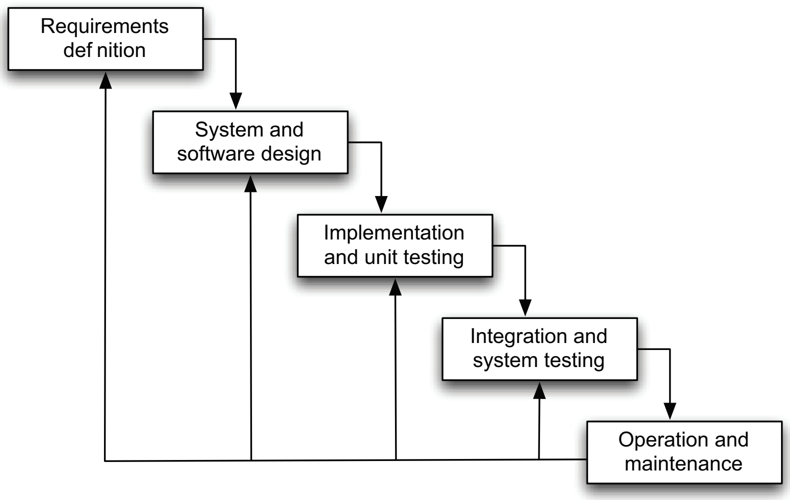


Figure 5.1: The waterfall software development process (Figure adapted from [135, p.9].)

of the system are conceptually developed. The third phase is the *implementation and testing*, where the designed components are realised in the form of program(s) or program components. The units are then verified for meeting their specifications during the unit testing. The fourth phase is the *integration and the system testing*, where the individual program components are integrated and tested as a complete system. The final phase is the *operation and maintenance*, where the system is put to practice, and where errors that were not discovered earlier are corrected in the course of the system usage.

It can be seen that the waterfall model is an iterative process where the different stages feed information to each other and where in practice they sometimes overlap. However, as Sommerville explains, an iterative process makes it difficult to identify checkpoints for planning and reporting, thus after several iterations some of the early phases are frozen, and the process continues with the later phases. This premature freezing could lead to the system not doing what the user expects it to do [135, p.10].

In general, the waterfall model can be used as a basis for CCBM development process, as it complements the intuitive development process identified in Chapter 3. However, in order to be applicable in the case of causal models, it has to be seriously adapted for the needs of CCBM and activity recognition. This is mainly based on the fact that the waterfall model is very general model that could be applied to variety of applications but the detailed procedure for the different fields has to be added. Even more, it is unable to cope with the development of the model's probabilistic structure.

5.2.1.2 Evolutionary development [135, p.11]

In difference with the waterfall model, evolutionary development is based on the idea of developing an initial implementation, exposing it to user comments, then refining it through different versions until the desired system has been developed [135, p.11]. The separate development phases in the waterfall model here are carried out concurrently with feedback exchange between the activities. Figure 5.2 shows the evolutionary development process where the specification, development and validation are done in parallel based on a simple outline description[1].

[1]Here by *simple description* is meant that the model specification is simpler than the more complex and detailed analysis and design phases in the waterfall model.

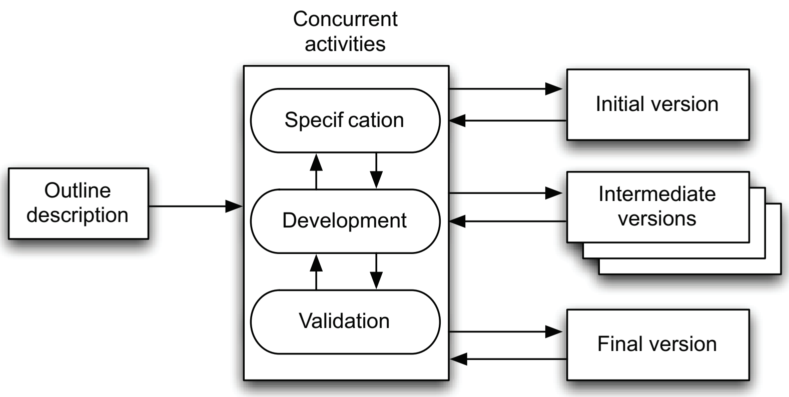


Figure 5.2: The evolutionary software development process (Figure adapted from [135, p.11].)

There are two types of evolutionary development. The first is *exploratory programming* which aims at working with the customer to identify the requirements and deliver a final system. The initial development is done on those parts of the system that are understood and continues to add new features proposed by the customer. The second type is *throw-away prototyping* which strives to understand the customer requirements and use them as basis for defining better system requirements. The prototype then concentrates on experimenting with the parts of the requirements that were poorly understood.

Sommerville argues that although the evolutionary development process is more successful than the waterfall model from a user point of view, there are three basic problems. The first is that *the process is not visible*, and there are no documents which reflect the different versions of the system. The second problem is that *the system is usually poorly structured* as the continuous change corrupts the software structure. This in turn makes the software evolution difficult and costly. The last problem is that *often special skills are required.* This is based on the fact that most available systems developed in such manner are implemented by small teams of highly skilled and motivated people [135, p. 12].

The problems concerning evolutionary development are actually the core problems that led to the need of a structured CCBM modelling process. The lack of documentation and rapid prototyping led to the inability to track the reasons behind modelling solutions, and the discovery of implementation or design bugs. On the other hand the CCBM modelling toolkit presented in Chapter 4 has parallels with throw-away prototyping, as among other things it examines the model requirements and strives to provide solution for vague or not well understood requirements.

5.2.1.3 Boehm's spiral model [15]

An alternative approach was proposed by Boehm [15]. He suggests that a model which explicitly recognises risk can be used as the basis for a generic software engineering process. Boehm's model has the form of a spiral, hence the name. Each loop in the spiral is a process phase. This means that the innermost loop could concern system feasibility, the next – system requirements, etc. Fig. 5.3 shows an example of the Boehm's spiral model. However, it is just an example and the general model itself does not have fixed phases. The decision over the phases' structure is left for the stakeholders.

Furthermore, each loop in the spiral is split into four sectors. The first is the *objective setting* where the objectives for the project phases are defined, the constraints on the process are identified and a detailed management plan is constructed. Additionally, possible project

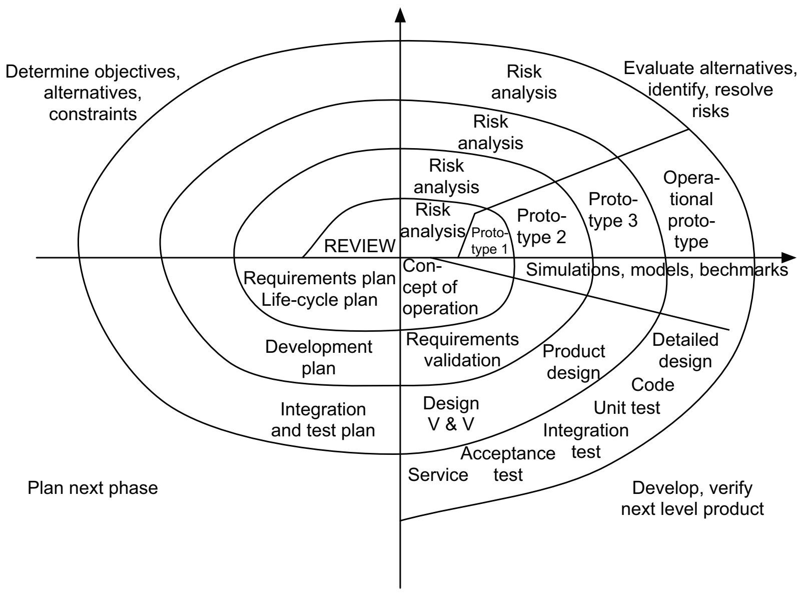


Figure 5.3: The Boehm's spiral software development process (Figure adapted from [135, p.13].)

risks are identified. The second sector is the *risk assessment and reduction*. In it for each of the identified risks a detailed analysis is carried out and precautions are taken to reduce the risks. The third sector is the *development and validation* where a development model for the system is chosen. Based on the dominating risks different development models could be chosen. The last sector concerns with the *planning* where the project is reviewed and a decision is made whether to continue with the next level of the spiral [135, p. 14].

The Boehm's spiral model in itself is relatively general and applicable to model based activity recognition. The problem could arise in the third sector of the model that deals with the development and validation as in this phase a concrete development process has to be chosen. Thus in order to use the spiral model, one should have a suitable development process that can satisfy the needs of model based activity recognition.

5.2.2 Modelling and simulation development process

An example of a model as the output of the development process comes from the field of modelling and simulation where model development processes are used for model creation and maintenance in a more structured and easily controllable and reproducible manner [11, 121].

5.2.2.1 Discrete-event system simulation process [12]

An example of a development process from the field of modelling and simulation is that for discrete-event systems simulation [12, p. 34]. Such simulations deal with systems where the state variable changes at a discrete set of points in time. Banks et al. [12] propose a simulation study lifecycle for such systems that is composed of 12 steps. The process can be seen in Fig. 5.4. The first step is the *problem formulation*, where the problem to be solved is stated and where possible conflicts in the problem statement are resolved. The second step is the *setting of the objectives and the overall project plan*, where the questions to be answered are defined, and the plans of the study is made in terms of participants involved, cost of the study, number of days required for accomplishing each phase of the project, and the results expected at the end of each phase. The third step is the *model conceptualisation* which deals with the construction

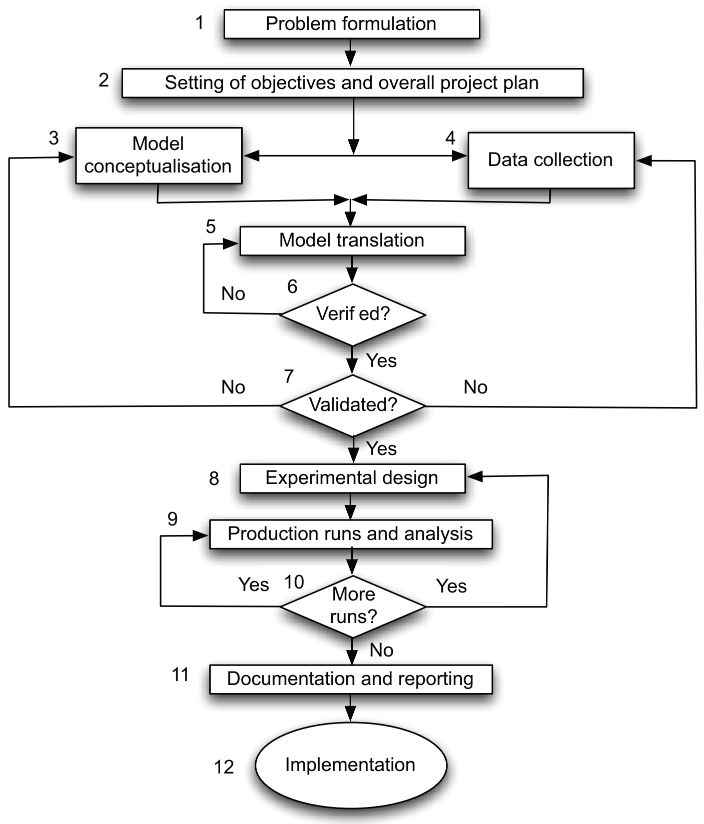


Figure 5.4: The discrete-event system simulation process (Figure adapted from [12, p. 35].)

of the model. Banks et al. [12, p. 36] advise the developer to start with a simple model and then build toward greater complexity. They also advise the participation of the model user in the model conceptualisation as it will enhance the quality of the resulting model and increase the confidence of the user.

The fourth step is the *data collection*, which deals with gathering the needed input data. As the complexity of the model changes, also the required data elements may change. The fifth step is the *model translation* where the conceptual model is implemented in a computer-recognisable format. The sixth step is the *model verification*. It involves checking whether the implemented model is performing properly, namely whether the input parameters and logical structures of the model are correctly represented. This step is followed by the *model validation* where the model is calibrated by iteratively comparing the model behaviour with the actual system behaviour. The model is then iteratively improved until a satisfiable accuracy is achieved. The eighth step is the *experimental design* where the alternatives to be simulated are defined. For each system design that is to be simulated, the length of the runs, the number of replications, and the initialisation period are decided. The ninth step is the *production runs and analysis* which estimate the measures of performance for the system designs that are being simulated. The tenth step decides whether *more runs* are needed and what design they should have. The eleventh step is the *documentation and reporting* which is divided into program documentation and progress documentation. While the first documents how the program operates, the latter provides information about the simulation project history. The last step is the *implementation* where based on the simulated solutions and their performance, the client makes the decision about the solution to be implemented.

This simulation study process has similarities to the intuitive activity recognition process, although it cannot be directly applied to activity recognition in its current form. The reason for that is that while the output of the simulation process is information about which problem solution would be most appropriate, in the case of activity recognition, the output is that of the implemented system, and the question then is not only what is the best way of predicting the user actions but also how well it is able to predict the user actions. Also in the case of activity recognition, we cannot change the available data so that it will represent the needs of the model, but rather the other way around – the model has to be able to represent and reason about the available data. Furthermore, the process is designed for a setup where the actual data is recorded after the simulation model is developed and evaluated. In the field of activity recognition it is often the case that the data is recorded beforehand and due to the cost of the experiment (in terms of participants, settings, infrastructure etc.) one is unable to repeat the experiment. It is also the case that the model is built in order to explain the data, and not vice versa[2]. Another aspect that simulation processes do not cover is that an activity recognition model has to cope with future data, so it cannot rely only on learning from the available data. For that specific reason prior knowledge is utilised in the activity recognition models.

5.2.2.2 A systematic methodology for developing discrete event simulation models [121]

An alternative methodology for developing discrete event simulation models was proposed by Rus et al. [121]. It has five separate phases and could be considered as an adaptation of the waterfall model. Fig. 5.5 shows the process phases and the steps of each subphase. The first phase involves the simulator *requirements identification and specification* where the

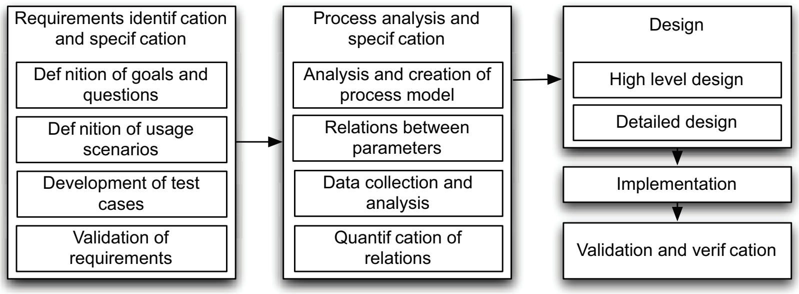

Requirements identif cation and specif cation
Def nition of goals and questions
Def nition of usage scenarios
Development of test cases
Validation of requirements
Process analysis and specif cation
Analysis and creation of process model
Relations between parameters
Data collection and analysis
Quantif cation of relations
Design
High level design
Detailed design
Implementation
Validation and verif cation

Figure 5.5: The discrete-event system simulation process proposed by Rus et al. [121].

purpose and the usage of the model are defined. Also the questions the model has to answer are determined as well as the data needed to answer these questions. The phase is divided into four steps: the first is the definition of goals, questions, and the necessary metrics for answering the questions; the second is the definition of the usage scenarios where the use cases are defined that are used for answering the problem questions. The third step is the development of the test cases which are used to verify and validate the model and the resulting simulation. The last step is the validation of the requirements, where the customer involved in the activity must agree with the content of the model specifications.

The second phase is the *process analysis and specification* which involves the understanding, specification and analysis of the process that is to be modelled. This phase is also divided into four steps. The first is the analysis and creation of a static process model which means that the problem is analysed, an abstract model is defined and later refined. The process model

[2]This brings an interesting discussion about combining simulation processes and activity recognition. One could produce simulated data on which the model is validated, and afterwards to conduct the recording of a real data for the model evaluation.

then shows which activities transform which artefacts and how information flows through the model. The second step is the creation of the influence diagram for describing the relations between parameters of the process. Here influence factors are such that change the result or behaviour of other project parameters. The third step is the collection and analysis of empirical data for deriving the quantitative relationships between process parameters. The last step in this phase is the quantification of the relations and the distinguishing of the different parameter types. The output of the second phase are models of the software development process and parameters that have to be simulated, as well as the description of the assumptions that were made during the phase.

In the third phase is the model *design* is developed that is independent of the model implementation. The design is divided into two parts: high level design and detailed design. In the high level design the surrounding infrastructure is defined, as well as the way in which the input and output data is managed and represented. In the detailed design the decisions about which activities to be modelled are made, as well as the items that are to be represented, and the attributes they should possess. Finally, the flow of the different items is defined.

The fourth phase is the model *implementation*, where all the design decisions and the necessary information are transferred into a simulation model. The last phase is the *validation and verification*, where it is proved whether the model is suitable for the problems it should address.

This development process, similarly to the previous simulation process, cannot be directly applied to an activity recognition problem. This is due to the fact that while simulation deals with the problem of determining what will be the most suitable solution to a problem, model-based activity recognition already provides a model that is a solution of the problem, and the question then is "how well the model performs". Additionally, the fact that the early phases are no longer revisited at a later stage, means that problems discovered during the implementation and validation phases cannot be fixed by returning to the early stages.

5.2.2.3 A life cycle for modelling and simulation [11]

Another modelling and simulation development process is the life cycle proposed by Balci [11]. In it the development process is viewed from four different perspectives – process, product, people, and project. The life cycle specifies the work *products* that are to be created under the corresponding *processes*, together with the integrated verification and validation activities. It also structures the development and provides guidelines for *project* management, and finally, identifies areas of expertise in which to employ qualified *people*. Fig. 5.6 shows the phases in the proposed model. Although the arrows are sequential, the process is not intended to be linear or sequential. It is rather an iterative process where reverse transitions are expected.

The first phase in the development process is the *problem formulation* where the problem at hand is systematically analysed executing the following steps: establish the problem domain boundary; gather data and information about the problem; identify the stakeholders and decision makers; specify the needs and objectives of the stakeholders and decision makers; identify and specify the constraints; and clearly specify all assumption made so far. The second phase is the *requirements engineering* which deals with identification and specification of requirements based on the formulated problem. Balci argues that a use case based requirements engineering is the best practice for requirements identification and specification, because a use case represents a small amount of the work the model is supposed to perform, thus it allows decomposing the complex problem into subproblems. The third phase is the *conceptual modelling* which deals with developing the highest layer of representation of conceptual constructs and knowledge. The conceptual model is created to assist in designing many different types of simulation

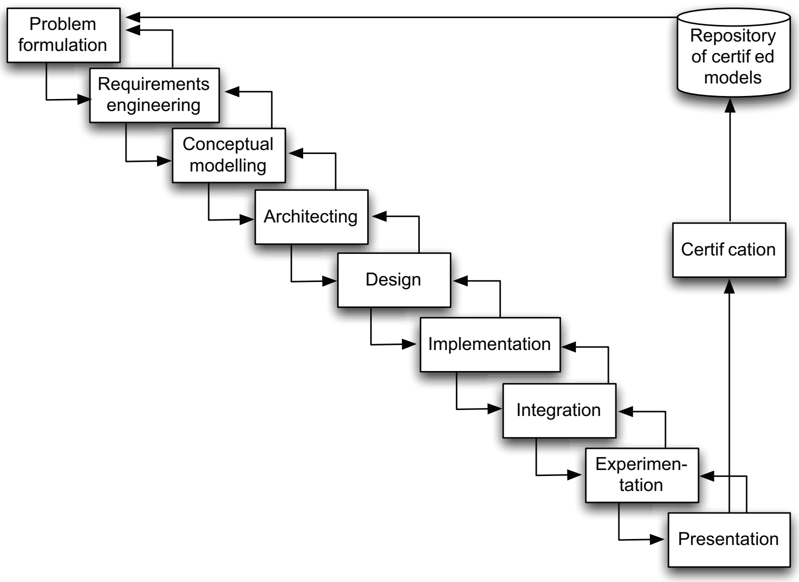


Figure 5.6: The life cycle for modelling and simulation proposed by Balci et al. [11] (Figure adapted from [11].).

models, thus creating model reusability on a conceptual level. The fourth phase is the *architecting*. It deals with specifying the model architecture based on the conceptual model. Balci explains that there is a distinct difference between design and architecture, as a design is an instantiation of an architecture. This is also the next phase in the development process where the model *design* is derived from the architecture specification. Balci proposes decomposition and modularisation as solutions to reducing and managing the design complexity. The sixth phase is the model *implementation*, which takes the model design from the previous phase and programs it using a simulation software product. To overcome the model complexity, it is decomposed into submodels. The seventh phase is the model *integration* which deals with combining the individually developed submodels into an integrated simulation model. The eight phase is the *experimentation, exercise or use*. During this phase the model is experimented with. This phase produces the simulation results based on which a solution for the problem could be chosen. The ninth phase is the model *presentation* which involves the interpretation and documentation of the simulation results, and their presentation to the decision makers. Aside from these nine phases there is the model *certification* that awards a certificate that the model satisfies a specific quality criteria. Balci's life cycle also regards the model *storage* together with all the documentation and data in a repository that allows future *reuse*.

Although more complex than the process proposed by Rus et al. [121] it basically has the same structure and limitation. To be applied to the field of activity recognition such model should be adjusted to the specific need of this field. The biggest drawback in it is that while it provides a basis for choosing the best problem solution, an activity recognition model is already a problem solution, so the choice of what and how to be modelled should be made already in the first phases of the process.

The three presented methodologies could all be fitted in a general model of a simulation experiment. Leye et al. discuss different modelling and simulations life cycles and propose a general structure for modelling and simulation experiments [88]. It contains six phases – specification where the experiment is defined, configuration where the model parameters are selected, simulation where the model is executed, data collection where simulated data of interest is collected, and evaluation where the model results are assessed. These phases resemble parts of the intuitive process from Chapter 3 – the model implementation and evaluation where the model is applied to the available data and the estimated behaviour is analysed and evaluated. However, these are just the later parts of the activity recognition model development. Before that the test data has to be collected, annotated, and the model itself has to be developed. Furthermore, development processes do not concern themselves with the problem of developing the probabilistic model structure that is responsible for providing adequate activity recognition.

5.2.3 Development processes for ontologies

One way of encoding prior knowledge for activity recognition is by using ontologies [104], and although there are no development processes for models for activity recognition, there are some for developing and maintaining ontologies [60]. An ontology development process resembles a software engineering process in that it is divided in similar phases – specification, conceptualisation, formalisation, implementation, and maintenance [48]. Below we discuss several different development processes for ontologies.

5.2.3.1 Uschold and King's ontology development method [149]

According to [60, p. 115] the first method for building ontologies was proposed by Uschold and King in [149] and later extended in [148]. The development process they proposed was based on experience gathered from developing the Enterprise Ontology [150]. Uschold et al.

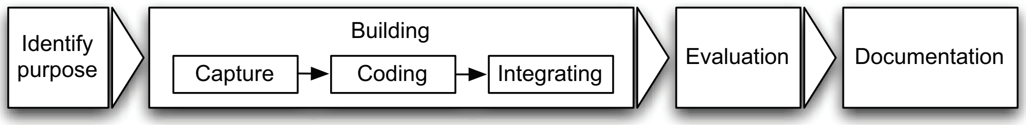


Figure 5.7: The ontologies development process proposed by Uschold and King [149] (Figure adapted from [60, p. 115].).

propose a process that consists of four phases. Fig. 5.7 shows the structure of the development process. The first phase consists of *identifying the ontology's purpose and scope*. It aims at defining why is the ontology being built, what applications it has, and what the relevant terms of the domain will be. The second phase is *building the ontology*. As can be seen in Fig. 5.7, it is divided into three activities: the first is the ontology capture which deals with identifying the key concepts and relationships in the problem domain. The second activity is the coding, where the basic terms that will be used for specifying the ontology are committed; and where the actual ontology is implemented. The third activity is the integration of existing ontologies and deals with the way in which already existing ontologies are used in the developed ontology.

The third phase in the proposed process is the ontology *evaluation* where a technical judgement is performed on the implemented ontology, its documentation and the associated software environment. The last phase is the ontology *documentation* where the guidelines for the documentation are established.

As Gomez-Perez et al. [60, p. 119] explain, the main drawback in this method is that it lacks a concept phase before implementing the ontology. This leads to problems in understanding the ontology when based on its implementation. This in turn causes the inability of experts to build such ontologies in their domain of expertise. This is also one of the main drawbacks of the intuitive model development presented in Chapter 3.

5.2.3.2 On-To-Knowledge [136]

The On-To-Knowledge project's [136] purpose was to apply ontologies to electronically available information in order to improve the quality of knowledge management in large and distributed organisations. Among other things the project includes a methodology for building ontologies. The proposed development process can be seen in Fig. 5.8. It consists of five

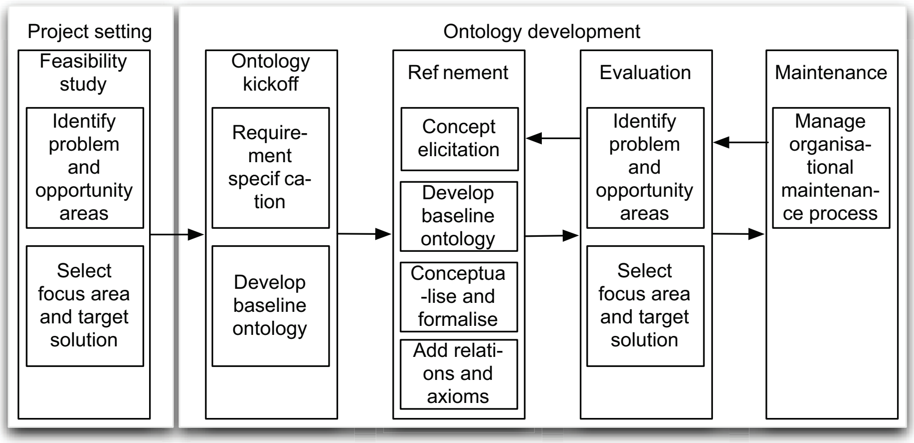

Project setting
Ontology development
Feasibility study
Identify problem and opportunity areas
Select focus area and target solution
Ontology kickoff
Require-ment specif ca-tion
Develop baseline ontology
Ref nement
Concept elicitation
Develop baseline ontology
Conceptua -lise and formalise
Add relati-ons and axioms
Evaluation
Identify problem and opportunity areas
Select focus area and target solution
Maintenance
Manage organisa-tional maintenan-ce process

Figure 5.8: The ontologies development process proposed by Staab et al. [136] (Figure adapted from [60, p. 147].).

phases, the first of which is the *feasibility study* where the problem is identified, as well as the most promising focus and the target solution. The second phase is the *ontology kickoff* where the ontology requirements are identified and specified together with the domain and the goal of the ontology, the design guidelines, the available knowledge sources, and the potential ontology users and use cases. Additionally, a baseline taxonomy is developed. The third phase is the ontology *refinement* where the goal is to create an application-oriented ontology based on the guidelines produced by the kickoff phase. This phase is divided into two activities: the first is the knowledge elicitation process with domain experts where the baseline taxonomy from phase two is further developed and refined. The second activity is the ontology formalisation where the ontology is implemented using the chosen ontology language. The language is selected based on the requirements of the application.

The fourth phase is the ontology *evaluation* and it deals with proving that the developed ontology and the corresponding software environment are serving their purpose. During this phase, two activities are executed: the first is checking the requirements and competency questions derived in the second phase; and the second deals with testing the ontology in the target application environment. The last phase is the *maintenance*. In the On-To-Knowledge project it was proposed that the ontology maintenance is integrated in the system software [60, p. 146–148].

In difference with the methodology proposed by Uschold et al. [149], this one has a more extensive conceptualisation phase. However, it still suffers from the fact that earlier phases are not included into the iteration process, thus making it impossible to correct conceptual problems discovered at the later phases. On the other hand, similarly to the intuitive development process in Chapter 3, it defines the conceptual elements to be modelled and the requirements based on which the implementation language is selected. Additionally, the evaluation process here is nearer to that of activity recognition than the validation and verification processes typical for simulation and software engineering.

5.2.3.3 Methontology [48]

Methontology is a structured methodology for building ontologies from scratch proposed by Fernandez et al. [48]. It is based on the experiences made in developing an ontology in the domain of chemicals [48].

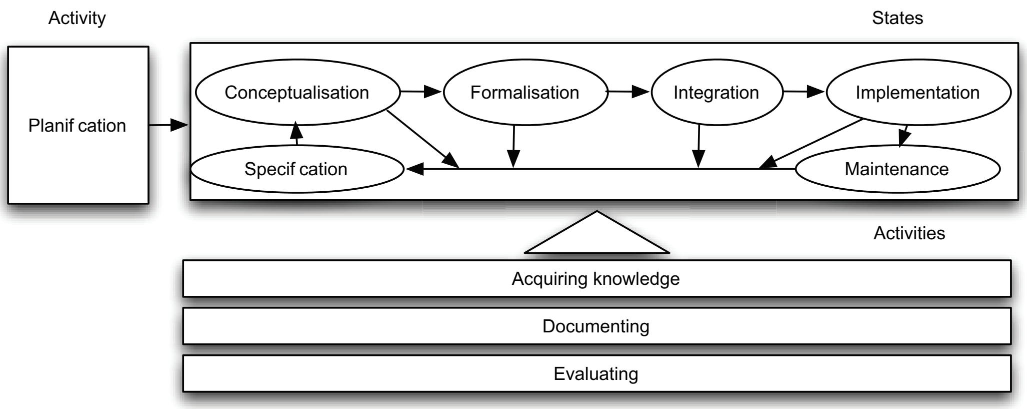


Figure 5.9: The ontologies development process proposed by Fernandez et al. [48] (Figure adapted from [48] and [60, p. 127].).

Fig. 5.9 shows the lifecycle for developing ontologies. The first step that has to be performed before building an ontology is the *planification* where the ontology designer plans the main tasks that have to be executed, how they should be arranged, what time and resources are needed for each task. After the planning, the ontology development begins. There are six states (or phases) through which the developer should pass in order to create successful ontology. The first phase is the ontology *specification* which deals with the questions *what is the purpose of the ontology*, *what are the ontology applications*, and *what are the users of this ontology*. The second phase is the ontology *conceptualisation* where the problem to be solved and its solution are described. The conceptualisation is based on the problem specification and the knowledge acquired about the problem domain. The third phase is the *formalisation* of the ontology. In it the conceptual ontology is transformed into a formal model, in this case using description logic or frame oriented representation systems. The fourth phase is the ontology *integration*. The purpose of this step is to reuse already existing ontologies by integrating them into the ontology being built. The next phase is the ontology *implementation*. In it the formalised ontology is implemented in the selected formal language. The final phase is the ontology *maintenance* which deals with modifying or including new definitions to the ontology.

In parallel to the phases above, three more activities are executed. These are the *knowledge acquisition* where the knowledge to be integrated into the ontology is elicited and where the sources of this knowledge as well as the techniques used for acquiring it are listed. The second action is the process of *documenting* where each step of the development process is carefully documented for future ontology use or reuse. The last action is the ontology *evaluation* where the technical soundness and the applicability to the problem domain are proved. These three actions are executed throughout the whole development process and are not considered as separate phases in the ontology development lifecycle.

In difference with the previous two methodologies, Methontology provides the option to iterate the process and to return to the early phases at any step of the development. This allows fixing conceptual problems that were not discovered during the early development stages. This also provides a better mechanism of adapting and evolving the ontology. Of course, the methodology cannot be directly applied to the field of activity recognition as it has different objectives but the process can be adapted for the purposes of model based activity recognition. Furthermore, it does not concerns itself with developing the probabilistic model structure.

5.2.4 Development processes for context aware systems

Human behaviour models for activity recognition aim at representing the available a priori knowledge needed for providing rich user related information during the inference phase. Thus, it is reasonable to assume that a model engineering process would improve this knowledge incorporation. Indulska et al. argue that one of the requirements for context modelling is the support for software engineering [70]. They also point out that the majority of approaches *are concerned with runtime context representation, querying, and reasoning, not on requirements analysis, design, or testing*. This can be seen in works such as [103, 41]. In this section we look at some methods for developing context aware systems.

5.2.4.1 Knowledge-based system development lifecycle [61, p. 304]

Gonzalez and Dankel propose a development lifecycle for knowledge-based systems, arguing that in the past knowledge engineering has been performed in an improvised manner [61, p. 307]. Additionally, they point out that adapting the popular waterfall model is not sufficient

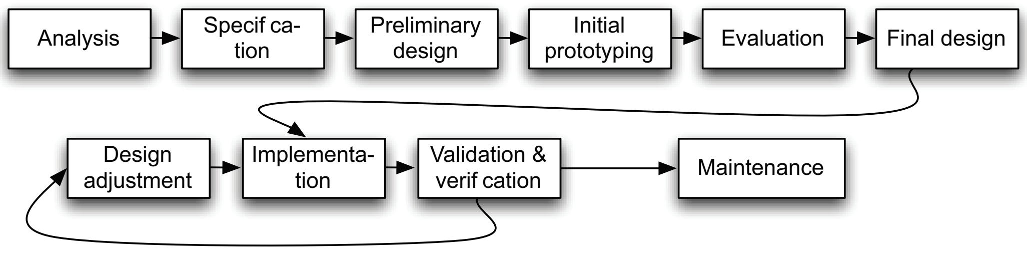


Figure 5.10: The knowledge-based systems development lifecycle proposed by Gonzalez and Dankel [61, p. 304] (Figure adapted from [61, p. 303].)

for knowledge engineering as it lacks rapid prototyping and incremental development. Thus they propose a model that combines rapid prototyping, incremental development and a cyclical lifecycle.

Fig 5.10 shows the lifecycle. It can be seen that it is divided into 10 phases with process iteration of subset of the phases. The first phase in the model is the *problem analysis* which regards the problem and the applicability of knowledge based solution to such problem. Additionally, the costs for developing such system are calculated in order to determine whether such system development is warranted. The second phase is the *requirements specification* where the knowledge obtained in the first phase is formalised. Based on that the objectives of the project are set as well as the means for obtaining them. The third phase is the *preliminary design* in which the high level decisions for the initial system implementation are defined. These are the knowledge representation paradigm, the tool chosen for prototyping, and the selection of system experts. The fourth phase is the *rapid prototyping* where the initial system prototype is designed that should look like the complete system but it should be limited in breadth. The prototype is then evaluated in the *evaluation* phase which deals with deciding whether the prototype could be further developed or discarded. The next phase is the *final design* which involves the selection of tools and resources needed for developing the system. Additionally, in this phase a high-level description of the system architecture is provided. The seventh phase is the actual system *implementation* where the complete knowledge regarding the system has to be implemented in a computer readable format. The eighth phase is the *validation and verification* where it is tested whether the system is able to solve the problem it was designed for and whether it is consistent with the requirements and objectives defined earlier. The ninth phase is the *design adjustment* where some changes in the design of the system could be made at the beginning of each iteration. The last phase in the process is the system maintenance and as in conventional software engineering it deals with documenting, storing and adapting the system.

A model-based activity recognition system can be considered as a knowledge-based system, thus the methodology proposed by Gonzalez and Denkel could generally be applied to such kind of problems. However, although the process allows iterations of the later phases, it does not have the ability to return to early phases where conceptual problems could be made. Furthermore, the process does not allow the iteration between some of the phases. This indicates that problems in the early phases, discovered later on, cannot be corrected. Finally, here once again comes the problem of developing the probabilistic model components.

5.2.4.2 A model driven development method for developing context-aware pervasive systems [129]

Serral et al. [129] propose a development method for context aware systems that consists of four phases. The motivation behind the method is to develop context-aware pervasive systems by developing a set of models that are automatically mapped into system code.

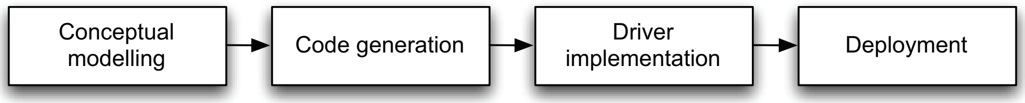


Figure 5.11: The development method for context-aware systems proposed by Serral et al. [129].

The development process can be seen in Fig. 5.11. It is rather simple and straight forward. The first phase is the *conceptual modelling* where the system is specified at a high level of abstraction in the form of PervML models [129]. These models are later used in the second phase for *code generation*. In it the PervML models are mapped into Java code and OWL specifications. The Java code represents the functionalities of the system whereas the OWL specifications the PervML ontology. The third phase is the *driver implementation* where the drivers needed for managing the access from the implementation framework to the devices are implemented. The final step is the system *deployment* where the Java implementation is configured to use the selected drivers.

The proposed method, although labelled as a software engineering method, is rather an example of the statement by Indulska et al. [70] that most context-aware systems deal with the runtime model implementation and configuration rather than with problem analysis and conceptual models. Furthermore, the method is linear and does not allow returning to earlier phases which could be a disadvantage when conceptual problems are discovered in the later phases.

5.2.4.3 A context-driven development methodology for context-aware systems [25]

A more complex method for developing context-aware systems is proposed by Choi et al. [25]. As they explain, context-aware systems demand custom development methodology because they have specific features such as the need of context modelling and the implementation of context-dependent services. The process they propose consists of three phases each of which

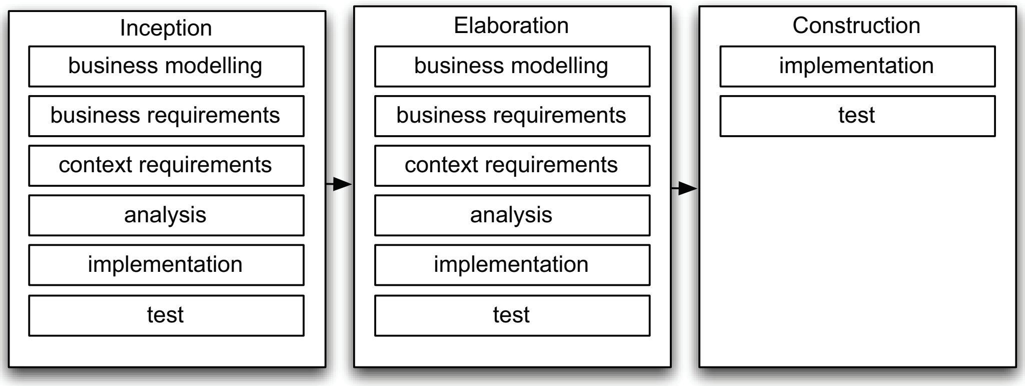


Figure 5.12: The development method for context-aware systems proposed by Choi et al. [25] (Figure adapted from [25].).

has various workflows. The first phase is the *inception* where stakeholders define the scope of the project, the associated risks and costs. Additionally, the relevant context is identified. The phase consists of six workflows. The first is the *business modelling* where the business rules are defined and the business information is gathered. The second workflow is the *business requirements* where the stakeholders requirements are collected, the use cases are gathered, and a specification of both is documented. The next workflow is the *context requirements* where in difference with the business requirements, the goal is to gather context information from previously identified context-sensitive use cases. The fourth workflow in this phase is the *analysis*. In it the draft solutions of the use cases are defined. The two main tasks here are to determine the system platform and to create context-aware use cases. The fifth workflow is the *implementation* where a prototype is realised in order to prove the developed concept. The last workflow is the *testing*. It involves establishing test cases for acceptance tests.

The second phase is the *elaboration phase* where the stakeholders analyse the problem, define the system structure architecture and implement the core architecture. This phase has the same workflows as the inception phase but in a different context. During the *business modelling* the business processes and rules are refined and the business modelling process is completed. During the *business requirements* the requirements are also refined and specified in detail. During the *context requirements* workflow the gathered context information is categorised in similar groups in preparation for the context modelling. Additionally, context requirements are elicited. The next workflow is the *context modelling* where the context model is designed to meet functional and nonfunctional requirements. During the *analysis* workflow the system architecture is determined such that it meets the functional, nonfunctional and context requirements. Additionally, preliminary system design is performed. This is followed by the system *design* where the preliminary design is refined and the realisation of the context representation and storage is defined. Later the preliminary architecture framework is implemented during the *implementation* workflow.

The last phase is the *construction phase* where the different system modules are implemented, tested and integrated into the complete system. This phase consists of only two workflows. The first is the *implementation* where the preliminary system implementation is completed and subcomponents and detailed services are added. The second is the *test* workflow, where the different modules are tested to check whether they operate correctly for a given context. Additionally, the system as a whole is tested whether it operates correctly for the given context.

The proposed methodology is much more detailed than the one proposed by Serral et al. [129]. It also allows refinement of the workflows and has a detailed analysis and conceptualising steps. Still, it is possible that the system needs more than just three iterations to be successfully completed and that could mean iterating the whole process, which is not part of the proposed process. Here the same problem as in the previous two processes exists: in the case there is a problem in the early stages, the process does not allow the ability to return to a previous phase. Furthermore, not surprisingly, it is not able to cope with the development of the model's probabilistic elements.

5.2.5 Data analysis processes

So far we looked at different methodologies from the fields of software engineering, modelling and simulation, knowledge and ontologies engineering. The common between them all is that they all deal with the engineering view of creating a system. They all have a set of questions that have to be answered, a set of requirements that have to be satisfied, and in return output a system that is the solution to those questions and requirements, and that is later tested for satisfiability, further developed and maintained.

On the other hand, the data analysis view is a bit different. In it empirical data is gathered and analysed in order to answer different questions. As Cohen explains, no matter whether one is conducting experiments with rats or with programs, there are always three basic research questions [26, p. 3].

How will a change in the agent's structure affect its behaviour given a task and an environment?

How will a change in an agent's task affect its behaviour in a particular environment?

How will a change in an agent's environment affect its behaviour on a particular task?

To answer any of these questions based on empirical data, one can apply one of four types of empirical studies [26, p. 7]. The first are the *exploratory studies* that deal with causal hypotheses that are tested in observation or manipulation experiments. Basically, that means that huge amounts of data are collected and later analysed for similarities. The second type of empirical studies are the *assessment studies*. They establish baselines and ranges of the behaviour of a given system, or the corresponding environment. The third type are *manipulation experiments* which test hypotheses about causal relations and factors by manipulating them, and determining effects on measured variables. The last type are the *observation experiments*. They disclose effects of factors on measured variables by observing associations between levels of factors and values of the variables.

As Cohen explains, in the early stages of a project one asks questions that are answered by exploratory studies, while as the project progresses the answers are obtained by experimental studies. This shift from exploratory to experimental studies defines the progress in science. This is also shown in Fig. 5.13 where with the progression form a specific system to more general, also our understanding shifts from descriptive to causal explanation.

To answer research questions based on empirical studies, Cohen proposes a strategy. This can also be regarded as a process for data analysis as data analysis is based on exploratory studies. This process consists of five steps [26, p. 6] and describes a typical empirical generalisation strategy that is usually used in any data analysis study.

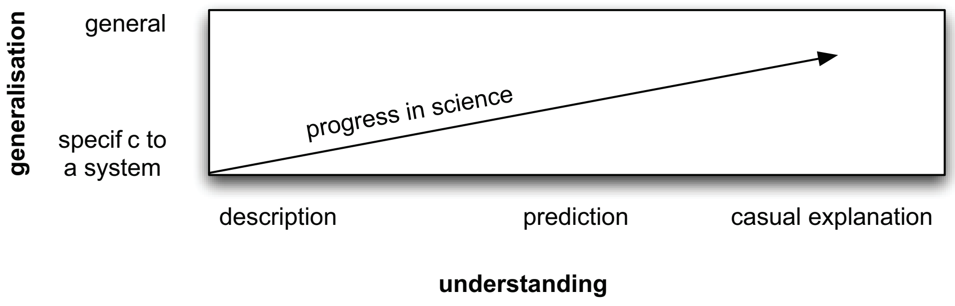


Figure 5.13: The generalisation and understanding of basic research questions [26, p. 3] (Figure adapted from [26, p. 3].).

1. Implement a program that exhibits a behaviour of interest performed in a specific environment.
2. Identify the program's features, tasks and environments that influence the target behaviour.
3. Develop and test a causal model of how the selected features influence the behaviour.
4. When the model is able to provide accurate predictions, generalise the features so that other variables, programs and features are included in the model.
5. Test whether the general model is able to accurately predict the behaviour of the larger set of programs, tasks, and environments.

This general approach for answering questions in empirical studies is also reflected in the state of the art for data analysis processes. Below we look in two such processes.

5.2.5.1 Pattern classification lifecycle [36, p. 14]

A more concrete study design lifecycle is proposed by Duda et al. [36, p. 14]. It concerns the exploratory study design of pattern recognition. As pattern recognition analyses data for repeating similarities, the lifecycle includes the data collection, the choice of features to be compared, the choice of model, the training of the model and its evaluation. Fig. 5.14 shows

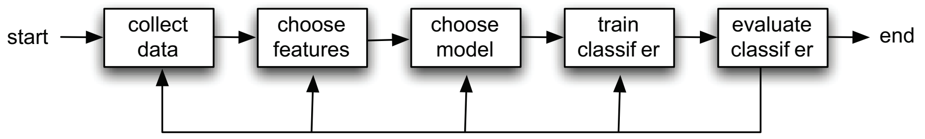


Figure 5.14: The pattern classification lifecycle proposed by [36, p. 14] (Figure adapted from [36, p. 14].).

the proposed process. The first phase is the *data collection* and deals with the data needed for training and testing the designated system. As a rule, the more the training data, the better the system performance [36, p. 14]. The second phase is the *feature choice*. The features to be compared are usually selected based on preliminary data analysis and on the available prior knowledge about typical features relevant for the problem domain. The third phase is the *model choice* where the model that is to be the solution of the problem is selected. Here, different models can be selected and later tested to find out which of them is able to represent the underlying patterns. The next phase is the *model training*. This phase deals with the process of using data to determine the classifier. Based on the patterns learned by the classifier it can later recognise and classify new data. The last phase is the *evaluation*, or with other words, how well is the classifier able to identify patterns in new data. Typical strategies and tests for performance evaluation can be found in [26, p. 185–235].

This is a typical data analysis approach that can also be recognised in the intuitive development process from Chapter 3. However, in our case the development process concentrates on the manner in which a model is developed. Furthermore, as CCBM proposes substitution of training data with a-priori knowledge, the fourth model phase is redundant.

5.2.5.2 Cross-Industry Standard Process for Data Mining [131, 21]

Another development method for data analysis comes from the field of data mining. The method proposed by Shearer is called **CR**oss-**I**ndustry **S**tandard **P**rocess for **D**ata **M**ining (CRISP-DM) [131] and consists of six recursive phases. The process can be seen in Fig. 5.15.

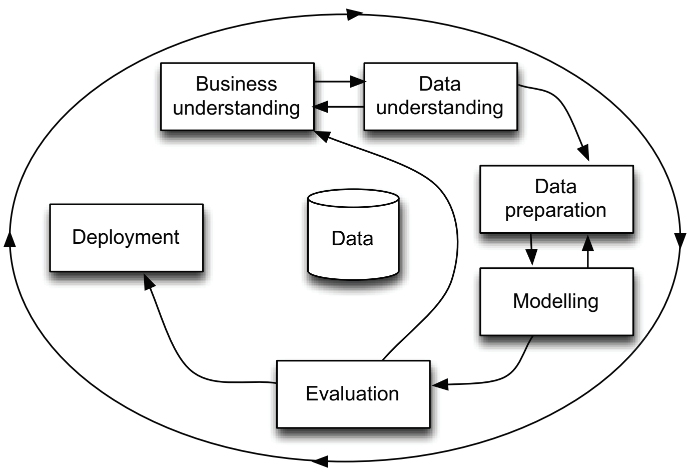


Figure 5.15: Cross-Industry Standard Process for Data Mining proposed by Shearer [131] (Figure adapted from [131].).

The first phase is the *business understanding* which tries to understand the problem from a business perspective and to convert the gathered knowledge to a data mining definition. It also deals with developing the preliminary plan for achieving the problem objectives. The second phase is the *data understanding* where the initial data is collected and analysed in order to discover data quality problems, initial insights into the data, or to identify interesting patterns that can help form hypotheses about the hidden information. The third phase involves the *data preparation* where the final dataset or the data to be fed into the modelling tool is prepared. It involves five steps: first the data to be used is selected based on the problem objectives, then the data is cleaned, or missing information in ambiguous subsets is estimated and added to the datasets. The third step is the data construction which involved preparing the data by deriving new attributes from existing ones or developing new records. Later the data is integrated by combining information from multiple tables or records to create new records or values. Finally, the data is formatted to fit the needs of the designated tool.

The fourth phase is the *modelling* where various modelling techniques are selected and applied to the problem so that their parameters can be calibrated and optimised. This involves the selection of the modelling technique, the generation of the test design, the creation of models, and finally, their assessment. According to Shearer the model assessment is based on the analyst domain knowledge, the data mining success criteria, and the desired test design.

The fifth phase is the model *evaluation* which deals with more detailed evaluation of the model performance and of its ability to satisfy the business objectives. It involves not only the evaluation of the model accuracy and generality, but also the representation of the business objectives, as well as any reasons why the model could be deficient. Finally, it is decided whether the model has to be finished and deployed or to initiate new iteration of the development process, or to set up a completely new data mining project.

The last phase in the project is the *deployment* where the knowledge gained throughout the project is organised and presented in a way understandable for the customer. This involves the development of a strategy for deployment, as well as one for the model monitoring and maintenance. Additionally, a final project report is produced and the whole project is assessed in terms of successes and failures.

The CRISP-DM process provides data analysis approach that generally meets the needs of CCBM and allows the process iteration and improvement. It also takes care of something that is not an issue in other fields of computer science but is important part of the activity recognition process: it provides a mechanism for collecting, preparing and using the data to be evaluated. On the other hand, as most data analysis approaches, it does not go into detail about the concrete modelling process. And from our experience with CCBM, exactly this is an essential part of a successful activity recognition process.

It can be seen that in difference with the engineering methods, the data analysis methods deal with how to collect, analyse and evaluate data, rather than with what is the best approach to building a model. They also do not concern themselves with extensive conceptualisation of the model to be developed. The experience from Chapter 3 showed that this is a major drawback as many conceptual problems appear first in the model evaluation, when they could be avoided all together in the presence of more extensive model analysis and design.

5.2.6 The gap between data analysis and engineering

From the above sections it became apparent that depending on the process application, each process puts more stress on the phases of interest for the concrete model developers. For example, in software engineering the processes deal with more detailed software conceptualisation and design[3] and output a validated and deployed software system. The software engineering processes are relatively general and can be adapted for different problems. Variations of the waterfall model can be seen in different fields of computer science like the modelling and simulation lifecycle proposed by Balci et al. [11], the ontology development process Methontology [48], or the knowledge-based development process proposed by Gonzalez and Dankel [61, p. 304]. The main disadvantage of these methods is that at some point the early analysis and design phases are no longer revisited which could account to conceptual problems that have to be solved during the implementation phase. This issue has been addressed by Parnas et al. in their work *A Rational Design Process: How and Why to Fake It* [107]. In it they discuss the problem of freezing early phases and point out that in reality if an error in the early phases is detected, one will always go back and fix it. They also argue that one usually does not follow the process exactly as described but rather later on produces documentation that pretends the process was followed.

[3]With the exception of the evolutionary software development process that produces a prototype as soon as possible.

To solve that problem, recursive development processes are proposed such as the Boehm's spiral model [15] or the evolutionary development process [135, p.11]. However, in the case of the Boehm's spiral model, the development process itself is still a choice of the developer, with the only requirement to be repeated until the desired product is outputted. And in the case with the evolutionary development process, the rapid prototyping does not allow thorough model conceptualisation which could lead to unexpected problems during the model implementation and evaluation.

Furthermore, the various processes described in the above sections deliver final products which differ from the output of a model based activity recognition system. For example, the output of a software engineering process is a software system that is validated and verified where by validation and verification the following is meant.

validation: *The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements* [IEEE-STD-610.12] [72, p. 212].

verification: *The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase* [IEEE-STD-610.12] [72, p. 213].

The processes for modelling and simulation, context-aware systems, and ontologies all have similar understanding for the concepts of validation and verification. The final output of this phase will be to prove that the system satisfies the previously defined requirements and that it is a solution to the problem at hand. However, the things look a bit different in the field of data analysis. Here in addition to the model correctness and suitability for the given problem, the developer should also concern herself with a more detailed evaluation of the model performance. This is due to the fact that there could exist a model that satisfies all requirements, and that is proved to be a solution of the problem, but that given real data, performs poorly[4].

On the other hand, the data analysis processes concentrate on the underlying data and the information the model can unearth from it rather on a detailed model development process. Chapter 3 has shown that this could be a drawback for the later model implementation and evaluation as some problems could have been easily avoided by more thorough conceptualisation.

Furthermore, most of the processes described above, produce the project documentation when the project is already completed. Still, the practice with the three experiments in Chapter 3 showed that when the documentation is not produced at the moment a decision is made, it is almost impossible to later reconstruct the given decision. This indicates that a better process for documenting the model development is needed.

Finally, one essential issue with the discussed development processes is the lack of mechanism for developing the model's probabilistic structure. A purely rule-based approach would easily fit into e.g. some adaptation of the waterfall model as the development of elements of the causal structure does not change the structure of the remaining elements. For example, we can sequentially develop all actions in the causal model and each newly implemented action will not change the causal structure of the remaining actions. On the other hand, when dealing with probabilities, the change of one probabilistic element will change all the remaining elements. For example, introducing weight to an action will not change just the one action, but will also have effect on all the remaining actions as each action is weighted with respect to the rest of

[4]Actually, the experience with modelling with CCBM so far has proven that first versions of the models that were successfully validated and verified, usually perform poorly provided sensor data.

the available actions. The same applied to introducing probabilistic action durations. An action duration will affect the execution time of all the remaining actions. This probabilistic aspect of approaches that combine logic with Bayesian inference is something that is not regarded in any of the existing development processes. This is also an essential aspect of the CCBM model development that has to be carefully considered.

It is obvious that there is a gap between the engineering view of model development and that of a data analyst. To bridge this gap, in the next sections a development process for engineering Computational Causal Behaviour Models for activity recognition is proposed.

5.3 Development Process for Computational Causal Behaviour Models for Activity Recognition

The human behaviour modelling process we propose is based on the process analysis performed in the previous section and on the intuitive development process identified in Chapter 3. It consists of five phases which include not only the model development but also its evaluation and documentation. The goal of the process is to systematically develop a working model with high activity recognition performance, and meanwhile to increase the ease of tracking and identifying modelling problems and of finding alternative solutions. It also aims at providing better model documentation, as in the field of activity recognition it is often the case that a detailed model documentation is missing [82].

Fig. 5.16 shows the process. It can be seen that the model has two layers. The first one resembles a standard waterfall model. The first phase in this layer is the analysis phase which consists of understanding and analysing the problem domain, identifying the model objectives, and later deriving the model requirements. It also takes care of collecting the data to be analysed, of deriving actions ontology, and later – of creating the data annotation based on the actions ontology. The second phase is the model design where modelling solutions for the objectives are selected. The third phase is the model implementation which is done with Computational Causal Behaviour Models and where in some cases the CCBM modelling toolkit could be used as it contains a library of solution templates. The fourth phase is the model validation in which the implemented model is validated against an existing plan or a set of plans and improved based on the annotation. As the developed model aims at recognising user activities, the modelling process consists of one more phase, namely the model evaluation. In it the model is employed for recognising activities of daily living and its performance is computed by comparing the recognised activities to the ground truth. Additionally, the modelling objectives derived in the first phase are used for success criteria in order to evaluate the model.

The second layer in the development process is introduced due to the need of coping with the model's structure that is a combination of causal and probabilistic elements. It consists of three phases. The first is the development of the causal model and the corresponding observation model. The second is the development of the action selection heuristics; and the third is the development of the actions durations. Each of the three phases spreads over three phases from the first layer: namely, design, implementation, and validation. During the development of the causal model the appropriate design solutions are chosen, both in terms of causal relations and constraints; and in terms of relations to the observations. Also the parameters to be inferred are decided upon. These are the parameters and sensor relations that will later be implemented in the observation model. After the design solutions are made, a minimal model is implemented, namely the simplest possible model that satisfies the problem. The model is then compiled and validated against existing plans, and against a collapsed action annotation

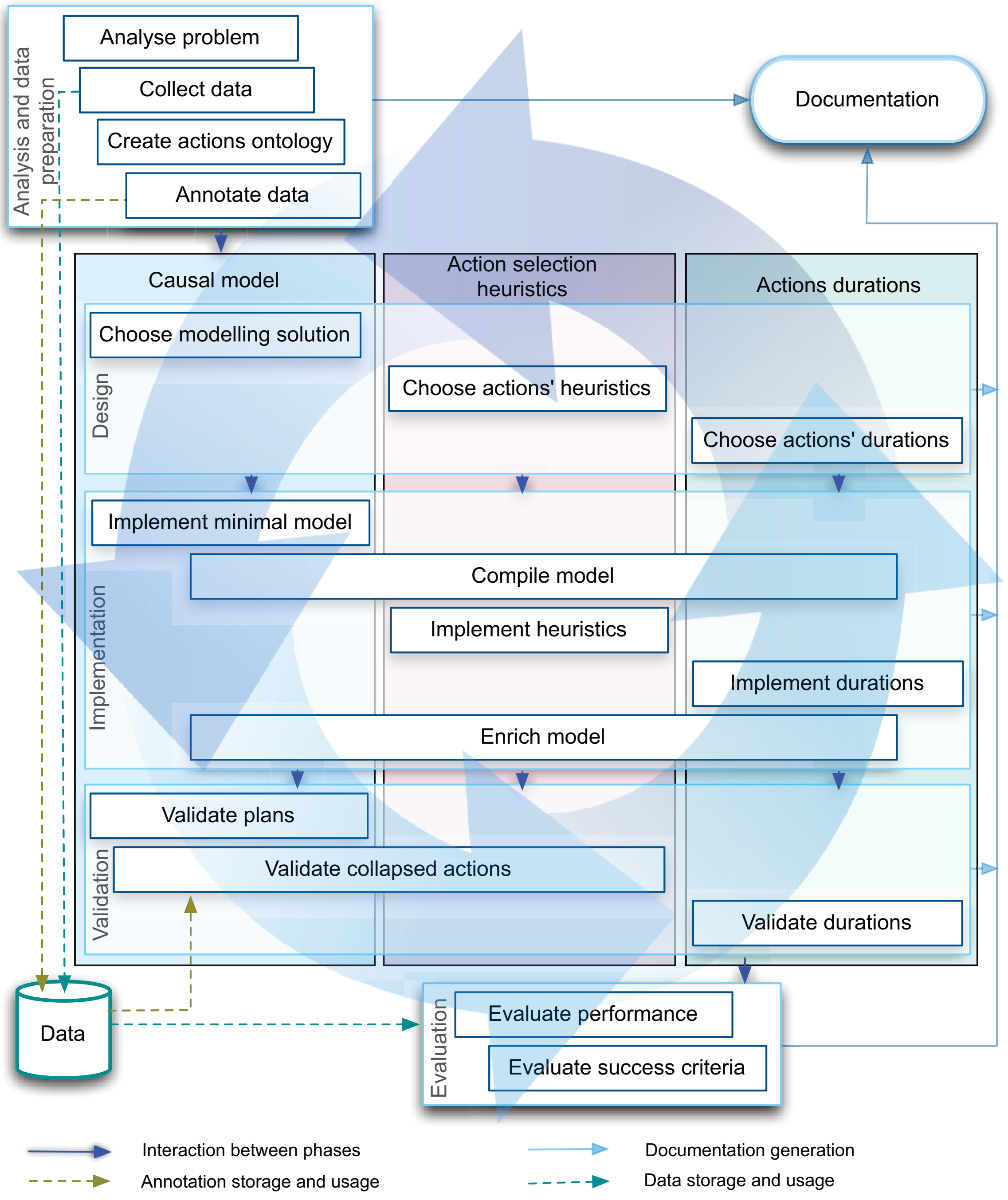


Figure 5.16: Process for developing human behaviour models for activity recognition.

(in other words, annotation without durations). In the case the model is successfully validated, it can then be enriched. During the second phase, the action selection heuristics are decided upon. First the heuristics to be used are selected together with the respective values. Later, the heuristics are incorporated into the already validated model. Finally, the model is once again compiled and validated against the plans and the collapsed annotation. During the last phase, the corresponding actions durations are decided upon, then incorporated into the model. The validation process is then repeated. In this case however, the durative annotation is used. The execution order of these three phases is necessary because the causal structure has to be completed in order to be able to introduce action selection heuristics. That is due to the fact that the selected heuristics will have influence on the whole model, not just on a local element they were assigned to. For that reason one has to first ensure that the model is causally correct, before introducing the probabilistic structure. Similarly, the inappropriate usage of action selection heuristics could render the correct action non-applicable by assigning too low probability to it. For that reason the correctness of the heuristics has to be validated before being able to proceed with introducing the probabilistic action durations. The opposite – first introducing the durations then the heuristics – could cause the problem that the correct action is never selected, thus the duration of this action cannot be validated. This could also lead to the designer's inability to determine whether the problem is due to the heuristics or to the durations.

Additionally, the process includes the storage of the data that is collected, annotated and later analysed throughout the developments lifecycle.

Furthermore, the process takes care that each phase is carefully documented so that the model development and evaluation as well as the decisions made can be easily backtracked.

Finally, the process is iterative one and it is repeated until the desired model effect is achieved: namely working model with high recognition rate. Furthermore, each of the phases can be iterated, and one can return to a previous phase without the need of starting the process from the beginning. Additionally, there are some more complex process dynamics, especially in the second model layer. They are discussed in detail in the following sections.

Although complex, Fig. 5.16 is just the surface of the process. To understand the different phases, their relationships and information flow, this section provides a detailed description of all model components and the corresponding process dynamics.

5.3.1 Analysis and data preparation

On one hand the analysis phase aims at understanding the application problem at hand and determining the model objectives. It also identifies what kind of human behaviour the model should be able to represent and finally derives the requirements the model should satisfy. On the other hand, it deals with the collection of the data that is to be analysed and its annotation. For that reason it has the following four objectives: (1) to analyse the problem domain; (2) to collect the data on which the model is to be tested; (3) to derive ontology of relevant activities to be annotated, theirs attributes and relation to the remaining activities; (4) to annotate the data based on the developed ontology. Later this will be the ground truth for the future model evaluation.

Below we discuss these questions and the practices through which to find their answers.

5.3.1.1 Domain analysis

The first step in the analysis phase deals with the problem to be solved, the elements involved in solving it, the objectives the future model should have, the requirements it should satisfy and the underlying sensor infrastructure. For that reason it is divided into three parts: the *problem analysis* where the problem at hand is analysed, the actions that need to be modelled are defined and the sensor infrastructure is decided upon. The second part is the *objectives analysis* where the model objectives are derived; and the third part is the *requirements analysis* where the requirements the model needs to satisfy are identified.

Problem analysis: The problem analysis deals with the answer of the following questions.

What is the problem to be modelled?: The answer to this question should provide information about what kind of behaviour is to be modelled; how many users are involved; what are their roles in the problem domain; what is the environment with which the user(s) is(are) interacting; which elements of the environment are to be modelled and which to be omitted.

What is the application for which the model will be used?: The general answer of this question is that the model is, of course, designed for activity recognition. However, there are different applications of activity recognition, e.g. it could provide information to an additional assistive component that proactively assists the user; it could monitor people who may exhibit erroneous behaviour caused by illness or age etc. Based on that, the various context information could be modelled.

What are the actions that will be modelled?: Based on the problem description, one should compile a list of actions that the user is able to execute within the given problem domain. These actions will be the building blocks of the corresponding user behaviour.

What kind of sensors do we need to capture these actions?: Based on what is to be modelled and what we want to recognise, an appropriate sensor infrastructure should be selected. As we are dealing with real data, it is important to analyse what kind of sensors could be used for capturing the behaviour to be recognised. Additional factors are the costs for such infrastructure, and the actual model need (after all it does not make sense to use complex sensors that capture every body movement when one needs to know only the relative user location).

All these questions basically represent the problem analysis produced in Chapter 2.3 (page 19). In it the three problems were identified, the elements to be modelled were defined, and the corresponding experiments and sensor infrastructure were selected.

Objectives analysis and hypotheses posing: Based on the use cases and their application domains, the model objectives describe what the designer aims to achieve by building a given model. They are derived from the problem domain and are based on the requirements[5] the model stakeholders have. The concept of an objective complies with that from the field of strategic planning and decision making where objective is defined as *some pre-established goal/s; these goals can apply to many different things, as for instance the manufacturing of a product with costs as low as possible (...)* [100, p. 3]. Each objective has a maximisation function and a threshold value. For example, an objective could be that the model should produce high recognition rate. The maximisation function then will choose the model with such parameters that provide the best performance, while the threshold will be that the model should perform at least as good as a handcrafted model. As an example from the meeting model, we can say that one of our objectives was to show that the team model performs as well as hand crafted model. We built the model, then evaluated the estimated activities and obtained results that were slightly better than a hand crafted HMM used on the same data. This means that our objective was satisfied. It is also possible that we build several different models, evaluate them and then choose the one with the highest performance to compare with the handcrafted model (see Section 3.3.1.3 on page 64). Based on that here objective is defined in the following manner.

Definition 28. ***(Objective):*** *A tuple $O := (P,T,N)$, where P is the problem the objective concerns, T is the objective's threshold value which defines whether an objective is successfully satisfied, and N is the objective's maximisation function.*

Definition 29. ***(Maximising an objective):*** *Given an objective $O := (P,T,N)$ and a set of models M that are solutions to the problem P, $M := \{m_1, m_2, ..., m_n\}$, with $f_O(m_i)$ being the*

[5]Here a stakeholder's requirement differ from the designer's requirements defined in Section 2.4 (page 26).

value associated with the model's performance concerning this objective. Then an objective's maximisation function is given by formula 5.1.

$$\arg\max_{m_{max}} f_O(m_{max}) := \{m_{max} | \forall m_n : f_O(m_n) \leq f_O(m_{max})\} \tag{5.1}$$

The derived objectives are later used to evaluate whether a given model was successful in solving the problem at hand, and in the case of multiple models – which of them showed the best results.

Along with the objectives definition, one also poses the hypotheses regarding the model that are to be statistically tested during the model evaluation. The hypotheses could also concern the data collected for evaluating the model. A hypothesis will be, for example, that a certain modelling mechanism significantly changes the model performance. Or that the collected data should have a certain underlying distribution.

The hypotheses could also be defined later during the model development but always before conducting the tests. The important thing here is that one has to define her hypotheses before analysing the data or the model results. The opposite – first looking into the data and the results and then posing hypotheses – could lead to incorrectly interpreting the results. For example, it is possible that a sensor is malfunctioning producing incorrect readings. If one first poses a hypothesis about what the data should look like and then sees the deviation from the expected value, it would be much easier to find out there is a malfunctioning sensor. On the other hand, personal experience showed that if one first looks in the data and then poses a hypothesis, it is likely that she will find a reasonable explanation of the readings instead of finding the problem in the sensor.

Requirements analysis: In Chapter 2 we used requirements analysis in order to select an appropriate modelling formalism. Beyond that, the requirements are later used in the model design for identifying additional model relations beside those identified during the annotation process. They also provide information about how to handle contradicting requirements based on their priority. The requirements analysis comes from the field of software engineering and complies with the definition given in Section 2.4 (page 26). The requirements analysis is divided into four steps. These are the *requirements collection*, *requirements classification*, *prioritisation*, and *requirements validation*.

Requirements collection: The first step toward defining the modelling requirements is their identification. That is done based on the problem analysis and the functionality the model has to possess. The requirements reflect the needs of the model application as well as the user behaviour to be recognised.

Requirements classification: After collecting the requirements, the next step is classifying them in different functionality groups. For example, the requirements for the three modelling use cases discussed in this work, were divided into behaviour-based and application-based, and each of these groups was further divided into two subgroups (see Section 2.4).

Prioritisation: Each requirement could be assigned a different priority with respect to the rest of the requirements. Such prioritisation is important when evaluating candidate formalisms that could satisfy different subsets of the requirements. In such case, the formalism satisfying the more important requirements would be preferred to the one with the less important requirements.

Requirements validation: To find out whether the correct and complete set of requirements was identified, the requirements are validated according to their semantic properties. The semantic properties of a requirement are based on the software requirements specification (SRS) semantic properties described in [39] and the model requirements properties defined in [11]. Namely, each model requirement specification (MRS) should be:

verifiable: represented by the accuracy with which the requirement can be transformed from higher levels of abstraction into its current form, i.e. how accurately a given abstract action requirement can be mapped to the concrete requirement in the described problems.

valid: represented by the accuracy with which the requirement represents the real need, i.e. does the requirement accurately represents the needs of the described problem.

clear: represented by the degree to which the requirement is unambiguous and understandable, i.e. are there any doubts about the requirement's meaning.

complete: represented by the degree to which all parts of a requirement are specified without missing information, i.e. is there something missing in the requirement's specification.

consistent: represented by the degree to which the requirements are specified using uniform notation, terminology, and symbology, and any one requirement does not conflict with any other.

feasible: represented by the degree of difficulty of implementing a single requirement and simultaneously meeting competing requirements, i.e. can an action with a given requirement be easily modelled without contradicting other requirements.

testable: represented by the degree to which the requirements can easily be tested. A testable requirement is one that is specified in such a way that a pass/fail criteria can be derived from its specification.

traceable: represented by the degree to which the requirements related to a particular requirement can easily be found.

One way to discover if the properties of the different requirements hold, could be to use a questionnaire that aims at identifying problems within the requirements specification (see Appendix C). The data obtained from the requirements validation is then analysed in order to discover possible problems, discrepancies, or missing specification details. In the case when a questionnaire is used for validating the requirements, the results could be evaluated by analysing the following factors[6].

- Given n evaluation participants, the number of participants who understood a requirement specification is calculated according to $\sum_{i=1}^{n} r_i$, where $r_i = 1$ if the person i understood the requirement specification, and 0 otherwise. This could show us whether the requirement is defined in a clear and unambiguous way and can point out specifications that need to be redefined or refined.

- Given n evaluation participants, the number of comments for each requirement is calculated according to $\sum_{i=1}^{n} c_i$, where c_i indicates the number of comments a participant i provided for the requirement in question. This can also pinpoint to under-defined requirements specifications, and the comments themselves can provide useful information about how the specification can be improved.

[6]Of course, depending on the hypotheses one has posed, other analysis methods could be applied.

- Given n evaluation participants, the average score of a requirement property is calculated by $Median(s_i)$ where s_i is the score a participant i has assigned to the property. Here the median is used instead of the mean because we assumed ordinal data. This shows how well the requirements were specified, and which aspects of the specifications are problematic.

5.3.1.2 Data collection

In order to test and evaluate the model to be developed, an experiment has to be conducted where its participants are acting in an environment supplied with sensors and where they are asked to follow a specific scenario. The experience showed that to prepare such experiment and to collect the data, one should take care of the following details.

1. Prepare the use case scenario to be recorded (what is the goal of the experiment; what behaviour the participants should exhibit; are there some behaviour restrictions).
2. Prepare the sensors infrastructure and test if it works and records in the expected way (as it is usually not easy to repeat the experiment in case of problems, it is a good idea to have a backup plan in case of sensors failure).
3. Make sure the participants are aware what exactly their tasks are and how to execute them.
4. Make sure there is enough storage / capacity / time available to record and store the whole experiment.

After the data is collected, one should understand it and later prepare it for the future model evaluation. According to Nisbet et al. [102, p. 51] there are four main issues for data understanding. These are the *data acquisition*, or how to find the data needed for modelling; the *data integration*, or how to integrate the data from multiple sources; the *data description*, or what the data looks like; and the *data assessment*, or with other words – how clean[7] the dataset is.

The data preparation for future use includes the *data cleansing*, or how to clean the data; the *data transformation*, or how to express the data variables; the *data imputation*, or how to handle missing variables; the *data weighting and balancing*, or the way in which the different classes are treated; the *data filtering*, or the treatment of outliners and unwanted data; the *data abstraction*, or the handling of temporal data; the *data reduction*, or the reducing of the amount of needed data; and the *data derivation*, or with other words – the creation of new variables. Nisbet et al. provide a detailed description of all the involved processes [102, p. 51–75]. Although the data preparation process was not discussed in this work, it is an essential part of the activity recognition process which has to be performed before the designer is able to test her model on the data.

5.3.1.3 Actions ontology

The actions ontology concerns with deriving the relevant action categories, their attributes and the relations they have to the rest of the actions in the ontology. The ontology is essential for annotating the recorded data, as it should provide the needed information about the naming conventions, the objects that can be assigned to an action, the locations at which the actions

[7]Here by *clean* we mean the presence of noise in the sensor readings, as well as missing readings or readings produced by faulty sensors.

could be executed etc. It should be coordinated between the project participants and ensured that all parties later use the same conventions. In other words, the ontology is the baseline by which the activities are to be labelled, and later developed into model actions with the corresponding properties. It also ensures that there are no differences in the way a given activity, or a type of activity is labelled.

To our knowledge there is no general actions ontology or a method according to which to choose the actions granularity. Many authors decide on the actions granularity depending on the sensors granularity. If the sensors are not able to capture an action, then it is too fine-grained to be included into the actions ontology [35, 152]. This however makes the annotation dependent on the available sensors and not on the executed actions. A context-aware approach to activity recognition should be able to reason beyond what the sensors are able to observe[8]. Thus, it could be reasonable to provide a more fine-grained action ontology where details such as locations, objects, users are also included in the annotation. When deciding on the fine-grained annotation one option is to annotate actions that are the smallest intentionally executed actions. In our case they comply with the definition of *atomic actions* given in Chapter 1. Such actions will be those that manipulate objects, or change locations in order to accomplish more complex activities.

Although there is no general structure for an action ontology, in our experiments we decided on a uniform action representation structure that allows us to obtain information about the action being executed, the users, locations, and objects involved in it. Such representation has the structure *action_user(s)_location(s)_object(s)*. That way during evaluating the model's ability to cope with context information, one can easily access the corresponding ground truth. Fig. 5.17 shows an example structure for the action move from the cooking problem. It can

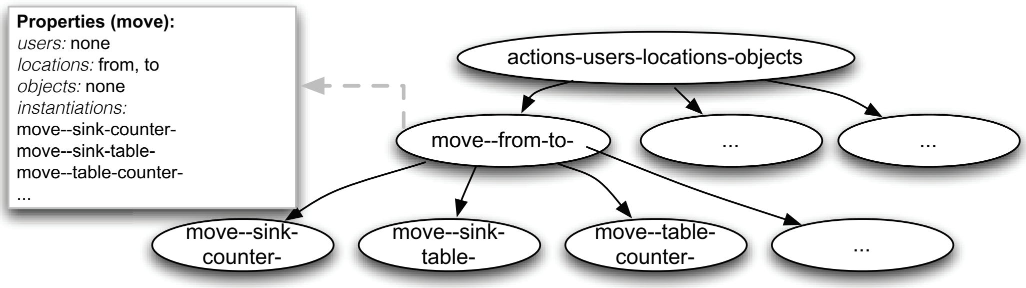


Figure 5.17: Example action ontology for activities annotation

be seen that the actions all include the action name, the users (when such are available), the locations, and the objects (when available).

5.3.1.4 Data annotation

The next step is the data annotation, where the data instances are assigned a label. In other words, the sensor readings are assigned the corresponding activity the user was executing at the time the readings were obtained. This step is important, as it provides the ground truth with which the model performance is evaluated. Furthermore, the annotation is used for generating the plans necessary for the model validation, and as observations to validate that the model is able to infer the given execution sequence and that the actions are assigned the correct durations.

[8]Chen et al. discuss the problem that sensors are sometimes not able to capture fine-grained activities such as preparing tea or coffee [23]. These are however typical daily activities that a system dealing with daily living activities has to be able to reason about.

To produce useful annotation, one should take care that the data is annotated in an appropriate way and format, as discrepancies in the annotation create problems during the validation and evaluation phases. The derived actions ontology is used for assigning the appropriate attributes to the corresponding action. It also takes care of the correct annotation format being used. To avoid such problems, one should make sure that:

1. There are no gaps in the annotation. This means that when one action ends, another one should directly start and there should not be any empty spaces between the actions.

2. The annotation is causally correct. This means that objects do not just appear from one place to another without being moved, people do not teleport, or do actions at places where they are not presently located, etc.

From previous experiences with publicly available datasets like the Carnegie Mellon University's Multi-Modal Activity Database [145] the annotation is usually filled with gaps, missing information and causally incorrect actions which could cause serious problems in the evaluation of a rule-based approach like CCBM. Having in mind that the annotation is nothing more than a set of labels without any underlying structure[9] we could conclude that nothing forbids annotating causally impossible sequences of events. This also is not a problem for approaches that just compare the labels to their estimation. However the moment more complex algebraic structures have to be validated, this poses a serious problem as the underlying annotation labels do not represent this structure. For example, the causally incorrect annotation will result in the definition of plans that are also causally incorrect, and these plans are needed to validate that the model is able to represent the appropriate behaviour. Furthermore, incorrect annotation will also reduce the model performance as to evaluate it, the estimated states are compared against the annotation states[10].

The annotation process can be considered as a simplified version of model-driven engineering where a procedure is developed that has different levels of abstraction and that describes how to derive a model from another one from the abstraction level immediately above it [45]. The process consists of four steps.

1. From the video log, the domain objects are identified. These are all objects within the environment that the user is interacting with. Additionally, the different locations within the environment are identified. These are all locations where the user is manipulating the objects. Atomic actions are identified based on the manipulated objects. As already mentioned these are the smallest intentional actions the user executes in order to achieve a goal.

2. The actions are abstracted from the corresponding objects. These abstracted actions correspond to the upper layers of the actions ontology and represent the action schemes that will later be implemented in the causal model. Additionally, the objects are abstracted to object classes that will later represent the parameters in an action scheme.

[9]That is regardless of the fact that we as a humans see a structure in the labels, e.g. what actions can be executed when etc. Still the labels themselves are not guided by any rules that define this structure, they are actually nothing more than a textual representation of the observed actions. This means that there are no rules against producing causally incorrect annotation.

[10]Of course, it is possible that the model incorrectly estimated the states in a way that they match the annotation. In that case the performance will not be decreased, but regardless, it will be misleading.

3. All instantiations of actions with the corresponding elements in the environment are identified. These populate the lower layer of the actions ontology and correspond to the grounded actions the causal model should contain.

4. Finally, the produced annotation sequences were checked for causal correctness to ensure that each annotation sequence would be a valid and causally correct path. This ensures that there are no acausal relations between the annotated actions.

5.3.2 Second development layer

The next three phases of the development process – design, implementation, and validation – fall under the second development layer. As explained before, the need of this layer comes from the combination of symbolic and probabilistic approaches. Were the model composed of only symbolic representation, the waterfall model could be easily adjusted to accommodate the development process. This is due to the fact that on symbolic level one can model each action in the model without influencing the rest of the action. In that manner one can recursively perform analysis, design, implementation, validation, and evaluation achieving the desired model behaviour. However, in the case when probabilities are present, it is not possible to develop the corresponding probabilistic structure together with the symbolic structure as a change in the probabilistic structure of one action will directly affect the probabilistic structure of all the remaining actions. For example, in Fig. 5.19 (page 167) the weight of one action is set to different values, while those for the remaining action stay fixed. However, when looking at the model dynamics during execution, one sees that also the rest of the actions are affected. In that sense, the symbolic structure of the model has to be developed before the designer is able to proceed with the probabilistic structure. Furthermore, it is essential to distinguish between the development of action selection heuristics and the development of action durations, because before proceeding with the action durations, one has to ensure that the correct action can be selected at all. This is due to the fact that incorrectly used action selection heuristics could assign too small probability to the correct action, rendering it improbable regardless of the fact that it is a valid execution sequence. Considering the probabilistic structure, it is also incorrect to start with assigning action durations before assigning correct action selection heuristics, as in the case the action has incorrect heuristics, it will never be selected, thus the action duration can never be validated. This interaction between causal structure, action heuristics, and action durations results in the need of introducing the second model layer, where the symbolic structure is developed first, then the action selection heuristics, and finally the corresponding action durations. The exact procedure of this process is explained below. The numbers in it correspond to the numbers in Fig. 5.18.

1. Choose design solutions for the actions to be modelled (e.g. causal relations, representation of context information, observations that correspond to the actions).

2. Based on the problem description, create a minimal model that is able to explain the problem. Avoid model overfitting by considering all variations of the behaviour simultaneously, rather than concentrating only on one of the datasets. One can create a minimal model by incrementally adding all actions with only the minimal set of necessary constraints.

3. Compile the model in order to check for syntactic and semantic errors.

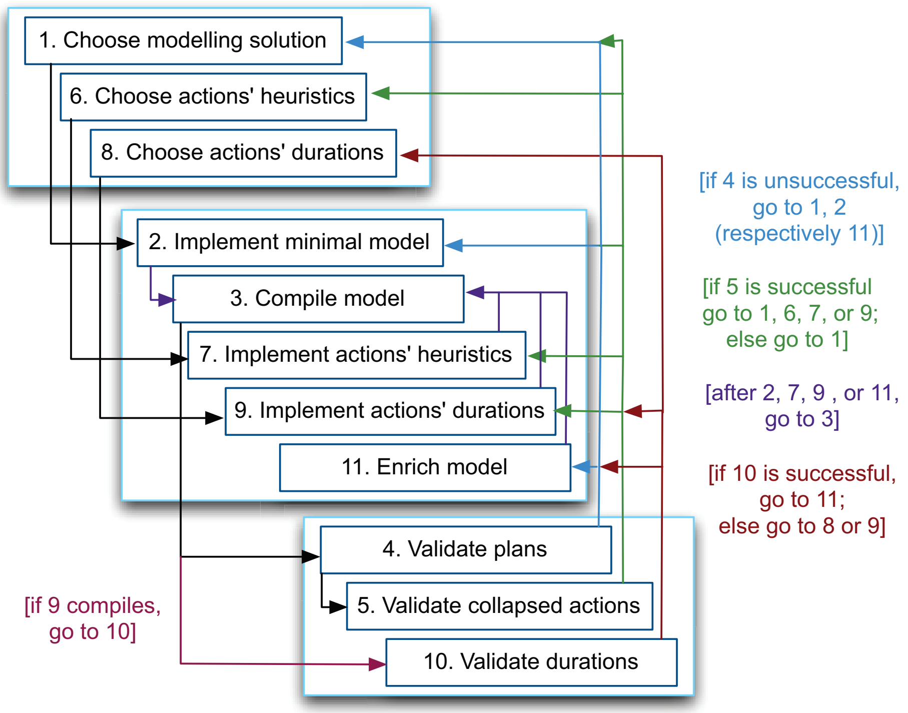

1. Choose modelling solution
6. Choose actions' heuristics
8. Choose actions' durations
2. Implement minimal model
3. Compile model
7. Implement actions' heuristics
9. Implement actions' durations
11. Enrich model
4. Validate plans
5. Validate collapsed actions
10. Validate durations
[if 4 is unsuccessful, go to 1, 2 (respectively 11)]
[if 5 is successful go to 1, 6, 7, or 9; else go to 1]
[after 2, 7, 9 , or 11, go to 3]
[if 10 is successful, go to 11; else go to 8 or 9]
[if 9 compiles, go to 10]

Figure 5.18: The second layer of the development process that includes designing, implementing and validating a model.

4. Validate the model with the plans generated from the annotation. In the case the validation was successful, continue to the next step, else return to step 1 or 2.

5. Validate the models with the collapsed observations generated from the annotation. This shows whether the inference engine is able to find the plan given the observations. In case the step is successful, continue to step 6, else to 1 or 2; in case the action selection heuristics were already implemented and validated, continue to step 8; in the case the heuristics were unsuccessfully validated return to 6 or 7 .

6. Choose appropriate action selection heuristics. These could be the goal distance, action weights, cognitive heuristics [2, p. 132–137] etc.

7. Implement the action selection heuristics if applicable for the problem. Continue to step 4.

8. Choose appropriate action durations based on the domain knowledge, and on the appropriate durations probability distribution.

9. Implement the actions durations.

10. In the case the model with its durations compiles, validate the durations.

11. In the case the durations validation was successful, one can now enrich the model by adding more context information, creating more complex causal relations or more constraints for reducing the model size, etc. and repeat the process from the start.

In the following we discuss in detail the different steps of this process starting with design, then implementation, and finally validation.

5.3.2.1 Design

The second phase in the CCBM development process is the model design. It deals with conceptual solutions regarding the model that is to be implemented.

Based on the problem analysis, the model objectives and the requirements, first an appropriate modelling formalism is chosen. As it was already mentioned earlier, throughout this work we use CCBM for the reasons described in Chapter 3.2. Of course, it is possible that based on the designer preferences or objectives another modelling formalism is selected. In that case a modelling formalism is selected according to Formula 2.1 described in Chapter 2 (page 49).

Choosing appropriate modelling solutions: Appropriate modelling solutions are chosen for the different model objectives and requirements. It is possible to select several different modelling solutions and later evaluate what is the impact of the given solution to the overall model performance. As in this phase an implemented model still does not exist, the impact of the solutions on the performance is evaluated based on a priori assumptions about those solutions. Later after the model implementation and evaluation, the design solutions could be reevaluated and substituted with other solutions. For example, requirements for design solutions could be such as having a small model with less context information versus a bigger model that contains more prior knowledge and could provide more information about the user activities.

The concrete impact of modelling solutions for reducing the model complexity was discussed in Chapter 4 and could be used as a reference when making decisions about the model design.

Additionally, as CCBM is a causal approach where the actions are described in terms of precondition-effect pairs, the preconditions and effects for the different actions are extended based on the actions ontology. The same applies to the causal relationships between the different actions and elements in the environment. This is done based on the actions ontology and the behaviour-based requirements. The handling of conflicting requirements is also decided upon based on the requirements importance. Furthermore, the mappings of the involved objects and users to the observations are introduced, as well as the necessary functions in the observation model that will link the high level states to the observations.

Choosing appropriate action selection heuristics: An important part of the model design is choosing the action selection heuristics. These are the heuristics that during the inference influence the decision about the action to be selected. Such heuristics can be:

the *goal distance* that gives the shortest distance from the initial to the goal state, namely how many actions have to be executed to reach the goal from the initial situation. The goal distance is discussed in Formula 3.4 on page 55.

landmarks that, in the case the goal distance cannot be computed due to the size of the model, search for predicates that have to be set to true in order to reach the goal. The landmarks are discussed in Chapter 4 on page 128. More details about them can be found in [116, 115].

the *revisiting factor*, that indicates whether states that were visited once, can be visited again. With other words, when the revisiting is not allowed, it forbids the execution of actions repeatedly if they lead to already visited state. The revisiting factor is discussed in Formula 3.2 on page 55.

some *cognitive heuristics* which Anderson describes in his book *The Architecture of Cognition* [2, p. 132–137], for example:

- saliency which assigns weight to an action thus allowing it to be prioritised with respect to the remaining actions. The saliency is also discussed in Formula 3.3 on page 55.
- recency which allows the most recently executed action to be prioritised.
- specificity which allows the action that fulfils the most predicates of the goal state to be prioritised.

The heuristics have to be introduced before the actions' durations so that artefacts from the durations cannot cause incorrect choice of heuristics. The opposite – choosing first the durations and then the heuristics – could result in the inability to select the correct action which will result in inability to validate the action duration. As no systematic analysis of the influence of these heuristics was done so far, here we take the saliency as an example of choosing an action selection heuristic.

An action's saliency tells us whether some actions are more important than other actions and what priority they have. The weights can be assigned to the action's saliency, that is the importance by which it stands out relative to its neighbours. It is calculated according to Formula 3.3 in Chapter 3 (page 55). Assigning different saliency values to an action has influence not only to the action itself, but also on the whole model dynamics. Fig. 5.19 shows

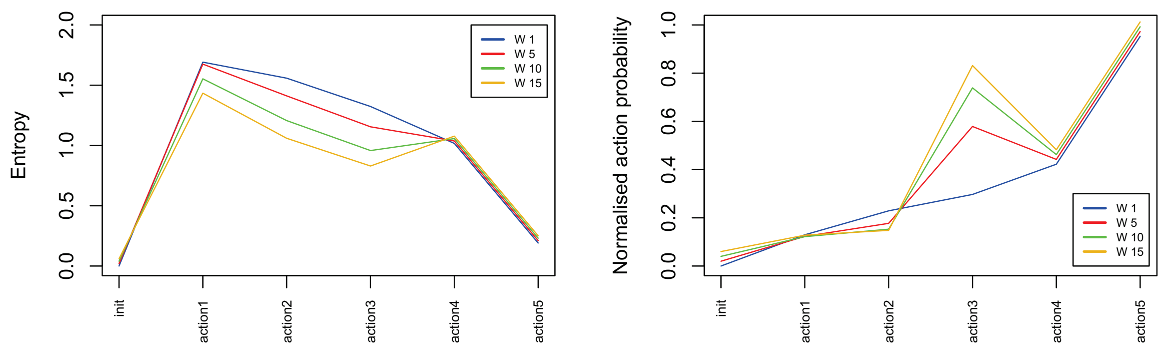


Figure 5.19: Influence of weight on the model entropy and action probability. In it the saliency of action 3 is changed while the rest of the actions have a default saliency of 1. The blue line shows the behaviour for saliency of 1; the red – of 5; the green – of 10; and the yellow – saliency of 15.

an example of how changing the weight of one action influences the whole model. In it, five actions are sequentially executed, where all actions have saliency of 1, except for action 3 that has varying saliency. It can be seen that increasing the action's saliency decreases the overall model entropy with a local minimum at action 3. At the same time it increases the action's probability and it can be seen that for bigger weights, the probability of action 3 is much higher than that of the rest of the actions.

In such manner whenever there is available prior knowledge about the importance of different actions in a model, one could assign weights to them. However, one should be aware of the changes they make to the actions' probabilities in the model. For example, in a model where from a given state there are many actions from which can be chosen, increasing the saliency of one of them will increase its probability of being selected before the other. However, the opposite is also true – assigning higher weight to action that should not be selected at a given situation will cause the wrong action to be selected, thus decreasing the model performance. As a conclusion, one could say that unless the weights are carefully selected, they could cause more problems than improve the model performance and it is advised to avoid them whenever possible.

The action selection heuristics can be considered as a separate topic that deserves further systematic investigation.

5.3.2.2 Choosing appropriate actions' durations

The next step in the model design is deciding on the model durations and the probability distribution representing them. These decisions are made based on the prior knowledge about the actions' durations, how often they appear in the problem, and whether there is a high variance in the durations of the different occurrences. Based on this knowledge one should look at the properties of candidate distributions and choose the most suitable. For example, Fig. 5.20

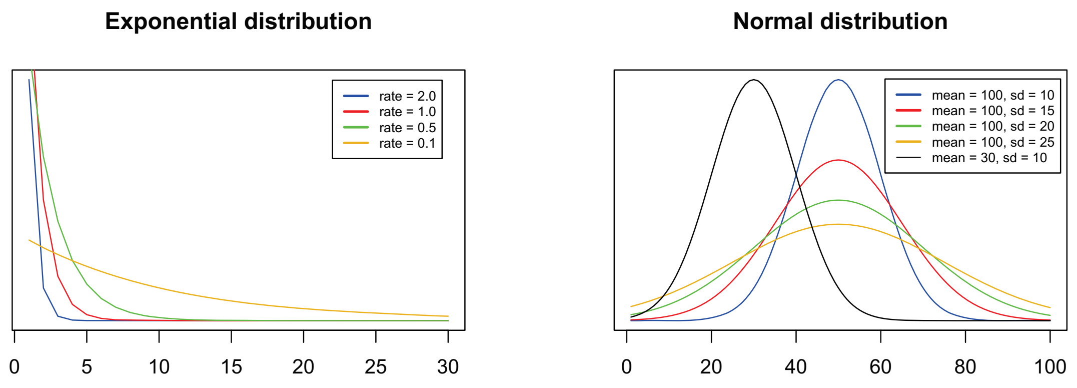


Figure 5.20: Different duration probability distributions. To the left the exponential probability distribution is shown, where the different colours represent different rate of decreasing the probability. To the right the normal probability distribution is shown, where the different colours represent varying mean and standard deviation.

shows two examples of different probability distributions with varying parameters. The x-axis is the time steps and the y-axis – the distribution density. The first is the exponential probability distribution which density is defined by the formula $f(x) = \lambda e^{-\lambda x}$ where λ is the rate parameter that defines the rate with which the density decreases. Such probability distribution will be suitable for short actions as the probability of the action happening at the beginning of the defined duration will be high and will decrease in a relatively short time, making long actions improbable. Additionally, based on the rate parameter, the length of the actions durations can be controlled. For example, if there is high variety between the durations of the different action's instances, then smaller rate will ensure that also a bit longer actions are included in the distribution. On the other hand, when the actions have similar length, a higher rate could be chosen.

The second example in 5.20 is the normal probability distribution that is controlled by its mean value and standard deviation according to the formula $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$ where μ is the mean of the distribution and σ the standard deviation. This kind of probability distribution is more suitable for long actions as it can be seen that the density pick is not at the beginning of the distribution. The mean allows moving the distribution peak along the x-axis, thus controlling the time at which the most durations are clustered. Similarly, the standard deviation allows concentrating the density at a given point, or distributing it more evenly along the x-axis, thus increasing the probability of action instances with varying length.

In such manner by applying the available prior knowledge about the actions' durations and comparing it with existing probability distributions, one can choose suitable durations for the model.

5.3.2.3 Implementation

Based on the design chosen in the previous phase, the initial model implementation is produced. The implementation is done with CCBM and the model compilation results in a runtime model with which an inference engine can perform probabilistic activity recognition Details about the inference engine are provided in [76]. The accompanying CCBM tool support [76] also allows discovering syntactic and semantic problems at an early stage which improves reevaluating the design solutions relatively early when an implementation problem is discovered.

Additionally, to support the model developer, the modelling toolkit introduced in Chapter 4 was created. It contains solution templates for reducing the model complexity.

Chapter 4 already discussed in detail what effect these solutions have on the model, how they should be implemented and in what situations they can be applied. Additionally, example implementations can be found in Appendix E. Furthermore, in this step the observation model with its functions is implemented in C++. It gives the connection between high-level activities and the observed data. It also determines which of the available information to be outputted during the inference process. Examples of how to build an observation model can be found in Appendix D.2 (page 216).

The implementation phase is part of all three phases in the second layer. It consists of first implementing the symbolic model which includes implementing all action templates in terms of preconditions and effects, implementing the initial and goal world state, the relations to functions in the observation model, and the observation model itself. Examples of symbolic models can be seen in Appendix E. Then the action selection heuristics are implemented. They can either be introduced in the action templates (e.g. the saliency), or as parameters for the model compilation (e.g. the revisiting factor). An example of action selection heuristic is the saliency defined in Figure 3.17 on page 76. Examples of how to use the revisiting factor can be found in [76].

Finally, the action durations are implemented in the action templates. Examples of duration implementation can be seen in Figure 3.17 on page 76 where the *:duration* slot defines a normal duration which exact value is later given in the problem description.

As the process of implementing the model is an iterative one where different steps from design, implementation and validation interleave with each other, there is no separate implementation phase in the standard waterfall model sense.

5.3.2.4 Validation

As it can be seen in Fig. 5.18, the validation of the model consists of three parts: validating the plans, validating whether the model is able to perform state estimation, and finally, validating the actions' durations. Below we look in more detail into these three steps.

Plan validation: In order to validate the model, a plan or a set of plans are generated based on the annotation. They represent existing execution sequences of the user behaviour for the given problem. An example of a plan can be seen in Fig. 5.21. In it the number indicates the time stamp when the action is executed, the asterisk indicates that the action at this time stamp is a new action. Alternatively, if the action has started in a previous time stamp the asterisk will be missing. Finally, the grounded action that has to take place is written in brackets.

```
0,*,(wash hand)
1,*,(wait)
2,*,(move sink counter)
3,*,(take carrot counter)
4,*,(move counter sink)
5,*,(wash carrot)
6,*,(move sink counter)
7,*,(take knife counter)
8,*,(put carrot cutting-board)
```

Figure 5.21: Extract of a plan from the cooking task.

Fig. 5.18 showed the process of validating and improving the model. The plan(s) could be created based on the annotation. The plans are then fed to a validator that proves whether the model contains such plan in its description. If that is not the case, then the model developer should return to the implementation phase and check whether the actions in the model are correctly specified and whether there are some contradictions in these specifications. One should try to identify problems in all available plans and not to concentrate on a single plan as our experience showed that this usually leads to model overfitting.

Although the annotation should be causally correct, it is sometimes possible that the plan is actually acausal, which results in the model's inability to explain it. In such cases one should return to the actions annotation process and correct any causal discrepancies in the annotation.

Model validation based on state estimation: As the developing of a model with good performance is an incremental process, one should try to perform activity recognition as early as possible directly with the first working model prototype. This could point out at problems that otherwise will be discovered when the final product is built. To validate the model, observations based on the annotation are generated and used for inferring the user(s) actions and states. The generated observations could have two forms – either a collapsed form (namely each action is represented by a single observation) so that the durations are not taken into account, or an extended form where the observations for a given action correspond to the actual actions' durations. An example of such observations is given in Fig. 2.3 on page 24. In it the observations are presented either with 1.0 when the observed even was sighted, or with 0.0 when it was not sighted. In the case of collapsed observations, each new event will be represented just by one sighting, while in the case of durative observations it will be represented by as many events as time steps it was observed in the experiment.

The state estimation based on the collapsed observations form aims at pointing out at the following problems.

- Problems with the action selection (when the incorrect action is selected because the correct one was assigned a low probability). On a causal level the solution to this problem would be to check how many actions are possible at the time step the problem appeared and try to narrow them down by introducing additional constraints; or to introduce better action selection heuristics.

- Problems with the action definition (incorrectly used expressions or predicates) that do not appear during the plan validation but that emerge during the state estimation.

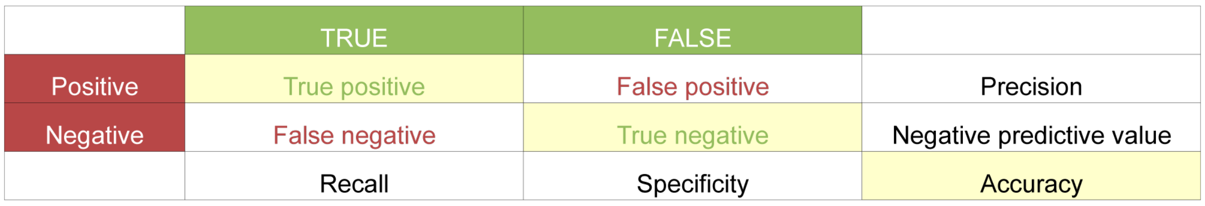
Durations' validation: To validate the durations, the extended observations are used. As explained in the design phase, the actions durations are based on prior knowledge about the relative actions durations and their variance. Depending on that information also the durations' probability distribution is selected. A good practice is to choose several different distributions and test them to see which one is most suitable. Additionally, before testing the durations, it is desired to test the model using exact durations[11] calculated from the observations. This is done to make sure that any other problems are not due to the actions' durations.

5.3.3 Evaluation

The last phase of the model development is the evaluation phase. In this phase the model is used for recognising user activities and for evaluating how accurately these activities were recognised. Additionally, the phase also provides information about the model correctness and could point at problems that could not be identified earlier. Based on the nature of the model application, its performance evaluation seriously depends on the underlying sensors infrastructure and the correctness of the ground truth. During the state and activity estimation, it could turn out that there are discrepancies between the model and the observations provided by the sensors. In such case the developer should return to the implementation phase and resolve these problems.

5.3.3.1 Performance evaluation

The first part of this phase is the performance evaluation. In the field of activity recognition that is actually the part that provides the most significant information – namely is the model able to perform activity recognition with a high accuracy. To acquire this information, the output of the activity estimation is compared to the ground truth (the annotation), and the percentage of accurately recognised instances is calculated. In order to build a model with high performance and to avoid unexpected problems late in the model development, it is recommended to start evaluating the model as soon as there is a compilable version and to continue iterating between evaluation and model implementation until the desired model performance is achieved. Below some evaluation metrics one should pay attention to are listed. Most of them are calculated based on Fig. 5.22 that shows the matrix for calculating accuracy, precision, recall, and specificity for a two-class problem. For a multi-class problem, a confusion matrix like the one in Table 5.1 is used.



	TRUE	FALSE	
Positive	True positive	False positive	Precision
Negative	False negative	True negative	Negative predictive value
	Recall	Specificity	Accuracy

Figure 5.22: Table for calculating some evaluation metrics. Based on this table the accuracy, precision, recall, and specificity can be calculated.

[11]These are durations that are taken from the recorded datasets.

Confusion matrix: A model's confusion matrix is a table that allows visualising the performance of a model. Each column of the matrix represents the instances of a predicted class, while each row represents the actual instances in a class. In other words, it shows whether the model is confusing classes, or in our case – actions. Table 5.1 shows an example of a confusion matrix for three different actions, where the diagonal contains the instances that were correctly identified while the remaining cells show how many instances were incorrectly recognised and with which actions they were confused.

		True		
		Action1	Action2	Action3
Predicted	Action1	10	1	2
	Action2	0	12	1
	Action3	0	0	15

Table 5.1: Confusion matrix

Accuracy: In the case of activity recognition, the accuracy represents the degree of closeness of an estimated behaviour to the actual behaviour. Accuracy for a two-class problem is calculated according to Formula J.1 (page 245) and according to Formulas J.2 (page 245) or J.3 (page 245) for a multi-class problem. Both formulas can be found in Appendix J.

Precision: The model's precision represents the proportion of positive test results that are correctly recognised. It reflects the probability that a positive test represents the underlying condition being tested for. Precision in a two-class problem is represented by Formula J.4 (page 245), and by Formula J.5 (page 245) for a multi-class problem.

Recall: The model's recall represents the proportion of actual positive instances that are correctly identified as such. It reflects the test's ability to identify positive results. Recall for a two-class problem is defined by Formula J.6 in Appendix J (page 246). In the case of a multi-class problem, the recall equals the accuracy and is calculated according to Formula J.3 (page 245).

Specificity: The model's specificity represents the proportion of actual negative instances that are identified as such. It reflects the test's ability to identify negative results. Specificity is represented by Formula J.7 (page 246) for a two-class problem, and according to Formula J.8 (page 246) for a multi-class problem. Both formulas can be found in Appendix J.

Hypothesis testing and results significance: It was mentioned earlier that one should pose hypotheses regarding the model performance, results significance, or simply data distribution. In order to test whether a given hypotheses is accepted or rejected based on the model results, different statistical methods can be used. Depending on the available data, its underlying distribution, the number of samples, the number of categories to be compared etc., different statistical methods can be used. A useful guide for selecting the appropriate method is provided in [73] where categorisation of the various problems and the corresponding statistical tests is presented. Fig. J.1 in Appendix J (page 247) shows the proposed ontology where it can be seen that based on the problem (e.g. how many groups are to be compared), on the underlying distribution (e.g. normal or non-normal), the type of data (e.g. nominal or ordinal) one can make a decision about the test to be applied. After selecting the appropriate tests, the hypothesis testing strategy is the following. It is based on the one proposed by Cohen [26, p. 110] and on practical experiences with hypothesis testing.

1. A null hypothesis is formulated, as well as an alternative hypothesis in the case the null hypothesis is rejected. For example, in our case this could be that two models are performing identically.
2. A sample from each category to be tested is obtained. In our case this could be the performance values of the two models to be compared where each category is represented by a model.
3. The selected statistical test is performed on the available samples. For example, if only two models are compared and we do not have any knowledge about the underlying distribution, Wilcoxon test can be performed.
4. The probability of obtaining the tested sample given the null hypothesis is compared.
5. If the probability is low, the null hypothesis is rejected and the alternative is accepted. High probability will indicate that the null hypothesis explains the condition for which the sample is being tested.

Examples of performance evaluation can be seen in Chapter 3 in e.g. Fig. 3.10 on page 66 where the accuracy, precision, and specificity for the meeting model were plotted. Examples of hypothesis testing can be seen in Appendix H where Friedman test was performed, or in Appendix I where Wilcoxon test was performed.

5.3.3.2 Success criteria evaluation

Finally, the model success criteria is revisited and empirically proven or disproven whether the model is able to satisfy it. The success criteria is based on the model objectives, their maximisation functions and the associated thresholds. Model that has performance under the objectives' threshold is considered unsuccessful, or in special cases – partially successful. Whether the model is partially successful is defined by the requirements and expectations of the model designer and stakeholders.

Given Equation 5.1, an objective is successful if

$$N \geq T, \tag{5.2}$$

where N is the result from the model with maximum performance and T is the objective's threshold value. N is calculated according to Formula 5.1.

5.3.4 Documentation

Important part of the development process is the generation of documentation. This is done throughout the whole process, with different types of documents and information outputted during the various phases. The reason for that is the need of documenting every decision or change along the way of developing the final product, as it is difficult for both developers and stakeholders to remember decisions made in previous phases. This is especially true for complex problems, or for problems where more than one person are involved in the project. For example, changes in the annotation will reflect the future model design and implementation, but

Table 5.2: Documentation provided in the different development phases

Analysis	Design	Implementation	Validation	Evaluation
1) problem description	1) causal relations between actions	1) description of modelled actions	1) plans to be validated	1) evaluation scripts
2) actions to be modelled	2) conceptual modelling solutions	2) description of available context	2) log of changes	2) description of parameters used for results reproducibility
3) environment elements to be modelled	3) action durations	3) description of model parameters		3) description and interpretation of model results in terms of accuracy, precision, recall, specificity, etc.
4) datasets description	4) action heuristics	4) log of model changes		4) visualisation of model results
5) annotation description	5) relation between high level actions and observations			5) log of changes
6) requirements specification	6) log of changes			
7) success criteria and hypotheses				
8) description of the collapsed actions' observations				
9) log of changes				

if these changes are not documented or conveyed to the model developer, the result could be a model inconsistent with the ground truth.

Table 5.2 shows the documents associated with each development phase. It can be seen that each step also provides a log of the changes done, as it is often the case that changes in modelling decisions, annotation, model, evaluation scripts etc. are not described in detail. This leaves the model user as well as the developer with sometimes unclear current model version.

5.3.5 Results reproducibility

A serious issue in the model development and evaluation is the results reproducibility. By just changing a single parameter in the model, removing or adding a predicate, the model could perform in completely different manner. And often such changes are not mentioned, so later when trying to reproduce given results, one discovers that this is an impossible task. Even more, changes in the ground truth, or the scripts used for running or evaluating the model, will most probably also result in different outcome. In a previous work we already discussed such issues and the resulting problems they could create [82]. Although not a phase in the development process, but rather a need throughout the process, here we propose some practices one should perform to ensure that the reported results are reproducible.

Data

- Make a copy of the original data that is not a subject to changes.
- In case data preprocessing is made, it has to be documented together with the corresponding scripts, tools and parameters used.

Annotation

- Avoid changes in the annotation.
- If such are necessary, provide documentation for each evaluation that describes which version of the annotation was used.

Model

- After each change of the model provide information with the time stamp, the change made and the reason for the change. This could be done in the header of the model, in a separate file, or through a version control system to ensure that no information about the model changes is lost.
- Provide a copy of the model used for producing the results together with the results.
- Provide a copy of the observation model used for producing the results.

Inference

- Automate the process so that the information about parameters with which the model was compiled and the inferred data produced is not lost.
- In the case of approximate inference, avoid using random numbers and instead set the values to such that can be reproduced later.
- Provide documentation about the time stamp when the inference was performed, the tool version that was used, the specific parameters with which the execution script was called.

Evaluation

- Automate the evaluation process in the form of e.g. evaluation scripts so that the evaluation can be reproduced.
- Provide a copy of the evaluation script (in the form and version it was used) together with the data that was evaluated.

Results

- Store the results in a directory(ies) with a reproducible path(s).
- Avoid reporting results with unclear origin, unknown inference settings, or evaluation scripts.

General

- For each experiment create a directory structure that allows easy reproduction of the existing scripts and a better traceability of all the elements involved in the activity recognition and evaluation process. Fig. K.1 in Appendix K (page 249) shows an example directory structure. It can be seen that for each model there is a separate directory that contains the model components (domain files, problem files and observation files). On the same level there is a directory containing the data used as observations for the experiment and an additional *Model results* directory that is automatically generated by the experiment script. The script also generates all files in the results directory and it can be seen that together with the estimated behaviour, there are copies of the model files, the script that was used for running the experiment, and a text file containing all additional information necessary for repeating the experiment.

5.4 Discussion

The development process introduced in this chapter is based on experiences made on modelling with CCBM and on the state of the art models also discussed in this chapter. The development process however was never tested on a new problem in order to validate its advantages to intuitive modelling. It could also be validated on the problems discussed in this work, however that has to be done from a different designer to avoid the biased expertise already gathered during the intuitive model development. For that reason the validation of the model by applying it to new problems is left to a future work. On the other hand, here we can list the obvious advantages it has to the intuitive modelling.

- It provides a detailed procedure describing the model-based activity recognition lifecycle that clearly states when and what practices have to be applied in order to develop and evaluate a model for a given problem. On the other hand, the intuitive process discussed in Chapter 3 is based on trials and errors often resulting from the designer's inexperience, or from the inability of the experienced designers to convey the process of modelling.

- The development process poses clear model objectives and clear hypotheses. It also points at a concrete point during the model development that the hypotheses and objectives have to be defined. On the other hand, the intuitive modelling does not explicitly state when to pose the hypotheses which often leads to the situation where the hypotheses and objectives are defined after the results are obtained, thus adjusting them to the results instead to the problem at hand. This makes the objectives and hypotheses biased.

- It provides a clear mechanism for building correct annotations. It also puts a stress on the problem that this is an important step in the model development. On the other hand, the intuitive development process suffered greatly from the lack of correct annotation that resulted in time consuming procedure of trials and errors in order to identify the need of such annotation and the subsequent procedure of producing it. Of course, an experienced designer who has faced this problem will not fall into the same pitfalls, but for inexperienced designers it will be a major issue.

- It provides a clear procedure for designing, implementing and validating the model. This is an essential contribution to developing models that combine symbolic and probabilistic structures. On the other hand, the intuitive process consumed incredible amount of time before coming to the realisation that symbolic structures have to be explicitly built before the probabilistic structure. This is due to the fact that in a software engineering sense one does not distinguish between the implementation of the causal and the probabilistic structure or the order in which to implement them, because they are just two different elements that have to be developed. This once again can be avoided when the designer is experienced but for a novice that would be a problem.

- It gives clear guidelines about validating the model and the associated procedures. This is an aspect of activity recognition that is not well documented as the designers strive to obtain high recognition rate without thorough model validation beforehand. This is once again an aspect that is caused by the data analysis view where the model is validated on a subset of the data for performance and not for consistency in the model structure.

- It provides an extended summary on methods for model evaluation and significance testing. Although there are extended works on methods and procedures for data analysis and evaluation, it is often the case that the inexperienced designer has to discover for herself what are meaningful methods and evaluation procedures. In that sense the development process provides guidelines for how to proceed in such situations.

- It addresses the problem of results reproducibility, which is a major issue in activity recognition. Serious amount of time during the intuitive model development was spent in attempting to reproduce obtained results. This is due to variety of factors such as the usage of random seed based on the current system time, or using version of the model that was later changed without documenting the changes. Once again one could argue that an experienced model developer would have avoided this problem but for the inexperienced this is a time consuming process of discovery.

In general, the advantages of the development process to an intuitive one may not lie in the model performance. They are however easy to see (and even appreciate) in the way in which the model is developed, and the corresponding time it takes to model something without guidelines, compared to developing the same when one can follow clear instructions. One could argue that an experienced designer will not run into all the problems discussed above. However such designer will not be the one in need of guidance. And even more, to reach the state in which she does not need to follow an explicit process, she would have spent the same amount of time and faced the same problems in previous projects. In that sense, an intuitive model could potentially yield better results, but the amount of time and effort spent to produce such results will be much more. And there will be no guarantee that the reported results can be reproduced unless clear structure and guidelines are followed.

Naturally, the question arises whether the introduced development process represents the designer needs better than state of the art processes. Unfortunately, there are no development processes proposed in the field of activity recognition, at least not in the model engineering sense addressed here. For example, Lyons et al. introduce a toolkit for gesture and activity recognition [91]. According to them it enables the development of gesture based activity recognition applications. Still it provides just an abstraction to machine learning algorithms suitable for modelling and recognising gestures. It however does not address the problem of how to develop a model.

Another example is the lifecycle of ontology-based activity recognition proposed by Chen et al. [24]. In it they discuss the process of developing ontology-based systems which basically reflects the ontology-based processes discussed in the beginning of this chapter. It also addresses how an ontology-based model is build and how activity recognition is performed. It however, does not address the problem of introducing probabilistic structures as such are not used. It also does not concern itself with the problems of correct annotation production, model design, validation, evaluation, or results reproducibility. It is a general ontology-based process that does not fit the needs of symbolic models making use of probabilistic inference.

Another process for activity recognition proposed by Hartmann aims at describing the process of activity recognition of a worker. It includes the collection of the sensor data, the sensor data processing, followed by the low level activity recognition, which is concluded with the recognition of more complex activities [63]. Still the proposed process is nothing more than a variation of the typical data analysis processes discussed in the beginning of this chapter. It does not provide any information about the way in which the underlying model for activity recognition has to be developed but rather concerns itself with the typical data mining and model learning problems.

In that sense, the process proposed here is the first reported attempt to combine software engineering techniques with typical data mining processes. It is also the first attempt at dealing with the effects of combining symbolic and probabilistic representations in a human behaviour model for activity recognition. As a conclusion, we hope that it will provide valuable guidelines for the new and inexperienced researchers that want to explore the potential of symbolic models for activity recognition.

5.5 Outlook

In this chapter we investigated different development processes from the fields of software engineering, modelling and simulation, knowledge based systems, ontological design, and data mining. The results showed that each field has its specific needs that are covered (sometimes partially) by the development processes. There are also processes that can be adopted to fit the needs of CCBM, yet there are no such that can be taken directly and applied to model based activity recognition. Were they to be used in their current form, some of the needs discovered during the model analysis in Chapter 3 will not be satisfied. For that reason, a development process was proposed that adopts already existing development models and combines workflows from software engineering and data mining together with experiences made during modelling with CCBM. The resulting development process provides a workflow for developing models for activity recognition problems that enclose the steps from recording an experiment, through building the model, to evaluating it and documenting the results. Although centred on CCBM, the process can be looked upon as a guideline for developing any rule-based activity recognition system that makes use of probabilities.[12].

[12]Of course, some phases will differ depending on the modelling formalism and its specification, but the general steps through which a data analyst has to go through in order to develop a model based activity recognition system will be the same.

Chapter 6

The Art of Modelling - A Discussion

"It does not appear reasonable, however, to suppose that one could provide a general "recipe" for making models, nor that one could do very much more than modestly enhance the process of developing intuition."
William T. Morris

Chapter Summary: *This is the conclusion chapter in which Computational Causal Behaviour Models as a solution to activity recognition problems are discussed. Additionally, the application of such models to real world problems is discussed as well as the need and applicability of the corresponding development process. The chapter then summarises the goals and achievements of this work, as well as its limitations. Finally, it discusses possible future research based on this work.*

Chapter Sources: *This chapter contains previously unpublished work.*

Questions to be answered in the chapter:

Do we need Computational Causal Behaviour Models for activity recognition problems? (In Section 6.1)

Do we need the CCBM development process? (In Section 6.1)

What are the goals and achievements of this work? (In Section 6.2)

What are the limitations of the introduced work? (In Section 6.3)

What future research can be done based on this work? (In Section 6.4)

As William T. Morris argues in his article *On the Art of Modelling* [98], the process of modelling could be greatly considered as a creative and intuitive one. In it the designer relies more on her creativity and practice then on some underlying process. On the other hand, he point out that although one *could not provide a general "recipe" for making models*, to develop ones intuition, *this process of development must have a beginning*. In other words, even in the distant year of 1967 Morris points out that to build successful and appropriate models, there should be some general guidelines that can be followed.

This is especially true for inexperienced designers that are faced with the problem of either using their (sometimes misleading) intuition, or imitating the experienced designers. The latter however brings the problem of following some pattern or steps without possessing the understanding of their necessity. And while developing her first models with often unexpected or surprising effects, one often wonders about the choices made, both in terms of modelling formalisms, implementation strategies, and approach all together.

This work attempted to find the answers to all those questions that an experienced designer and data analyst might find trivial, but that for a beginner are essential for building successful models. In this chapter we point out some questions that might have arisen during the thesis. Furthermore we discuss the chosen formalism and approach and their applicability to real world problems; and finally, the need of a development process and its usefulness in actual problems. Moreover, the achievements and limitations of this work are summarised and the relevant future work is discussed.

6.1 Is there a real need of Computational Causal Behaviour Models for activity recognition and the corresponding development process?

Presenting the model results in Chapter 3 was preceded by a relatively long process of discovering the modelling requirements, evaluating different formalisms and finally choosing a rather complex modelling approach that required incredible amount of time and effort to make ”things work right”. So one may wonder, do we really need such complicated and relatively new approach for which there are no guidelines to show us the pitfalls. Would it not be much easier to use some already well established formalism with an also well established workflow and save ourselves the frustration accompanying any new approach?

This are, of course, valid questions and one can easily imagine that using some classifier or some small HMM will provide the same, or even better results with a much less effort. Yet, there is not much challenge in such approaches – there is an ever growing amount of research concerning statistical methods that just label the data based on whatever they are trained to recognise, or probabilistic approaches that provide exact inference and there is no much that can go wrong. But what happens when the ever curious user wants to have more information about the situation she is in? Or when there is the need to reason about the current user states based on histories – did she got here by mistake or did she execute some previous action that could explain her presence here? In such situation just mapping the sensor readings to a predefined label is not enough. And exactly here comes the role of complicated approaches that are still not well investigated but that promise to be the answer (or one of the answers) to the growing demand for context awareness and adequate user support. Because in difference to approaches that just ”label” the data, CCBM provides the ability to incorporate extensive context information and reason about the user state and the environment surrounding it, based on some noisy sensor readings. And in difference with other symbolic approaches that rely on rules alone, CCBM provides the link between logic and probability by taking the high level states and binding them to a more flexible probabilistic inference engine. This then provides a mechanism for building models that consist of hundreds of millions of states and that are still able to reason about the user state in an adequate manner.

In the case of controlled experiments, that would not be a great issue as they are usually restricted to just a small set of actions and situations that can be represented by simple models. But for a real world problems, and that should be the ultimate goal of any activity recognition system, the ability to encode and later capture diverse behaviour variations in different situations, suddenly plays an essential role. Simply because otherwise the system would not be able to cope with the diversity of the human behaviour even in the simplest of situations.

For such problems exactly, CCBM shows to be a promising solution. Of course, one has to take into account that like any new approach, CCBM, and respectively the designers using it, are like toddlers learning to walk. There seems to always be a variety of unexpected problems that one has to understand before being able to go forward. Still, based on those problems we learn the potential and the limitations of CCBM. Even more they give us the insight needed for developing successful models without getting lost in the process. The CCBM development process was the result of gathering those experiences and incorporating them in a structured workflow. Even more, the corresponding modelling toolkit is a collection of modelling experience that makes the complexity problem easier to cope with. And in that sense, even though the art of modelling might be more creativity and intuition than a structured process, the CCBM development process and modelling toolkit provide the guidelines for how to build this intuition.

Another question that could arise is whether we need a whole development process and a modelling toolkit for an approach that is not well known. Although some might argue, the answer is yes, we need it, because the experiences made so far and the developed workflow could save enormous time and effort for the future scientists that use CCBM. And even if CCBM evolves into a new formalism, or is no longer used, the development process is a guideline for many model-based activity recognition problems as it bridges the gap between engineering and data analysis. This alone is a contribution to the field of activity recognition as it usually concerns itself with runtime models but does not stress on the process of developing the model itself. On the other hand, established development processes from other fields of research do not include specific data analysis and activity recognition issues.

In conclusion, the answer to the question *Do we really need CCBM and the corresponding development process?* is yes. We needed CCBM as it allowed us to incorporate large amounts of context information and reason about parameters beyond just the user actions as well as causes and effects of these actions. The corresponding development process is also necessary as it incorporates the experiences with CCBM made so far, but it also combines state of the art engineering and data analysis workflows. Finally, it gives the bridge between these two different fields of computer science and provides guidelines for any designer who wants to use the combination of model-based and probabilistic activity recognition.

6.2 Goals and achievements

This work started as an attempt at showing that generative rule-based models are suitable for context aware activity recognition problems. However in the course of research it became apparent that even seemingly simple problems are difficult to model without the presence of a structured and reproducible process. This led to the second part of the thesis – namely, introducing a development process for Computational Causal Behaviour Models and the corresponding modelling toolkit.

To summarise the achievements of this work regarding the first problem, it introduced three problems from our daily life, identified the requirements needed for expressing the behaviour dynamics of the three problems, and finally modelled them using Computational Causal Behaviour Models. The results showed that the selected rule-based approach is able to represent the user behaviour and to correctly recognise her actions. It was also shown that symbolic models are able to reason beyond the activity being executed. Based on the encoded context information one could also reason about the user whereabouts such as locations, environmental elements with which she interacts, or changes in the environment or the user herself based on the actions' execution. These results led to the conclusion that such symbolic models, and CCBM in particular, are a powerful means for model-based activity recognition, given they is applied properly.

To achieve the second goal of the thesis – namely, the introduction of a structured development process, the intuitive models were analysed for successful modelling practices, as well as for the underlying development structure. Based on the need of reducing the model complexity, the successful modelling practices were collected in the CCBM modelling toolkit and their influence on the model size and dynamics was investigated. It was shown that they are able to reduce the complexity in terms of states in the model, valid plans, and branching factor. It was further shown that this also increases the actions probability and decreases the model entropy. Relevant patterns were also tested on the three modelling problems, and the results compared to those of the intuitive models. The comparison showed that in general the patterns improve the model performance and reduce the model complexity.

Later, different state of the art development processes were analysed and their relevance to the field of model-based activity recognition was discussed. Based on this, and the intuitive development process identified previously, the CCBM development process was introduced. It combines typical software engineering phases with relevant practices from data analysis and activity recognition. Although limited to the specific usage of Computational Causal Behaviour Models, the general structure of the development process is applicable to any activity recognition system that makes use of symbolic models. Furthermore, the process can be considered as a guideline to building Computational Causal Behaviour Models in a structured manner. It also ensures traceability of modelling solutions and reproducible model results.

Looking back at the goals of the thesis listed in Chapter 1, one can conclude that the corresponding objectives were satisfied. The work was able to show that CCBM is suitable for activity recognition problems, and based on the gained experiences the development process was introduced along with a collection of modelling patterns.

6.3 Limitations

Along with the achievements of this work, there are, of course, some limitations that have to be considered. The first originates in the modelling formalism itself. The ability of CCBM to generate every valid execution sequence also leads to the problem that huge models are easily generated. Then to achieve the needed effect, additional constraints have to be added to the actions' preconditions and effects making it difficult to trace problems in the model implementation. Even the introduction of a structured development process is unable to entirely eliminate the traceability problems in a CCBM model. Moreover the resulting model contains enormous amount of states which often leads to decrease in its ability to correctly recognise the user actions.

Another problem arises from the combination of a rule-based approach with a probabilistic inference engine like the particle filter. This sometimes leads to situations where during resampling, some states are not sampled, thus removed from the available states although they exist in the model. It is also possible that although the correct action exists in the model, it has so low probability that it is never chosen. The influence of this combination of symbolic models with probabilistic reasoning was never investigated in detail but should be an important point in the future research concerning CCBM.

Furthermore, the introduced modelling toolkit contains only patterns concerning the model complexity. Other useful patterns that can ensure model reusability were never discussed. Additionally, the influence of the existing patterns on real models was investigated only in the case of combination of patterns. The influence on isolated patterns on the performance of a complex model was however never discussed.

Finally, the applicability of the development process to other rule-based approaches for activity recognition is still to be tested. The process however contains detailed guidelines for developing and evaluating CCBM models and we hope that they could be adapted also for other formalisms.

6.4 Future work

As already mentioned in the previous section, CCBM as a formalism and the corresponding tool still evolve. It is then reasonable to assume that in the future there will be changes in the way it looks, its underlying algorithms for probabilistic inference and functionality. Based on that, there are different aspects that deserve further investigation.

One essential problem in this approach is the huge state space that the causal models can generate. To cope with that, the modelling toolkit proposed some patterns for state space reduction. Still, the structured investigation of the size reduction on causal level could be further pursued.

On the other hand, the problem with the state space is due to the combination of causal modelling and probabilistic inference. The inference engine then is sometimes unable to assign the correct action probability. Thus, artificially reducing the state space on a causal level is equivalent to curing the symptom and not the cause. Another option could be instead to investigate different action selection heuristics that should solve the problem without the need of reducing the state space. The action selection heuristics were mentioned in this work but were not systematically investigated as they are a whole separate topic in themselves.

Another aspect that deserves further research is the creation of a catalogue with abstract action templates that can be directly reused without the need of further specialising them. At present that is problematic without the presence of appropriate action selection heuristics as a model with such templates can generate incredibly huge state spaces. On the other hand, once the inference engine is able to assign the highest probability to the correct action, a catalogue of abstract action templates will seriously reduce the designer workload associated with building the model.

Another interesting topic that came up during the experiments evaluation is the systematic incorporation of sensor information into the system model. It then should be investigated whether removing sensors and embedding their information into the causal description reduces the model performance. In case they do not reduce the performance, this could be a way of reducing the number of sensors needed for successfully recognising the user activities.

Moreover, the systematic identification and investigation of modelling patterns should be considered. The modelling toolkit in its current form contains only patterns for reducing the model complexity, but there are a variety of other modelling mechanisms that can serve as templates for reusable model components. Their systematic collection should be part of the future research on modelling with CCBM.

One issue of the modelling process that cost considerable time and effort was discovering implementation errors and bugs in the models. This was due to the lack of a tool for developing CCBM models. Without such tool the discovery of syntactic or semantic errors was time consuming especially in large models with a lot of constraints. The implementation of such tool should be a matter of future research. The tool then should provide an environment for designing, implementing, validating, and evaluating Computational Causal Behaviour Models. It should incorporate the development process and be able to provide automatic documentation and reproducible evaluation results. It should also provide a mechanism for tracking and discovering design and implementation errors.

Another aspect that can be pursued is the development of meta models that provide more user-friendly interfaces for building causal models. At the moment, as the model grows, it is sometimes difficult to track errors and even small errors could lead to hours of debugging. A meta model would then allow the designer to develop CCBM models without the need of manually coding the model. This, of course, would mean one more model transformation, and one should be well aware of the effects it would cause on the final probabilistic model.

Finally, it was shown that symbolic models can be applied to problem domains that generate large state spaces. Still the conducted experiments were performed in a controlled environment and did not contain the full variability a human behaviour could exhibit. A logical next step would be to show that they are also applicable to real world problems where a varying number of users interact with the environment in various situations. Furthermore, so far it was shown that the approach is able to reason about the user whereabouts but it was not empirically shown, that it is able to perform accurate intention recognition. And in real world situations, the system should be able to follow multiple goals and from various initial situations. Showing that symbolic approaches are able to cope with real situations will be the first step to providing adequate user assistance.

As a conclusion, there are many directions in which the research described in this thesis can continue. There are even more when one also considers the research on the probabilistic models and the inference engine. Generally, one could say that this work is not the end, but rather just the beginning of an ongoing and future research concerning Computational Causal Behaviour Models, their evolution and employment in the field of activity and intention recognition.

Bibliography

[1] M. Abidine and B. Fergani. Evaluating C-SVM, CRF and LDA classification for daily activity recognition. In *International Conference on Multimedia Computing and Systems (ICMCS)*, pages 272–277, Tangiers, Morocco, 2012. IEEE.

[2] J. R. Anderson. *The Architecture of Cognition.* Cambridge, MA: Harvard University Press, 1983.

[3] J. R. Anderson. *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press, 2007.

[4] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, 2004.

[5] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *Proceedings of the 4th International Conference on Ambient Assisted Living and Home Care*, IWAAL'12, pages 216–223, Berlin, Heidelberg, 2012. Springer-Verlag.

[6] R. B. Ash. *Basic Probability Theory.* Dover Publications, second edition, 2008.

[7] M. A. Azam and J. Loo. Using wireless proximity data to infer the behaviour of mobile people. *International Journal of Pervasive Computing and Communications*, 9(1):6–36, 2013.

[8] A. Aztiria, A. Izaguirre, R. Basagoiti, and J. C. Augusto. Learning about preferences and common behaviours of the user in an intelligent environment. In B. Gottfried and H. Aghajan, editors, *Behaviour Monitoring and Interpretation - BMI: Smart Environments*, volume 3 of *Ambient Intelligence and Smart Environments*, pages 257 – 288. IOS Press, Amsterdam, 2009.

[9] S. Bader, G. Ruscher, and T. Kirste. A middleware for rapid prototyping smart environments. In *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing*, pages 355–356, Copenhagen, Denmark, SEP 2010. ACM.

[10] J. C. M. Baeten. A brief history of process algebra. *Theoretical Computer Science.*, 335(2-3):131–146, May 2005.

[11] O. Balci. A life cycle for modeling and simulation. *Simulation: Transactions of the Society for Modeling and Simulation International*, 88(4):1–14, 2012.

[12] J. Banks, J. S. Carlson II, B. L. Nelson, and D. M. Nicol. *Discrete-Event-System Simulation.* Pearson, 2010.

[13] N. L. Batsch and M. S. Mittelman. World alzheimer report 2012: Overcoming the stigma of dementia. Technical report, Alzheimers Disease International, September 2012.

[14] C. Baum and D. F. Edwards. Cognitive performance in senile dementia of the alzheimer's type: The kitchen task assessment. *The American Journal of Occupational Therapy*, 47(5):431–436, 1993.

[15] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, May 1988.

[16] M. Born. *Natural Philosophy of Cause and Chance*. Oxford University Press, 1949.

[17] R. A. Brualdi. *Introductory Combinatorics*. Pearson, fifth edition, 2009.

[18] C. Burghardt, M. Giersich, and T. Kirste. Synthesizing probabilistic models for team activities using partial order planning. In *KI'2007 Workshop: Towards Ambient Intelligence: Methods for Cooperating Ensembles in Ubiquitous Environments (AIM-CU)*, Osnabrück, Germany, 2007.

[19] C. Burghardt and T. Kirste. Synthesizing probabilistic models for team-assistance in smart meetings rooms. In *Adjunct Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work, CSCW'08*, San Diego, CA, USA, 2008.

[20] J. Carifio and R. J. Perla. Ten common misunderstandings, misconceptions, persistent myths and urban legends about likert scales and likert response formats and their antidotes. *Journal of Social Sciences*, 3(3):106–116, 2007.

[21] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth. CRISP-DM 1.0: Step-by-step data mining guide. Technical report, SPSS, Chicago, February 2000.

[22] C.-W. Chen, A. Aztiria, and H. Aghajan. Learning human behaviour patterns in work environments. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 47–52, 2011.

[23] L. Chen, J. Hoey, C. Nugent, D. Cook, and Z. Yu. Sensor-based activity recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 42(6):790–808, November 2012.

[24] L. Chen and I. Khalil. Activity recognition: Approaches, practices and trends. In L. Chen, C. D. Nugent, J. Biswas, and J. Hoey, editors, *Activity Recognition in Pervasive Intelligent Environments*, volume 4 of *Atlantis Ambient and Pervasive Intelligence*, pages 1–31. Atlantis Press, 2011.

[25] J. Choi, R. Arriaga, H.-J. Moon, and E.-S. Lee. A context-driven development methodology for context-aware systems. In G. Lee, D. Howard, and D. Slezak, editors, *Convergence and Hybrid Information Technology*, volume 6935 of *Lecture Notes in Computer Science*, pages 429–436. Springer Berlin Heidelberg, 2011.

[26] P. R. Cohen. *Empirical Methods for Artificial Intelligence*. The MIT Press, 1995.

[27] D. J. Cook, G. M. Youngblood, and G. Jain. Algorithms for smart spaces. In A. Helal, M. Mokhtari, and B. Abdulrazak, editors, *The Engineering Handbook of Smart Technology for Aging, Disability and Independence*, pages 767–783. John Wiley & Sons, Inc., 2008.

[28] M. J. Crawley. *Statistics: An Introduction Using R*. Wiley and Sons, 2005.

[29] J. Crossman. Aspects and soar: A behavior development model. 27th Soar Workshop, Ann Arbor, Michigan, May 2007.

[30] S. Das and D. Cook. Designing smart environments: A paradigm based on learning and prediction. In S. Pal, S. Bandyopadhyay, and S. Biswas, editors, *Pattern Recognition and Machine Intelligence*, volume 3776 of *Lecture Notes in Computer Science*, pages 80–90. Springer Berlin / Heidelberg, 2005.

[31] D. Davidson. *Essays on Actions and Events*. Oxford University Press, 2001.

[32] J. Dawes. Do data characteristics change according to the number of scale points used? an experiment using 5-point, 7-point and 10-point scales. *International Journal of Market Research*, 50(1):61–77, 2008.

[33] A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, January 2001.

[34] C. Dietrich. Decision making: Factors that influence decision making, heuristics used, and decision outcomes. *Online Academic Student Journal Student Pulse*, 2(2), 2010.

[35] M. Donnelly, T. Magherini, C. Nugent, F. Cruciani, and C. Paggetti. Annotating sensor data to identify activities of daily living. In B. Abdulrazak, S. Giroux, B. Bouchard, H. Pigot, and M. Mokhtari, editors, *Toward Useful Services for Elderly and People with Disabilities*, volume 6719 of *Lecture Notes in Computer Science*, pages 41–48. Springer Berlin Heidelberg, 2011.

[36] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley and Sons, 2001.

[37] K. El Emam, S. Benlarbi, N. Goel, and S. Rai. The confounding effect of class size on the validity of object-oriented metrics. *Software Engineering, IEEE Transactions on*, 27(7):630–650, 2001.

[38] European Commission Eurostat. Old-age-dependency ratio. Retrieved: 06.03.2014 from http://epp.eurostat.ec.europa.eu.

[39] S. R. Fauk. Software requirements: A tutorial. In R. Thayer. and M. Dorfman, editors, *Software Requirements Engineering*, pages 1–22. IEEE Computer Society press, second edition, 1997.

[40] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, IJCAI'71, pages 608–620, San Francisco, CA, USA, 1971. Morgan Kaufmann Publishers Inc.

[41] G. Fischer. Context-aware systems: the 'right' information, at the 'right' time, in the 'right' place, in the 'right' way, to the 'right' person. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '12, pages 287–294, New York, NY, USA, 2012. ACM.

[42] R. A. Fisher and F. Yates. *Statistical Tables for Biological, Agricultural and Medical Research.* Oliver and Boyd Ltd., 1963.

[43] J. Fix and D. Moldt. A reference architecture for modelling of emotional agent systems. In L. Braubach, W. Hoek, P. Petta, and A. Pokahr, editors, *Multiagent System Technologies*, volume 5774 of *Lecture Notes in Computer Science*, pages 189–194. Springer Berlin Heidelberg, 2009.

[44] J. Fix, C. von Scheve, and D. Moldt. Emotion-based norm enforcement and maintenance in multi-agent systems: Foundations and petri net modeling. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 105–107, New York, NY, USA, 2006. ACM.

[45] F. Fondement and R. Silaghi. Defining model driven engineering processes. In *3rd Workshop in Software Model Engineering Satellite workshop at the 7th International Conference on the UML*, Lisabon, Portugal, 2004.

[46] D. Franklin. Cooperating with people: The intelligent classroom. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence, AAAI'98*, pages 555–560, Madison, WI, USA, 1998.

[47] D. Franklin and K. Hammond. The intelligent classroom: Providing competent assistance. In *Proceedings of the Fifth International Conference on Autonomous Agents*, AGENTS '01, pages 161–168, New York, NY, USA, 2001. ACM.

[48] M. Frenandez-Lopez, A. Gomez-Perez, and N. Juristo. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. In *Proceedings of the Spring Symposium on Ontological Engineering of AAAI*, pages 33–40, Stanford, USA, March 1997.

[49] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, December 1937.

[50] W. Fu and J. R. Anderson. From recurrent choice to skill learning: A reinforcement-learning model. *Journal of Experimental Psychology: General*, 135(2):184–206, 2006.

[51] Future Shape. SensFloor - large area sensor system, 2013. Retrieved: 07.03.2014.

[52] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wiseley, 1995.

[53] B. Geisler. An empirical study of machine learning algorithms applied to modeling player behavior in a "first person shooter" video game. Master's thesis, University of Wisconsin-Madison, 2002.

[54] M. P. Georgeff, B. Pell, M. E. Pollack, M. Tambe, and M. Wooldridge. The belief-desire-intention model of agency. In *Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages*, pages 1–10, London, UK, 1999. Springer-Verlag.

[55] A. Gerevini and D. Long. Preferences and soft constraints in pddl3. In *Proceedings of the ICAPS-2006 Workshop on Preferences and Soft Constraints in Planning*, pages 46–54, Glasgow, June 2006.

[56] M. Ghallab, A. Howe, C. A. Knoblock, D. V. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL — the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, October 1998.

[57] M. Giersich. *Concept of a Robust and Training-free Probabilistic System for Real-time Intention Analysis in Teams*. PhD thesis, University of Rostock, Rostock, Germany, October 2009.

[58] M. Giersich, P. Forbrig, G. Fuchs, T. Kirste, D. Reichart, and H. Schumann. Towards an integrated approach for task modeling and human behavior recognition. In J. Jacko, editor, *Human-Computer Interaction. Interaction Design and Usability*, volume 4550 of *Lecture Notes in Computer Science*, pages 1109–1118. Springer Berlin / Heidelberg, 2007.

[59] C. Girault and R. Valk. *Petri Nets for Systems Engineering: A Guide to Modeling, Verification and Applications*. Springer, 2002.

[60] A. Gomez-Perez, M. Frenandez-Lopez, and O. Corcho. *Ontological Engineering*. Springer-Verlag, 2004.

[61] A. J. Gonzalez and D. D. Dankel. *The Engineering Book of Knowledge-based Systems: Theory and Practice*. Prentice Hall, 1993.

[62] D. Gordon, J. Czerny, and M. Beigl. Activity recognition for creatures of habit. In *Personal and Ubiquitous Computing*, pages 1–17. Springer-Verlag, 2013.

[63] B. Hartmann. *Human Worker Activity Recognition in Industrial Environments*. PhD thesis, Karlsruher Institut fr Technologie, Karlsruhe, Germany, May 2011.

[64] A. Hein, C. Burghardt, M. Giersich, and T. Kirste. Model-based inference techniques for detecting high-level team intentions. In B. Gottfried and H. Aghajan, editors, *Behaviour Monitoring and Interpretation - BMI: Smart Environments*, volume 3 of *Ambient Intelligence and Smart Environments*, pages 257 – 288. IOS Press, Amsterdam, 2009.

[65] A. Hein and T. Kirste. Unsupervised detection of motion primitives in very high dimensional sensor data. In B. Gottfried and H. K. Aghajan, editors, *Proceedings of the 5th Workshop on Behaviour Monitoring and Interpretation, BMI'10*, CEUR Workshop Proceedings, Karlsruhe, Germany, September 2010. CEUR-WS.org. Best presentation award.

[66] L. M. Hiatt, A. M. Harrison, and J. G. Trafton. Accommodating human variability in human-robot teams through theory of mind. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, IJCAI'11, pages 2066–2071, Barcelona, Spain, 2011. AAAI Press.

[67] C. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[68] P. D. Hol. Resampling in particle filters. Technical Report LiTH-ISY-EX-ET-0283-2004, Division of Automatic Control, University of Linköping, Linköping, Sweden, May 2004.

[69] C. J. Hooper, A. Preston, M. Balaam, P. Seedhouse, D. Jackson, C. Pham, C. Ladha, K. Ladha, T. Plötz, and P. Olivier. The french kitchen: task-based learning in an instrumented kitchen. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 193–202, Pittsburgh, Pennsylvania, 2012. ACM.

[70] J. Indulska and K. Henricksen. Context awareness. In A. Helal, M. Mokhtari, and B. Abdulrazak, editors, *The Engineering Handbook of Smart Technology for Aging, Disability, and Independence*, Computer Engineering Series, pages 585–606. John Wiley & Sons, Inc., 2008.

[71] Institute of Electrical and Electronics Engineers. IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990), 1990.

[72] Institute of Electrical and Electronics Engineers. *IEEE Standard Computer Dictionary. A Compilation of IEEE Standard Computer Glossaries*. IEEE Press, Piscataway, NJ, USA, 1991.

[73] Jaykaran. How to select appropriate statistical test? *Journal of Pharmaceutical Negative Results*, 1(2):61–63, October 2010.

[74] K. Kaiser and S. Miksch. Treating temporal information in plan and process modeling. Technical Report Asgaard-TR-2004-1, Institute of Software Technology and Interactive Systems, Vienna University of Technology, Vienna, February 2004.

[75] J. Kiefer. Modeling individual strategic behavior in human multitasking. In *Proceedings of the 28th Annual Conference of the Cognitive Science Society*, pages 25–30, Vancouver, British Columbia, Canada, 2006.

[76] T. Kirste and F. Krüger. CCBM-a tool for activity recognition using computational causal behavior models. Technical Report CS-01-12, Institut für Informatik, Universität Rostock, Rostock, Germany, May 2012.

[77] T. Klug and J. Kangasharju. Executable task models. In *Proceedings of the 4th international workshop on Task models and diagrams*, TAMODIA'05, pages 119–122, New York, NY, USA, 2005. ACM.

[78] A. Kojima, M. Izumi, T. Tamura, and K. Fukunaga. Generating natural language description of human behavior from video images. In *Proceedings of the 15th International Conference on Pattern Recognition*, volume 4, pages 728–731, 2000.

[79] F. Krüger, K. Yordanova, C. Burghardt, and T. Kirste. Towards creating assistive software by employing human behavior models. *Journal of Ambient Intelligence and Smart Environments*, 4(3):209–226, May 2012.

[80] F. Krüger, K. Yordanova, A. Hein, and T. Kirste. Plan synthesis for probabilistic activity recognition. In J. Filipe and A. L. N. Fred, editors, *Proceedings of the 5th International Conference on Agents and Artificial Intelligence (ICAART 2013)*, pages 283–288, Barcelona, Spain, February 2013. SciTePress.

[81] F. Krüger, K. Yordanova, and T. Kirste. Tool support for probabilistic intention recognition using plan synthesis. In *Proceedings of the 3rd International Joint Conference on Ambient Intelligence*, pages 439–444, Pisa, Italy, November 2012.

[82] F. Krüger, K. Yordanova, V. Köppen, and T. Kirste. Towards tool support for computational causal behavior models for activity recognition. In *Proceedings of the 1st Workshop: "Situation-Aware Assistant Systems Engineering: Requirements, Methods, and Challenges" (SeASE 2012) held at Informatik 2012*, pages 561–572, Braunschweig, Germany, September 2012.

[83] J. Krumm and E. Horvitz. Predestination: Inferring destinations from partial trajectories. In *Proceedings of The International Conference on Ubiquitous Computing, UbiComp'06*, pages 243–260, Orange County, CA, USA, 2006.

[84] H. Lausberg and H. Sloetjes. Coding gestural behavior with the neuroges-elan system. *Behavior Research Methods*, 41(3):841–849, August 2009.

[85] S. Lee, Y.-J. Son, and J. Jin. An integrated human decision making model for evacuation scenarios under a bdi framework. *ACM Transactions on Modeling and Computer Simulation*, 20(4):23:1–23:24, November 2010.

[86] S. M. Lee, R. Remington, U. Ravinder, and M. Matessa. Developing human performance models using apex / cpm-goms for agent-based modeling and simulation. In *Proceedings of the Advanced Simulation Technologies Conference (ASTC)*, pages 49–53, Arlington Virginia, April 2004. The Society for Modeling and Simulation International.

[87] A. Legay, B. Delahaye, and S. Bensalem. Statistical model checking: An overview. In H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. Pace, G. Ro?u, O. Sokolsky, and N. Tillmann, editors, *Runtime Verification*, volume 6418 of *Lecture Notes in Computer Science*, pages 122–135. Springer Berlin Heidelberg, 2010.

[88] S. Leye and A. M. Uhrmacher. Guise - a tool for guiding simulation experiments. In *Proceedings of the Winter Simulation Conference*, pages 305:1–305:2, Berlin, Germany, 2012. Winter Simulation Conference.

[89] J. Li, F. Liying, X. Qing, Z. Shi, and X. Yiliu. Interface generation technology based on concur task tree. In *International Conference on Information Networking and Automation (ICINA)*, volume 2, pages 350–354, Kunming, China, 2010.

[90] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):5–55, 1932.

[91] K. Lyons, H. Brashear, T. Westeyn, J. Kim, and T. Starner. Gart: The gesture and activity recognition toolkit. In J. Jacko, editor, *Human-Computer Interaction. HCI Intelligent Multimodal Interaction Environments*, volume 4552 of *Lecture Notes in Computer Science*, pages 718–727. Springer Berlin Heidelberg, 2007.

[92] T. Maekawa, Y. Yanagisawa, Y. Kishino, K. Ishiguro, K. Kamei, Y. Sakurai, and T. Okadome. Object-based activity recognition with heterogeneous sensors on wrist. In *Proceedings of the 8th International Conference on Pervasive Computing*, Pervasive'10, pages 246–264, Helsinki, Finland, 2010. Springer-Verlag.

[93] P. Maier, D. Jain, and M. Sachenbacher. Compiling AI engineering models for probabilistic inference. In J. Bach and S. Edelkamp, editors, *KI 2011: Advances in Artificial Intelligence*, volume 7006 of *Lecture Notes in Computer Science*, pages 191–203. Springer Berlin Heidelberg, 2011.

[94] F. Mastrogiovanni, A. Scalmato, A. Sgorbissa, and R. Zaccaria. Smart environments and activity recognition: A logic-based approach. In L. Chen, C. D. Nugent, J. Biswas, and J. Hoey, editors, *Activity Recognition in Pervasive Intelligent Environments*, volume 4 of *Atlantis Ambient and Pervasive Intelligence*, pages 83–109. Atlantis Press, 2011.

[95] M. Matessa and J. R. Anderson. Towards an act-r model of communication in problem solving. In *Proceedings of the 1999 AAAI Fall Symposium: Psychological Models of Communication in Collaborative Systems*, 1999.

[96] B. Meyer. *Introduction to the Theory of Programming Languages*. Prentice Hall, 1991.

[97] S. Miksch, Y. Shahar, and P. Johnson. Asbru: A task-specific, intention-based, and time-oriented language for representing skeletal plans. In E. Motta, F. v. Harmelen, C. Pierret-Golbreich, I. Filby, and N. Wijngaards, editors, *Proceedings of the 7th Workshop on Knowledge Engineering: Methods and Languages (KEML-97)*, pages 1–25, Milton Keynes, UK, 1997.

[98] W. T. Morris. On the art of modeling. *Management Science*, 13(12):707–717, August 1967.

[99] J. Muncaster and Y. Ma. Activity recognition using dynamic bayesian networks with automatic state selection. In *Proceedings of the IEEE Workshop on Motion and Video Computing*, WMVC '07, pages 30–38, Washington, DC, USA, 2007. IEEE Computer Society.

[100] N. Munier. *A Strategy for Using Multicriteria Analysis in Decision-Making*. Springer, 2011.

[101] K. P. Murphy. *Dynamic bayesian networks: Representation, inference and learning*. PhD thesis, Computer Science Division, Berkeley, CA, USA, July 2002.

[102] R. Nisbet, J. Elder, and G. Miner. *Handbook of Statistical Analysis and Data Mining*. Elsevier, 2009.

[103] Y. Oh, A. Schmidt, and W. Woo. Designing, developing, and evaluating context-aware systems. In *Proceedings of International Conference on Multimedia and Ubiquitous Engineering, MUE '07*, pages 1158–1163, Seoul, Korea, 2007.

[104] G. Okeyo, L. Chen, H. Wang, and R. Sterritt. Ontology-based learning framework for activity assistance in an adaptive smart home. In L. Chen, C. D. Nugent, J. Biswas, and J. Hoey, editors, *Activity Recognition in Pervasive Intelligent Environments*, volume 4 of *Atlantis Ambient and Pervasive Intelligence*, pages 237–263. Atlantis Press, 2011.

[105] Online Oxford Dictionary of English. Definition of the word ”action”, 2014. Retrieved: 07.03.2014 from http://oxforddictionaries.com/definition/english/action.

[106] Online Oxford Dictionary of English. Definition of the word ”intention”, 2014. Retrieved: 07.03.2014 form http://oxforddictionaries.com/definition/english/intention.

[107] D. L. Parnas and P. C. Clements. A rational design process: How and why to fake it. *IEEE Trans. Softw. Eng.*, 12(2):251–257, Feb. 1986.

[108] F. Paterno. *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, 1999.

[109] D. J. Patterson, L. Liao, K. Gajos, M. Collier, N. Livic, K. Olson, S. Wang, D. Fox, and H. Kautz. Opportunity Knocks: A System to Provide Cognitive Assistance with Transportation Services. In *Proceedings of The International Conference on Ubiquitous Computing, UbiComp 2004*, pages 433–450. Springer, 2004.

[110] J. Pearl. The art and science of cause and effect. *Causality: Models, Reasoning, and Inference*, pages 401–428, 2009. A public lecture delivered November 1996 as part of the UCLA Faculty Research Lectureship Program.

[111] J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, second edition, 2009.

[112] L. M. Pereira and H. T. Anh. Intention recognition via causal bayes networks plus plan generation. In L. Lopes, N. Lau, P. Mariano, and L. M. Rocha, editors, *Progress in Artificial Intelligence*, volume 5816 of *Lecture Notes in Computer Science*, pages 138–149. Springer Berlin Heidelberg, 2009.

[113] D. Powers. Evaluation: from precision, recall, and f-measure to ROC, informedness, markedness, and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, December 2011.

[114] M. Ramirez and H. Geffner. Goal recognition over pomdps: Inferring the intention of a pomdp agent. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, volume 3 of *IJCAI'11*, pages 2009–2014, Barcelona, Spain, 2011. AAAI Press.

[115] S. Richter. *Landmark-Based Heuristics and Search Control for Automated Planning*. PhD thesis, Institute of Intelligent and Integrated Systems, Griffith University, Brisbane, Australia, November 2011.

[116] S. Richter and M. Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39(1):127–177, September 2010.

[117] M. Robards and P. Sunehag. Semi-markov kmeans clustering and activity recognition from body-worn sensors. In *Proceedings of the 9th IEEE International Conference on Data Mining, ICDM '09*, pages 438–446, Miami, USA, 2009.

[118] P. Robinson, M. Hinchey, and K. Clark. Qu-prolog: An implementation language for agents with advanced reasoning capabilities. In M. G. Hinchey, J. L. Rash, W. F. Truszkowski, C. Rouff, and D. Gordon-Spears, editors, *Formal Approaches to Agent-Based Systems*, volume 2699 of *Lecture Notes in Computer Science*, pages 162–172. Springer Berlin Heidelberg, 2003.

[119] P. C. Roy, S. Giroux, B. Bouchard, A. Bouzouane, C. Phua, A. Tolstikov, and J. Biswas. A possibilistic approach for activity recognition in smart homes for cognitive assistance to Alzheimer's patients. In L. Chen, C. D. Nugent, J. Biswas, J. Hoey, and I. Khalil, editors, *Activity Recognition in Pervasive Intelligent Environments*, volume 4 of *Atlantis Ambient and Pervasive Intelligence*, pages 33–58. Atlantis Press, 2011.

[120] W. W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pages 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press. Reprinted in Proceedings of the International Conference on Software Engineering (ICSE) 1989, ACM Press, pp. 328-338.

[121] I. Rus, H. Neu, and J. Münch. A systematic methodology for developing discrete event simulation models of software development processes. In *Proceedings of the 4th International Workshop on Software Process Simulation Modeling (ProSim) at International Conference on Software Engineering*, Portland, Oregon, USA, May 2003.

[122] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach*. Prentice Hall, second edition, 2003.

[123] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach*. Prentice Hall, third edition, 2010.

[124] A. Sadilek and H. Kautz. Assistance in human-machine interaction: a conceptual framework and a proposal for a taxonomy. *Theoretical Issues in Ergonomics Science*, 6(2):129–155, 2005.

[125] F. Sadri. Logic-based approaches to intention recognition. In N.-Y. Chong and F. Mastrogiovanni, editors, *Handbook of Research on Ambient Intelligence: Trends and Perspectives*, pages 346–375. IGI Global, 2011.

[126] S. Sahni. Backtracking. In *Data Structures, Algorithms, and Applications in C++*, pages 751–778. McGraw-Hill, 1998.

[127] P. D. Schreuders. Engineering modeling using a design paradigm: A graphical programming-based example. *Journal of Technology Education*, 19(1):53–70, 2007.

[128] A. Serna, H. Pigot, and V. Rialle. Modeling the progression of alzheimer's disease for cognitive assistance in smart homes. *User Modeling and User-Adapted Interaction*, 17(4):415–438, September 2007.

[129] E. Serral, P. Valderas, and V. Pelechano. A model driven development method for developing context-aware pervasive systems. In F. Sandnes, Y. Zhang, C. Rong, L. Yang, and J. Ma, editors, *Ubiquitous Intelligence and Computing*, volume 5061 of *Lecture Notes in Computer Science*, pages 662–676. Springer Berlin Heidelberg, 2008.

[130] Y. Shahar, S. Miksch, and P. Johnson. The Asgaard project: a task-specific framework for the application and critiquing of time-oriented clinical guidelines. *Artificial Intelligence in Medicine*, 14(1–2):29–51, 1998.

[131] C. Shearer. The crisp-dm model: The new blueprint for data mining. *Journal of Data Warehousing*, 5(4):13–22, 2000.

[132] T. B. Sheridan. *Humans and Automation: System Design and Research Issues*. John Wiley & Sons, 2002.

[133] G. Singla, D. J. Cook, and M. Schmitter-Edgecombe. Incorporating temporal reasoning into activity recognition for smart home residents. In *Proceedings of the AAAI Workshop on Spatial and Temporal Reasoning*, Chicago, Illinois, 2008.

[134] D. E. Smith. The case for durative actions: A commentary on pddl2.1. *Journal of Artificial Intelligence Research*, 20:149–154, 2003.

[135] I. Sommerville. *Software Engineering*. Addison-Wiseley, 1996.

[136] S. Staab, R. Studer, H.-P. Schnurr, and Y. Sure. Knowledge processes and ontologies. *Intelligent Systems, IEEE*, 16(1):26–34, 2001.

[137] G. L. Steele. *Common Lisp the Language*. Digital Press, second edition, 1990.

[138] A. Steinhage and C. Lauterbach. Sensfloor and navifloor: Large-area sensor systems beneath your feet. In N.-Y. Chong and F. Mastrogiovanni, editors, *Handbook of Research on Ambient Intelligence: Trends and Perspectives*, pages 41–55. IGI Global, 2011.

[139] R. J. Sternberg. *Cognitive Psychology*. Wadsworth, 2009.

[140] H. Storf, M. Becker, and M. Riedl. Rule-based activity recognition framework: Challenges, technique and learning. In *Proceeding of the 3rd International Conference on Pervasive Computing Technologies for Healthcare, (PervasiveHealth)*, pages 1–7, London, Great Britain, April 2009.

[141] A. Subramanya, A. Raj, J. Bilmes, and D. Fox. Hierarchical models for activity recognition. In *2006 IEEE 8th Workshop on Multimedia Signal Processing*, pages 233–237. IEEE, 2006.

[142] G. R. Sukthankar. *Activity Recognition for Agent Teams*. PhD thesis, Robotics Institute School of Computer Science Carnegie Mellon University, 2007.

[143] A. S. Tanenbaum and A. S. Woodhull. *Operating Systems Design and Implementation*. Prentice Hall, third edition, 2006.

[144] H. Tonn-Eichstädt. Measuring website usability for visually impaired people-a modified goms analysis. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility*, Assets '06, pages 55–62, New York, NY, USA, 2006. ACM.

[145] F. d. Torre, J. Hodgins, J. Montano, S. Valcarcel, R. Forcada, and J. Macey. Guide to the carnegie mellon university multimodal activity (CMU-MMAC) database. Technical Report CMU-RI-TR-08-22, Robotics Institute, Carnegie Mellon University, July 2009.

[146] J. G. Trafton, L. M. Hiatt, A. M. Harrison, F. P. Tamborello, S. S. Khemlani, and A. C. Schultz. Act-r/e: An embodied cognitive architecture for human-robot interaction. *Journal of Human-Robot Interaction*, 2(1):30–55, 2013.

[147] Ubisense. Real-time location systems (RTLS) and geospatial consulting, 2014. Retrieved: 07.03.2014.

[148] M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *The Knowledge Engineering Review*, 11:93–136, June 1996.

[149] M. Uschold and M. King. Towards a methodology for building ontologies. In *In Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*, pages 6.1–6.10, 1995.

[150] M. Uschold, M. King, S. Moralee, and Y. Zorgios. The enterprise ontology. *The Knowledge Engineering Review*, 13:31–89, March 1998.

[151] T. L. M. van Kasteren. *Activity recognition for health monitoring elderly using temporal probabilistic models*. PhD thesis, University of Amsterdam, 2011.

[152] T. L. M. van Kasteren and B. J. A. Kröse. A sensing and annotation system for recording datasets in multiple homes. In *Proceedings of the 27 Annual Conference on Human Factors and Computing Systems*, pages 4763–4766, Boston, USA, April 2009.

[153] T. L. M. van Kasteren, A. Noulas, G. Englebienne, and B. Kröse. Accurate activity recognition in a home setting. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, UbiComp '08, pages 1–9, New York, NY, USA, 2008. ACM.

[154] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, December 1945.

[155] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.

[156] R. E. Wray, J. E. Laird, A. Nuxoll, D. Stokes, and A. Kerfoot. Synthetic adversaries for urban combat training. In *Proceedings of the 16th Conference on Innovative Applications of Artifical Intelligence*, IAAI'04, pages 923–930, San Jose, California, 2004. AAAI Press.

[157] M. Wurdel. *An Integrated Formal Task Specification Method for Smart Environments*. PhD thesis, University of Rostock, Rostock, Germany, October 2011.

[158] M. Wurdel, C. Burghardt, and P. Forbrig. Supporting ambient environments by extended task models. In M. Mühlhäuser, A. Ferscha, and E. Aitenbichler, editors, *Constructing Ambient Intelligence*, volume 11 of *Communications in Computer and Information Science*, pages 131–138. Springer Berlin Heidelberg, 2008.

[159] M. Wurdel, D. Sinnig, and P. Forbrig. Ctml: Domain and task modeling for collaborative environments. *Journal of Universal Computer Science*, 14(19):3188–3201, 2008.

[160] K. Yordanova. Toward a unified human behaviour modelling approach. Technical Report CS-02-11, Institut für Informatik, Universität Rostock, Rostock, Germany, May 2011. ISSN 0944-5900.

[161] K. Yordanova, F. Krüger, and T. Kirste. Strategies for modelling human behaviour for activity recognition with precondition-effect rules. In B. Glimm and A. Krüger, editors, *KI 2012: Advances in Artificial Intelligence*, volume 7526 of *Lecture Notes in Computer Science*, pages 257–261. Springer Berlin Heidelberg, 2012.

[162] Z. Zeng and Q. Ji. Knowledge based activity recognition with dynamic bayesian network. In *Proceedings of the 11th European conference on Computer vision: Part VI*, ECCV'10, pages 532–546, Crete, Greece, 2010. Springer-Verlag.

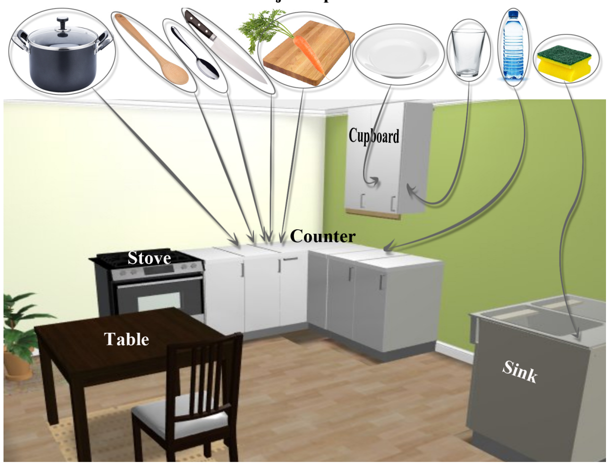
Appendices

Appendix A

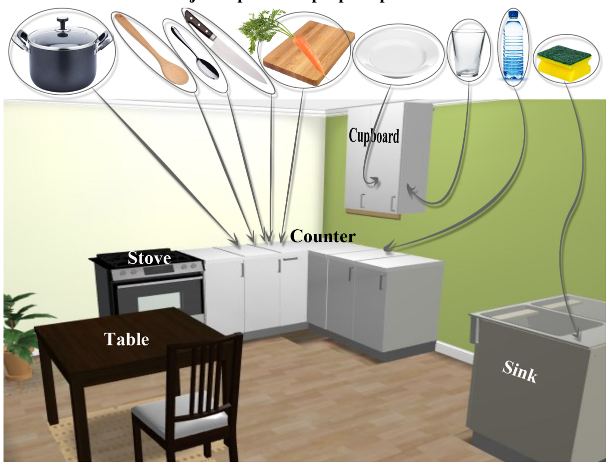
Objects' Locations throughout the Cooking Task Problem

This appendix provides information about the objects' positions throughout the different experiment phases. The figures where there is more than one arrow pointing out of an object, indicates that during this phase, the object could be located at different positions.

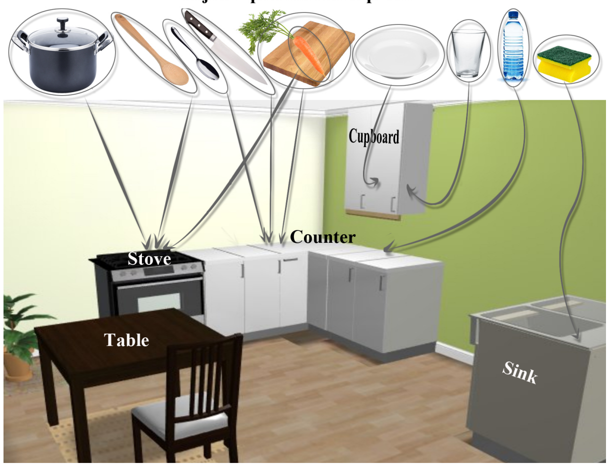
Initial objects' position



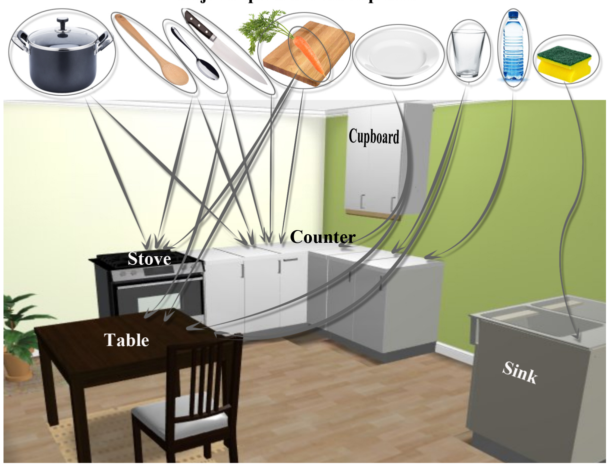
Objects' position: prepare phase



Objects' position: cook phase



Objects' position: serve phase



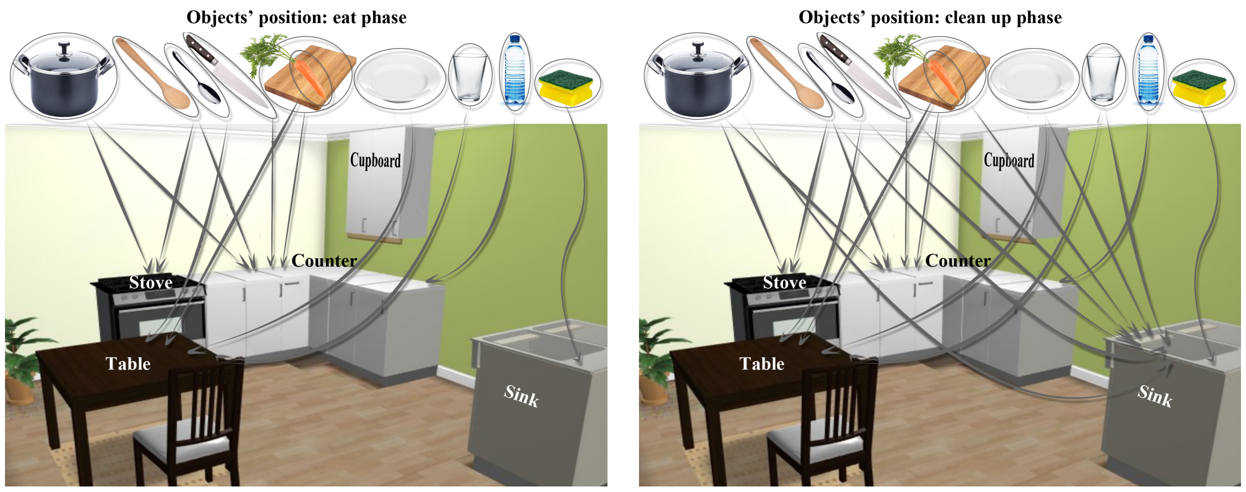

Objects' position: eat phase
Cupboard
Counter
Stove
Table
Sink
Objects' position: clean up phase
Cupboard
Counter
Stove
Table
Sink

Figure A.1: Objects and their locations throughout the different stages of the cooking task. The arrows indicate the places where a given object could be located at a given stage of the task execution.

Appendix B

Introduction to Communicating Sequential Processes

CSP were first introduced by C.A.R. Hoare in 1985 in his book with the same name [67]. In it an object in the world around us acts and interacts with us in terms of events (respectively actions) that build up behaviour patterns called processes. Here we give a short introduction to CSP based on the book *Communicating Sequential Processes* [67].

To express a process, the following conventions are used.

1. Words in lower-case letters denote distinct events, e.g. in the context of activity recognition *move*, *present*, *eat*. The same applies for the letters *a*, *b*, *c*, *d*, *e*.
2. Words with upper-case letters denote specific defined processes, e.g. *COOKING*, *CLEANING*, *MEETING*.
3. Upper-case letters *P*, *Q*, *R* denote arbitrary processes.
4. Lower-case letters *x*, *y*, *z* are variables denoting events.
5. Upper-case letters *A*, *B*, *C* describe sets of events.
6. Upper-case letters *X*, *Y*, *Z* are variables denoting processes.
7. The alphabet of a process *P* is denoted as αP, e.g. $\alpha MEETING = \{enter, present, discuss, leave\}$

The process with alphabet A which never actually engages in any of the events of A is called $STOP_A$. This process describes the behaviour of a broken object.

B.1 Prefix

Let x be an event and let P be a process. Then

$$(x \rightarrow P) \tag{B.1}$$

describes an object which first engages in the event x and then behaves exactly as described by P. The process $(x \rightarrow P)$ has the same alphabet as P so the notation cannot be used unless $x \in \alpha P$

For example, the process (*enter* $\rightarrow$ *MEETING*) is initiated with the action *enter* and then continues with the same behaviour as described in the process *MEETING*.

B.2 Recursion

The prefix notation can be used for expressing the behaviour of an entire process. For example, consider a clock whose sole purpose is to tick. The process that describes this clock will be called *CLOCK* with an alphabet $\alpha \mathit{CLOCK} = \{\mathit{tick}\}$. Now consider a process that has exactly the same behaviour as *CLOCK* but that first emits a single tick. Such process can be described as $(\mathit{tick} \rightarrow \mathit{CLOCK})$. As the behaviour of the second process is indistinguishable from that of the first, we can conclude that $\mathit{CLOCK} = (\mathit{tick} \rightarrow \mathit{CLOCK})$. By simple substitution of equals, we can now extend the equation:

$$\mathit{CLOCK} = (\mathit{tick} \rightarrow \mathit{CLOCK}) = (\mathit{tick} \rightarrow (\mathit{tick} \rightarrow \mathit{CLOCK})) = (\mathit{tick} \rightarrow \mathit{tick} \rightarrow \mathit{CLOCK})$$

This equation can be unfolded as many times as necessary providing a mechanism for repeating behaviour. However, this notation could be long and tedious, especially if the process is repeated indefinitely. For that reason a recursion construct is introduced. Given a process X with an alphabet A, and a prefix with which the process starts $F(X)$, then we express recursion in the following way.

$$\mu X : A.F(X) \tag{B.2}$$

If we return to the clock example, then the notation $\mathit{CLOCK} = \mu X : \{\mathit{tick}\}.(\mathit{tick} \rightarrow X)$ indicates that we have a clock ticking indefinitely.

B.3 Traces

A trace of a behaviour of a process is a finite sequence of events that describes the process's behaviour until a certain moment in time. A trace is expressed as a sequence of symbols, separated by commas and enclosed in angular brackets. Given the events x and y

$$\langle x, y \rangle \tag{B.3}$$

indicates that the trace consists of two events: x followed by y. We can also express the following:

$\langle x \rangle$ is a trace with a single event x.

$\langle \rangle$ is an empty sequence containing no events.

For example the process *MEETING* with an alphabet
$\alpha \mathit{MEETING} = \{\mathit{enter}, \mathit{presentA}, \mathit{presentB}, \mathit{presentC}, \mathit{discuss}, \mathit{leave}\}$ has the traces

$\langle \mathit{enter}, \mathit{presentA}, \mathit{presentB}, \mathit{presentC}, \mathit{discuss}, \mathit{leave} \rangle$

$\langle \mathit{enter}, \mathit{presentA}, \mathit{presentC}, \mathit{presentB}, \mathit{discuss}, \mathit{leave} \rangle$

$\langle \mathit{enter}, \mathit{presentB}, \mathit{presentA}, \mathit{presentC}, \mathit{discuss}, \mathit{leave} \rangle$

...

B.4 Processes

Above we introduced the idea of a process and the events it is composed of. As we are talking about communicating sequential processes, there are of course different relations between these processes. The full set of these relations can be found in Hoare's book *Communicating Sequential Processes* [67]. Here we list just some of them that are important for the definition of the requirements for human behaviour modelling.

B.4.1 Nondeterministic choice

Given processes P and Q, then the notation

$$P \sqcap Q \tag{B.4}$$

denotes a process which behaves either like P or like Q, where the selection is made arbitrarily and the external environment does not have the knowledge or control over this choice.

For example, we have the process *CLEANING-1* $=$ (*take-sponge* $\rightarrow$ *CLEANING*) which starts the process of cleaning by taking the sponge, and then we have the process *CLEANING-2* $=$ (*turn-on-water* $\rightarrow$ *CLEANING*) in which the process starts by turning on the water. Then the notation (*CLEANING-1* $\sqcap$ *CLEANING-2*) indicates that the process can start either by taking the sponge or turning on the water tap but we cannot control which one it will be.

B.4.2 Deterministic choice

Given processes P and Q, then the notation

$$P[]Q \tag{B.5}$$

denotes a process which behaves either like P or like Q where the environment can control which of the two will be selected, provided the control is exercised on the very first action.

In the example with the process *CLEANING* the notation (*CLEANING-1*[]*CLEANING-2*) indicates that either the first or the second process will be executed but which of the two can be controlled by the environment.

B.4.3 Concurrency

Concurrent processes in CSP are expressed by the operator $||$. Given processes P and Q, then the notation

$$P||Q \tag{B.6}$$

denotes that both processes are executed in parallel. The alphabets of this compound process is the union of the alphabets of P and Q, namely $\alpha P \cup \alpha Q$.

For example let's take the process *PRESENTING* and the process *LISTENING*. The notation (*PRESENTING*||*LISTENING*) indicates that the two processes can be executed in parallel. They indicate that while one person is presenting the other is listening.

B.4.4 Interleaving processes

To express processes that have interleaving actions, CSP uses the operator $|||$. Given two processes P and Q, then the notation

$$P|||Q \tag{B.7}$$

indicates that the actions in the two processes are interleaved so that each action in the compound process belongs either to P or to Q, or when to both, then the choice of which process will execute the action is nondeterministic.

For example, let's take the processes *EATING* and *DRINKING* where α*EATING* $=$ {*take-spoon*, *eat*, *put-spoon*} and α*DRINKING* $=$ {*take-glass*, *drink*, *put-glass*}. Then the notation (*EATING*|||*DRINKING*) indicates that the actions of the two processes are interlaced with each other and any of the actions of the first can be executed in between actions of the second process.

B.4.5 Sequential composite processes

To express a composition of sequential processes one uses the operator ;. Given the processes P and Q, then the notation

$$P;Q \tag{B.8}$$

indicates that the process was first behaving like P but when it was successfully executed then it started behaving like Q. If the process P never terminates successfully, neither does the compound process $P;Q$.

For example, if we take the processes *EATING* and *DRINKING* and we assume that one can drink only after successfully finishing the meal, then we denote this relation as (*EATING*;*DRINKING*).

B.4.6 Interruption

To express a composition where the first process is interrupted by another process, the operator $^\wedge$ is used. Given processes P and Q, then the notation

$$P^\wedge Q \tag{B.9}$$

indicates that the process P was in progress when it was interrupted by the first occurrence of an event from process Q.

For example let's take the processes *EATING* and *PHONE-RINGING*. Then the notation (*EATING*$^\wedge$*PHONE-RINGING*) indicates that the process of eating was interrupted after the first event of the ringing phone.

B.4.7 Alternation

It is possible that we want to resume our process after it was interrupted from the state it was in before being interrupted. To denote that the operator $\otimes$ is used. Given processes P and Q, then the notation

$$P \otimes Q \tag{B.10}$$

indicates that the process P was interrupted by Q but later resumed from the state in which it was interrupted.

For example, (*EATING* $\otimes$ *PHONE-RINGING*) indicates that the eating was interrupted by the ringing phone but that later the person continued eating from where she was interrupted.

Appendix C

Questionnaire for Requirements Validation

This appendix gives an example for the questions used throughout the requirements analysis questionnaire, as well as the Likert scale used for each requirement feature.

C.1 Likert scales

The Likert scale is a scale usually involved in problems where questionnaires are used. It was invented by Rensis Likert in 1932 and basically proposes a format in which responses are scored along a range [90]. A typical Likert scale consists of a number of Likert items, where the scale represents the sum of responses from several items. The item, on the other hand, consists of a question and the possible responses of this questions. The data obtained from the Likert scales can be considered as ordinal or interval depending on whether one makes the assumption that the response options are equidistant. There is a lot of discussion whether the data can be accepted as ratio [20]. In our case the data is assumed to be ordinal and the median is used as a measure for describing the answers distribution. There is no fixed number of responses per item, however according to recent research a scale with 5 or 7 points will result in a slightly higher mean than a 10 point scale [32]. In the questionnaire presented below, a 5-point scale was used and the attempt was made to design the responses so that they are as equidistant as possible.

C.2 Example questionnaire

In the questionnaire below, first user related questions are asked to determine her experience in the field of modelling and activity recognition. Later, for each requirement the following properties are evaluated: accuracy, validity, clarity, completeness, feasibility, testability, traceability, and importance. Additionally, the requirement's consistence with the rest of the requirements is evaluated. The results from the study are shown in Appendix C.3.

What is your age?

....

What kind of academic degree do you have?

☐ Abitur ☐ Bachelor ☐ Master ☐ Diplom ☐ Doctor ☐ Professor ☐ Other:...................

Have you ever modelled human (user) behaviour with any kind of modelling formalism?

☐ yes, I often build models

☐ yes, I built several models

☐ yes, I built a complex model once

☐ yes, I built a simple model once

☐ no, I have never modelled human behaviour before

In which field of computer science do you work?

☐ activity recognition

☐ modelling and simulation

☐ visual computing

☐ software engineering

☐ student in computer science

☐ other:..........................

If relevant, with what kind of modelling formalisms have you worked before?

....

....

....

List two modelling formalisms with which you think the 3 problems could be modelled?

....

....

....

Requirement's Name

specification: *requirement's specification*

Give an example from one of the three problems (3-person meeting, kitchen task assessment, office scenario) that describes behaviour possessing this requirement.

....

....

....

Is the requirement verifiable? How accurately does the requirement's specification represent the concrete example you gave?

□ very accurately □ accurately □ not very accurately □ inaccurately □ very inaccurately

Is the requirement valid? How accurately does the requirement's specification represent the actual requirement needed in the example given by you? (Is that really what you need?)

□ very accurately □ accurately □ not very accurately □ inaccurately □ very inaccurately

Is the requirement clear? To what degree is the requirement unambiguous and understandable?

□ very clear □ clear □ relatively clear □ unclear □ very unclear

Is the requirement complete? Is there some missing information in the requirement's specification?

□ nothing missing □ almost nothing □ some □ most missing □ everything missing

If relevant, what information is missing in the requirement's specification?

....

....

....

Is the requirement feasible? How easy can the requirement be implemented without contradicting competing requirements?

☐ very easy ☐ easy ☐ moderate ☐ difficult ☐ very difficult

Is the requirement testable? Do you think the implementation of the requirement will be possible based on its specification?

☐ highly likely ☐ likely ☐ plausible ☐ unlikely ☐ highly unlikely

Is the requirement traceable? Can you easily identify requirements related to this requirement?

☐ very easy ☐ easy ☐ moderate ☐ difficult ☐ very difficult

How important is this requirement for solving the three modelling problems?

☐ very important ☐ important ☐ not important ☐ optional ☐ irrelevant

For each requirement is the requirement consistent? To what degree the requirement is specified using uniform notation, terminology, and symbology compared to the other requirements?

Requirement	very consistent	consistent	relatively consistent	inconsistent	very inconsistent
sequences					
composition					
parallelism					
interleaving activities					
repetition					
choice					
enabling					
disabling					
priority					
independence					
dependence					
synchronisation					
suspend					
resume					
probabilistic durations					
observation model					
unobserved actions					
activity recognition					

The list with requirements above should contain all requirements needed to describe the problems. Do you think that the list of requirements is complete?

....

....

If not, which requirements are missing?

C.3 Evaluation results for the requirements' features

This appendix shows the results for the features evaluation based on the questionnaire above and discussed in Chapter 2.

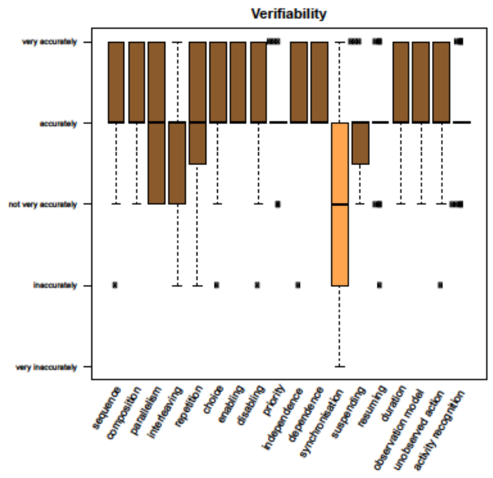


Figure C.1: Scores for requirements verifiability.

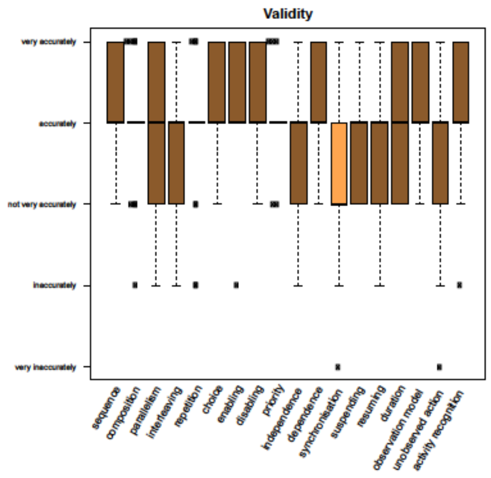


Figure C.2: Scores for requirements validity.

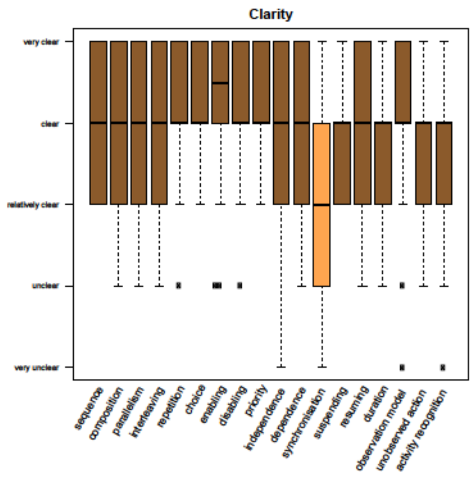


Figure C.3: Scores for requirements clarity.

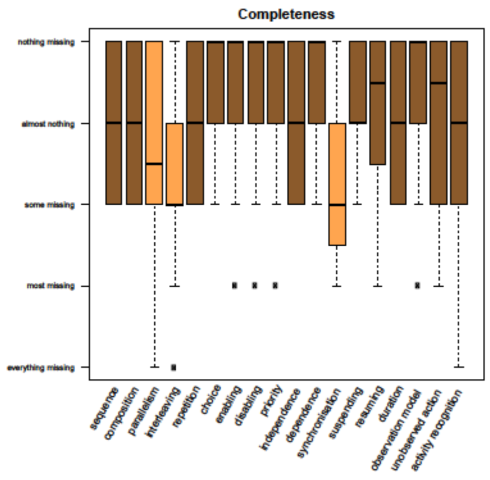


Figure C.4: Scores for requirements completeness.

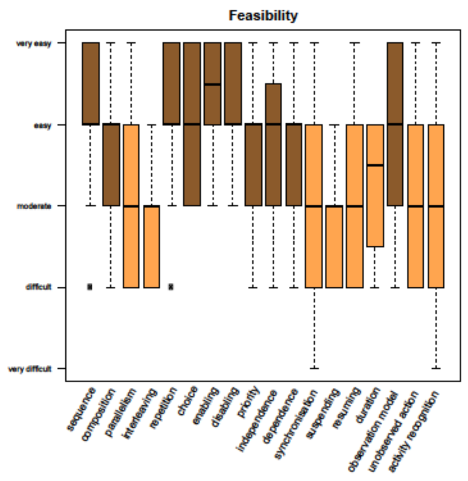


Figure C.5: Scores for requirements feasibility.

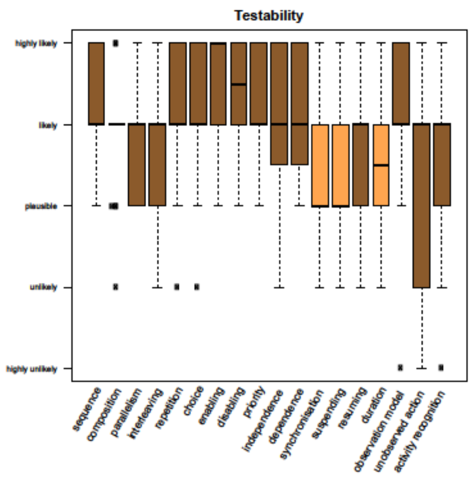


Figure C.6: Scores for requirements testability.

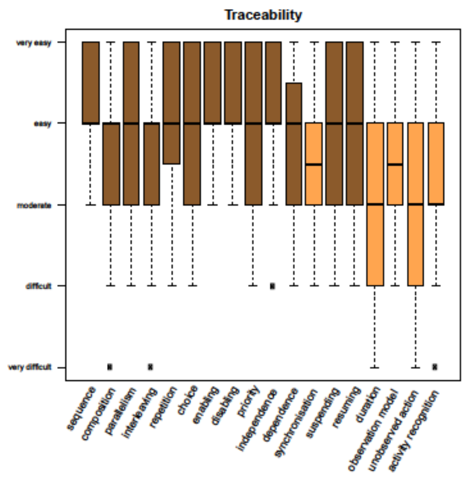


Figure C.7: Scores for requirements traceability.

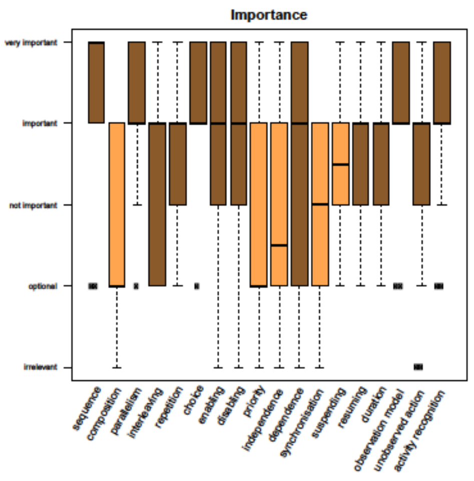


Figure C.8: Scores for requirements importance.

Appendix D

Introduction to Modelling with Computational Causal Behaviour Models

As it was already mentioned in Chapter 2, CCBM is divided into three parts – a causal model that consists of domain description and problem description, and an observation model that provides the connection between the high-level causal model and the observations. Below we discuss the model parts in detail and later give a short introduction into modelling with CCBM. The information in this section is based on the technical report ”CCBM-A tool for activity recognition using Computational Causal Behavior Models” [76].

Modelling with CCBM is relatively straight forward and consists of two parts – the first is creating causal models, while the second is defining observation models that contain the link between the causal high-level actions and the underlying observations. Although this work does not concern itself with the development of the observation models, here we briefly explain how they are created.

D.1 Causal modelling

As the probabilistic models are automatically generated from the causal model, the modelling load is concentrated in the building of the causal model. For simplicity, the causal model itself is divided into two separate files – the domain and the problem file. The first contains the abstract definition of the operators (or action templates) in terms of preconditions and effects as well as the observation clause that links the described actions to the functions in the observation model. The second file then contains all problem specific elements, such as the initial state of the world, the goal state, probabilistic distribution of the actions' durations, actions' saliencies etc.

D.1.1 Domain definition

The domain definition consists of five entries that describe the domain of the modelled problem. These are the types definition, the predicates definition, the constants definition, the action templates and the observation clause (see Fig. D.1).

The type definition describes all object types that are used in the domain as well as their subtypes. To illustrate the CCBM syntax, we take a dummy example from the meeting domain. In Fig. D.2 for the meeting problem there are five types defined, two of which are subtypes (type

```
domain = (define (domain name){domain-entry})

domain-entry  =
        (:types {name{name} - name}{name})
        | (:predicates  {(name[declarations])})
        | (:constants {constant{constant} - type-name}{constant})
        | (:action {action-formula})
        | (:observation {observation-formula})

type-name      = name
declarations  = {{var} - type-name}{var}
```

Figure D.1: The domain description consists of domain name, types declaration, predicates declaration, constants declaration, actions declaration, and observations declaration. Figure adapted from Kirste et al. [76].

seat and *stage* are subtypes of *location*). All types that do not have explicitly defined supertype belong to the general type *object*.

Furthermore, Fig. D.3 shows an excerpt of the predicates specification for the meeting use case where three predicates are defined. Here the first element is the name of the predicate whereas the remaining elements are the variables that are required for the given predicate and their type.

```
(:types
      seat stage - location
      person
      activity
)
```

Figure D.2: The types definition for a meeting scenario.

```
(:predicates
      (at ?p - person ?l - location)
      (has-presented ?p - person)
      (seated ?p - person)
)
```

Figure D.3: The predicates definition for a meeting scenario.

The next element in the domain definition is the action formula which is an abstract representation of an action that later is to be grounded with the objects defined in the problem description. Fig. D.4 shows the action elements – it consists of a name, a declaration of the parameters to be assigned to the action after grounding, a saliency value which gives the weight an action has with respect to the rest of the actions. The next element is the agent definition

```
action = (:action name
        [(:parameters (declarations))]
        [(:saliency expression)]
        [(:agent atomic-expression)]
        [(:irrational)]
        [(:duration atomic-formula)]
        [(:precondition formula)]
        [(:effect effect-formula)]
        [(:callbacks callback-list)] )
```

Figure D.4: The action template elements. Square brackets indicate that the entry is optional. An action consists of parameters, saliency, agent description, irrational behaviour definition, duration, preconditions, effects, and callbacks that connect the model states to the observation model. Figure adapted from Kirste et al. [76].

which, when present, implies that the action is used in multi-agent mode; the irrational clause defines that the action is not to be weeded out even if it is not causally needed for achieving the goal; the duration clause describes the duration that is assigned to the action – it could be either a probabilistic distribution or an exact duration. Finally, the precondition and effect clauses define the state of the world that has to be true in order the action to be executable, and the state of the world after the action is executed. The last element of the action definition is the callbacks which allows mapping callbacks to actions, so at each time step the callback of the action for each agent will be invoked.

```
(:action sit-down
   :parameters (?p - person ?s - seat)
   :duration (exact (sit-duration ?p))
   :precondition (and
        (at ?p ?s)
        (not (seated ?p))
        (empty ?s)
        )
   :effect (and
        (not (empty ?s))
        (seated ?p)
      )
)
```

Figure D.5: Action *sit-down* in CCBM notation.

Fig. D.5 shows an example of an action where a durative action is defined. The duration has an exact value which is later assigned in the problem description. The preconditions and effects in the action template describe the action *sit down* that can be executed only when the person is at the given seat and the seat is empty, while the effect of this action will be that the person is seated and the seat is no longer empty. Additionally, the action has two parameters, namely *?p* of type *person* and *?s* of type *seat* indicating that it can be parameterised only with constants of these types.

The last element from the domain description is the observation clause which gives the relation between the model's actions and states and the functions in the observation model. Fig. D.6 shows an example of the observation clause for observing a person *?p* at seat *?s*. In it, whenever the predicate *(at ?p ?s)* is true, the location of the person in the observation model is

```
(:observation
  (forall (?p - person ?s - seat)
        (when (at ?p ?s)
            (senseLocation ?p (x-coord ?s)
                        (y-coord ?s)))))
```

Figure D.6: The observation clause for observing the person is seated.

set to this location with the function *(senseLocation ?p (x-coord ?s)(y-coord ?s))*.

D.1.2 Problem definition

Having defined the abstract actions in the modelled problem in the same manner as in Fig. D.5, the domain can now be parameterised. This is done by defining a set of objects that will take the places of the parameters with the corresponding type, a description of the initial world state, and the goals that are to be achieved. For that, a separate problem description is defined which has the structure shown in Fig. D.7, where *:domain name* is the name of the corresponding domain specification; *:objects* are the constants used to replace the variables in the domain specification; *:init* defines the initial state of the world, and the specified *:duration* provides the information about how long the initial state should last. Finally, the *:goal* clause defines the goal that has to be achieved in the given problem.

Fig. D.8 shows an example of a problem description for a meeting domain, where the problem name is *meeting* and the corresponding domain is also the *meeting* domain that contains the

```
problem = (define (problem name){problem-entry})

problem-entry =
      (:domain name)
      | (:objects {constant{constant} - type-name}{constant})
      | (:init [:duration atomic-formula]{init-elem})
      | (:goal formula)
```

Figure D.7: Problem definition and its elements. It consists of problem name, the corresponding domain name, the objects to populate the action templates, the initial state of the world, and the goal state. Figure adapted from Kirste et al. [76].

predicates and operators (actions) specifications. The objects in this case are the three users: alice, bob and charlie; and three seats: a, b and c. In the initial world state charlie and bob are not seated and two of the seats are empty. Additionally, the initial world state contains information about the durations of the abstract actions defined in the domain description. For example, the duration of action *sit-down* for agent *alice* has exact value of 10. Furthermore, the

```
(define (problem meeting)
    (:domain meeting)

    (:objects alice bob charlie- person
      a b c - seat)

    (:init
      :duration (exponential 0.1)

      (not (seated bob))
      (not (seated charlie))(empty a)
      (empty b)(seated alice))
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;;;;;;;location assignment ;;;;;;;;;
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      (= (loc-id seatA) 0)
      (= (loc-id seatB) 1)
      (= (loc-id seatC) 2)
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;;;;;;;duration assignment;;;;;;;;;
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      (= (sit-duration alice) 10)
      (= (sit-duration bob) 8)
      (= (sit-duration charlie) 20)


    (:goal (and
      (seated bob)
      (seated charlie)))
```

Figure D.8: Problem definition in CCBM notation

different locations are assigned specific values that are later passed to the observation model and are used during the estimation process. Finally, the goal that has to be achieved by the users is that both charlie and bob are seated.

D.2 Creating observation models

The last element needed for generating probabilistic models is the observation model. An observation model gives the relation $P(y_t \,|\, x)$, or with other words the probability of sighting an observation y at time t given the system model state x. The handling of observations is provided in a C++ code in the form of a header file that is later used by the CCBM tool to compile the models.

The observation model should provide the following functions.

`void *fetchObservation(void);` This function reads the observation y_t from the standard input, stores it wherever it is designated to store, and returns `NULL` when the end of the observation sequence is reached.

`double observe(StatePtr x);` The function provides the probability of seeing the last observation read by the `fetchObservation` given the state x, or with other words the value $p(Y_t = y_t \,|\, X_t = x)$. Here `x` is a pointer to a bit vector representing the current state. Furthermore, in the system model the `:observation` declaration can be used to call specific C-functions for a given state configuration. Then the execution of these helper functions is initiated by calling `stateObservation(x)` at the beginning of `observe`.

For example, if we take the following observation definition.

```
(:observation (forall (?p - person ?l - location)
                   (when (at ?p ?l)(setLocationSensor ?p ?l))))
```

Then the code generated by the compiler will assume that there is a C++ function `void setLocationSensor(Person p, Location l)` with the corresponding types Person and Location that also have to be defined in the observation file. Then the generator will insert the respective constant names in the call. This means that if there is a location `seat`, and a person `alice`, one of the generated calls will be `setLocationSensor(alice, seat).`

In a similar manner, the functions representing the actions or parameters to be estimates can be implemented.

Appendix E

Implementation of Patterns with Computational Causal Behaviour Models

The appendix contains sample implementation of the patterns discussed in Chapter 4. This is also the code with which the dummy models in the same chapter were created.

```
;;;; domain description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain repeat)
  (:predicates
    (allowed-repeat)
    (repeating)
    (repeated))

  (:action repeat
    :parameters ()
    :precondition (and
      (allowed-repeat)
      )
    :effect (and
      (repeating)))

  (:action repeat-finish
    :parameters ()
    :precondition (repeating)
    :effect (and
      (repeated)
      (not (allowed-repeat)))))

;;;; problem description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (problem repeat)
  (:domain repeat)
  (:init (allowed-repeat))

  (:goal (repeated))
)
```

Figure E.1: Implementation of an implicit repeating action.

```
;;;; domain description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(define (domain repeat)

  (:types
    counter
  )

  (:predicates
    (counting ?i ?j - counter)
  )

  (:constants
    0 1 2 - counter
  )

  (:action repeat
    :parameters (?i ?j - counter)
    :precondition (and
      (counting ?i ?j)
      (or
        (and (= ?i 0) (= ?j 0))
        (and (= ?i 0) (= ?j 1))
        (and (= ?i 1) (= ?j 0))
        (and (= ?i 1) (= ?j 1))
        (and (= ?i 1) (= ?j 2))
      )
    )
    :effect (and
      (when (counting 0 0)(and (not (counting 0 0))(counting 0 1)))
      (when (counting 0 1)(and (not (counting 0 1))(counting 1 0)))
      (when (counting 1 0)(and (not (counting 1 0))(counting 1 1)))
      (when (counting 1 1)(and (not (counting 1 1))(counting 1 2)))
      (when (counting 1 2)(and (not (counting 1 2))(counting 2 0)))
    )
  )
)

;;;; problem description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (problem repeat)
  (:domain repeat)
  (:init
    (counting 0 0)
  )

  (:goal
    (counting 2 0)
  )
)
```

Figure E.2: Implementation of an explicit repeating action.

```
;;;; domain description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(define (domain abstract)

  (:types
    type1 type2 type3 - object
  )

  (:predicates
    (manipulated ?o - object)

  )

  (:constants
    object1 object2 - type1
    object3 object4 - type2
    object5 - type3
  )

  (:action manipulate
    :parameters (?i - object)
    :precondition (and
      (not (manipulated ?i))
    )
    :effect (and
      (manipulated ?i)
    )
  )


)

;;;; problem description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (problem abstract)
  (:domain abstract)
  (:init

  )

  (:goal (and
    (manipulated object1)
    (manipulated object2)
    (manipulated object3)
    (manipulated object4)
    (manipulated object5)
    )
  )
)
```

Figure E.3: Implementation of an abstract action.

```
;;;; domain description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain nonabstract)

  (:types type1 type2 type3 - object)

  (:predicates
    (manipulated1 ?o - type1)
    (manipulated2 ?o - type2)
    (manipulated3 ?o - type3))

  (:constants
    object1 object2 - type1
    object3 object4 - type2
    object5 - type3)

  (:action manipulate-type1
    :parameters (?i - type1)
    :precondition (not (manipulated1 ?i))
    :effect (manipulated1 ?i)
  )

  (:action manipulate-type2
    :parameters (?i - type2)
    :precondition (not (manipulated2 ?i))
    :effect (manipulated2 ?i)
  )

  (:action manipulate-type3
    :parameters (?i - type3)
    :precondition (not (manipulated3 ?i))
    :effect (manipulated3 ?i)
  )
)

;;;; problem description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (problem nonabstract)
  (:domain nonabstract)
  (:init)

  (:goal (and
    (manipulated1 object1)
    (manipulated1 object2)
    (manipulated2 object3)
    (manipulated2 object4)
    (manipulated3 object5)
    )
  )
)
```

Figure E.4: Implementation of a specialised action.

```
;;;; domain description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain macro)

  (:types
     actionNumber)

  (:predicates
    (doing-something ?i - actionNumber)
    (did-something ?i - actionNumber)
  )

  (:constants
    1 2 3 - actionNumber)

  (:action begin-do-something
    :parameters (?i - actionNumber)
    :precondition (and
      (not (doing-something ?i))
      (not (did-something ?i))
      (imply (= ?i 1) (and (not (doing-something 2))(not (doing-something 3))(not (
          did-something 2))(not (did-something 3))))
      (imply (= ?i 2) (and (doing-something 1)(not (did-something 3))(not (doing-something 3))
          ))
      (imply (= ?i 3) (and (doing-something 1)(doing-something 2)))
    )
    :effect (doing-something ?i)
  )

  (:action end-do-something
    :parameters (?i - actionNumber)
    :precondition (and
      (doing-something ?i)
      (imply (= ?i 1) (and (did-something 2)(did-something 3)))
      (imply (= ?i 2) (and (doing-something 1)(doing-something 3)))
      (imply (= ?i 3) (and (doing-something 1)(did-something 2)))
    )
    :effect (and
      (did-something ?i)
      (not (doing-something ?i))))
)

;;;; problem description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (problem macro)
  (:domain macro)
  (:init)

  (:goal (and
    (did-something 1)
    (did-something 2)
    (did-something 3)))
)
```

Figure E.5: Implementation of an explicit macro structure.

```
;;;; domain description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain macro)

  (:types
     actionNumber
  )

  (:predicates
    (doing-something ?i - actionNumber)
    (did-something ?i - actionNumber)
  )

  (:constants
    1 2 3 - actionNumber
  )

  (:action begin-do-something
    :parameters (?i - actionNumber)
    :precondition (and
      (not (doing-something ?i))
      (not (did-something ?i))
    )
    :effect (and
      (doing-something ?i)
    )
  )

  (:action end-do-something
    :parameters (?i - actionNumber)
    :precondition (and
      (doing-something ?i)
    )
    :effect (and
      (did-something ?i)
      (not (doing-something ?i))
    )
  )

)

;;;; problem description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (problem macro)
  (:domain macro)
  (:init)

  (:goal (and
    (did-something 1)
    (did-something 2)
    (did-something 3)
    )
  )
)
```

Figure E.6: Implementation of an implicit macro structure.

```
;;;; domain description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(define (domain synchronise)

  (:types person)

  (:predicates
    (doing-something ?p - person)
    (can-do-something ?p - person)
    (doing-something-sync ?p - person)
    (synchronised))

  (:constants person1 person2 - person)

  (:action do-something
    :parameters (?p - person)
    :precondition (and
      (can-do-something ?p)
      (not (doing-something ?p))
    )
    :effect (and
      (doing-something ?p))
  )

  (:action do-something-sync
    :parameters (?p - person)
    :precondition (and
      (forall (?p - person)
        (imply (not (synchronised))(doing-something ?p)))
        (not (doing-something-sync ?p))
    )
    :effect (and
      (forall (?p - person)(and
          (not (doing-something ?p))
          (not (can-do-something ?p))
          (synchronised)
          ))
      (doing-something-sync ?p)
    )
  )
)

;;;; problem description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (problem synchronise)
  (:domain synchronise)
  (:init
    (can-do-something person1)
    (can-do-something person2))

  (:goal (and
    (doing-something-sync person1)
    (doing-something-sync person2))
)
```

Figure E.7: Implementation of synchronisation of team behaviour.

```
;;;; domain description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain synchronise)

  (:types person)

  (:predicates
    (doing-something ?p - person)
    (can-do-something ?p - person)
    (doing-something-sync ?p - person)
    (synchronised)
  )

  (:constants person1 person2 - person)

  (:action begin-do-something
    :parameters (?p - person)
    :precondition (and
      (not (doing-something ?p))
      (can-do-something ?p))
    :effect (and
      (doing-something ?p)))

  (:action end-do-something
    :parameters (?p - person)
    :precondition (and
      (doing-something ?p))
    :effect (and
      (did-something ?p)
      (not (doing-something ?p))
      (not (can-do-something ?p))))

  (:action do-something-sync
    :parameters (?p - person)
    :precondition (and
      (forall (?u - person) (did-something ?u))
      (not (doing-something-sync ?p)))
    :effect (and
      (doing-something-sync ?p)))
)

;;;; problem description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (problem synchronise)
  (:domain synchronise)
  (:init
    (can-do-something person1)
    (can-do-something person2))

  (:goal (and
    (doing-something-sync person1)
    (doing-something-sync person2)))
)
```

Figure E.8: Implementation of synchronisation of multi-agent behaviour.

```
;;;; domain description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain hierarchy)

  (:types
    type1 type2 type3 - object
  )

  (:predicates
    (manipulated ?o - object)

  )

  (:constants
    object1 object2 object3 object4 object5 - type1
    object6 object7 - type2
    object8 - type3
  )

  (:action manipulate
    :parameters (?i - type1)
    :precondition (and
      (not (manipulated ?i))
    )
    :effect (and
      (manipulated ?i)
    )
  )
)

;;;; problem description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (problem hierarchy)
  (:domain hierarchy)
  (:init)

  (:goal (and
    (manipulated object1)
    (manipulated object2)
    (manipulated object3)
    (manipulated object4)
    (manipulated object5)
    )
  )
)
```

Figure E.9: Implementation of hierarchical type system.

```
;;;; domain description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain combined)

  (:types twoObjects - object)

  (:predicates
    (manipulated ?o - object)
    (combined-object1 ?o - two-objects)
    (combined-object2 ?o - two-objects)
  )

  (:constants
    object1 object2 object3 object4 object5 object6 object7 object8 object9 object10 - object
    object1&2 object3&4 object5&6 object7&8 object9&10 - two-objects
  )

  (:action manipulate-two-objects
    :parameters (?i - two-objects)
    :precondition (and
      (not (manipulated (combined-object1 ?i)))
      (not (manipulated (combined-object2 ?i)))
    )
    :effect (and
      (manipulated (combined-object1 ?i))
      (manipulated (combined-object2 ?i))
    )
  )
)

;;;; problem description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (problem combined)
  (:domain combined)
  (:init
    (= (combined-object1 object1&2) object1)
    (= (combined-object2 object1&2) object2)
    (= (combined-object1 object3&4) object3)
    (= (combined-object2 object3&4) object4)
    (= (combined-object1 object5&6) object5)
    (= (combined-object2 object5&6) object6)
    (= (combined-object1 object7&8) object7)
    (= (combined-object2 object7&8) object8)
    (= (combined-object1 object9&10) object9)
    (= (combined-object2 object9&10) object10))

  (:goal (and
    (manipulated object1)(manipulated object2)
    (manipulated object3)(manipulated object4)
    (manipulated object5)(manipulated object6)
    (manipulated object7)(manipulated object8)
    (manipulated object9)(manipulated object10)))
)
```

Figure E.10: Implementation of combining objects with combined objects.

```
;;;; domain description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain combined)

  (:types)

  (:predicates
    (manipulated ?o - object)
    (can-be-together ?o1 ?o2 - object)
  )

  (:constants
    object1 object2 object3 object4 object5 object6 object7 object8 object9 object10 - object
  )

  (:action manipulate-two-objects
    :parameters (?i1 ?i2 - object)
    :precondition (and
      (not (manipulated ?i1))
      (not (manipulated ?i2))
      (can-be-together ?i1 ?i2)
    )
    :effect (and
      (manipulated ?i1)
      (manipulated ?i2)
    )
  ))

;;;; problem description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (problem combined)
  (:domain combined)
  (:init
  (can-be-together object1 object2)
  (can-be-together object3 object4)
  (can-be-together object5 object6)
  (can-be-together object7 object8)
  (can-be-together object9 object10)
  )

  (:goal (and
    (manipulated object1)
    (manipulated object2)
    (manipulated object3)
    (manipulated object4)
    (manipulated object5)
    (manipulated object6)
    (manipulated object7)
    (manipulated object8)
    (manipulated object9)
    (manipulated object10)
    )
  ))
```

Figure E.11: Implementation of combining objects with lock predicates.

```
;;;; domain description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain phases)

  (:types action-number)

  (:predicates
    (doing-something ?i - action-number)
)

  (:constants 1 2 3 4 5 6 - actionNumber)

  (:action do-something
    :parameters (?i - actionNumber)
    :precondition (and
      (not (doing-something ?i))
      (can-do-something ?i)

    )
    :effect (and
      (doing-something ?i)
      (not (can-do-something ?i))
      (when (and
            (not (can-do-something 1))
            (not (can-do-something 2))
            (= ?i 3)
          )
          (and
            (can-do-something 4)
            (can-do-something 5)
            (can-do-something 6)
          )
      )
    )
  )
)

;;;; problem description ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (problem phases)
  (:domain phases)
  (:init
    (can-do-something 1)
    (can-do-something 2)
    (can-do-something 3)
  )

  (:goal (and
    (doing-something 1)(doing-something 2)
    (doing-something 3)(doing-something 4)
    (doing-something 5)
    )
  )
)
```

Figure E.12: Implementation of phases.

Appendix F

Introduction to Combinatorics

Here we provide an introduction into combinatorics on which the model complexity discussed in Chapter 4 are calculated. The information in this appendix are based on the book *Introductory Combinatorics* by Richard Brualdi [17].

F.1 Basic counting principles

Let S be a set. Then a partition of S will be a collection $S_1, S_2, ..., S_n$ of subsets of S such that each element of S is in exactly one of those subsets:

$$S = S_1 \cup S_2 \cup ... \cup S_3, \tag{F.1}$$

and where any two subsets are disjoint

$$S_i \cap S_j = \emptyset. \tag{F.2}$$

Furthermore if we denote the number of elements in S as $|S|$, we say that $|S|$ is the size of S. To calculate the size of S based on the sizes of its partitions, we use the following definition provided in [17, p. 28].

Definition 30. ***(Addition principle)*** *Suppose that a set S is partitioned into pairwise disjoint parts S_1,S_2, ... ,S_n. The number of elements in S can be determined by finding the number of elements in each of the parts, and adding the numbers together to obtain:*

$$|S| = |S_1| + |S_2| + ... + |S_n|. \tag{F.3}$$

The second important principle is the multiplication principle which is used for calculating the number of combinations in sequentially ordered sets.

Definition 31. ***(Multiplication principle)*** *Let S be a set of ordered pairs (a,b) of objects, where the first element a comes from a set of size p, and for each choice of element a there are q choices for element b. Then the size of S is:*

$$|S| = p * q \tag{F.4}$$

F.2 Types of counting problems

According to Brualdi many of the counting problems can be divided into the following categories [17, p. 32].

1. Count the number of ordered arrangements or ordered selections of objects (called *permutation*)
   - without repeating any object,
   - with repetition of the permitted objects.
2. Count the number of unordered arrangements or unordered selections of objects (called *combination*)
   - without repeating any object,
   - with repetition of the permitted objects.

F.3 Permutations of sets

Let r be a positive integer. Then the *r-permutation* of a set S with n elements is an ordered arrangement of r of the n elements.

For example if we have the set $S = \{a,b,c\}$, then the three *1-permutations* of S will be

$$a \qquad b \qquad c,$$

the six *2-permutations* of S will be

$$ab \qquad ac \qquad ba \qquad be \qquad ca \qquad cb,$$

and the six *3-permutations* of S will be

$$abc \qquad acb \qquad bac \qquad bca \qquad cab \qquad cba.$$

Such permutations are denoted as $P(n,r)$ and the number of r-permutations can be computed according to the following formula.

$$P(n,r) = \frac{n!}{(n-r)!} \tag{F.5}$$

Here $n!$ denotes the factorial of a nonnegative integer n which is the product of all positive integers less than or equal to n,

$$n! = n(n-1)(n-2)...(n-(n-2))(1) \tag{F.6}$$

F.4 Combinations of sets

Let S be a set with n elements. A *combination* of S denotes an unordered selection of elements of S. The result of such selection is a subset A of the elements of S: $A \subset S$. Now let r be a nonnegative integer. Then the *r-combination* (or the *r-subset*) of a set S of n elements is an unordered selection of r out of n elements in the set.

For example if we all to obtain all *3-subsets* of the set $S = \{a,b,c,d\}$, this will result in

$$\{a,b,c\},\{a,b,d\},\{a,c,d\},\{b,c,d\}$$

We denote the number of $r - subsets$ of a set with n elements with $\binom{n}{r}$ and calculate it according to the following formula.

$$\binom{n}{r} = \frac{n!}{r!(n-r)!} \tag{F.7}$$

The number of all possible subsets of a given set S together with the empty set is called a *power set* and is denoted with $P(S)$. It computed by

$$P(S) = 2^{|S|} \tag{F.8}$$

For example, if S has n elements, then the number of subsets of S will be 2^n.

F.5 Permutations of multisets

A multiset is a set in which not all of the elements are distinct. For example, the set $S = \{a,a,b,c,c,c\}$ is a multiset where there are 2 repeating a elements and 3 repeating c elements. We can calculate the permutations of such set involving k different types of elements each type with a finite repetition number according to the definition provided in [17, p. 46]

Definition 32. *Let S be a multiset with objects of k different types with finite repetition numbers $n_1, n_2, ..., n_k$, respectively. Let the size of S be $n = n_1 + n_2 + ... + n_k$. Then the number of permutations of S equals:*

$$\frac{n!}{n_1!n_2!...n_k!} \tag{F.9}$$

Appendix G

Model Results for Models Utilising Patterns

G.1 Performance evaluation of the office scenario model

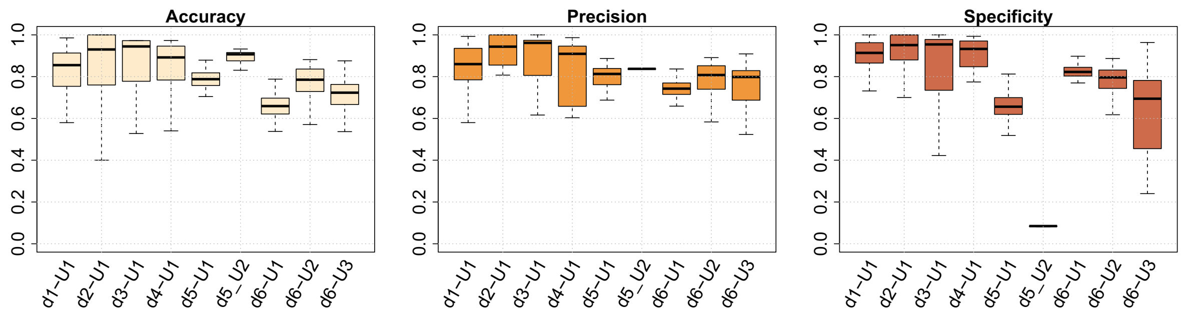


Figure G.1: The figure shows the model performance for the office scenario. In it the x-axis represents the different users for all 6 experiments, and the y-axis is the performance. For each of them accuracy, precision, and specificity are plotted.

G.2 Performance evaluation of the meeting model

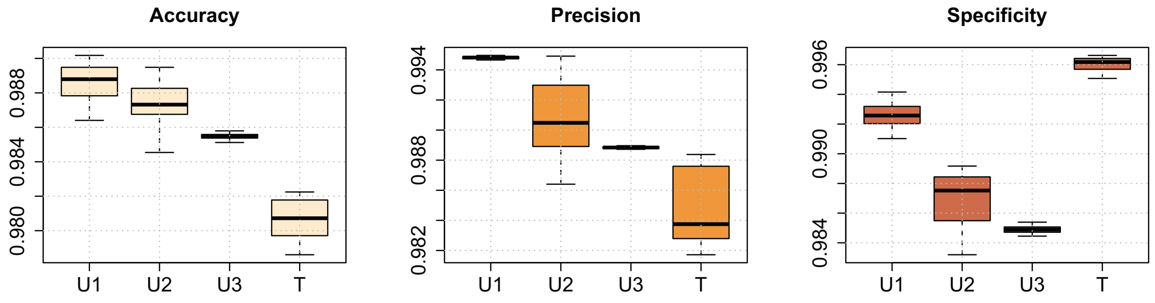


Figure G.2: The figure shows the model performance for the long meeting. Here the x-axis represents the three users and the team. For each of them, the accuracy, precision, and specificity are plotted.

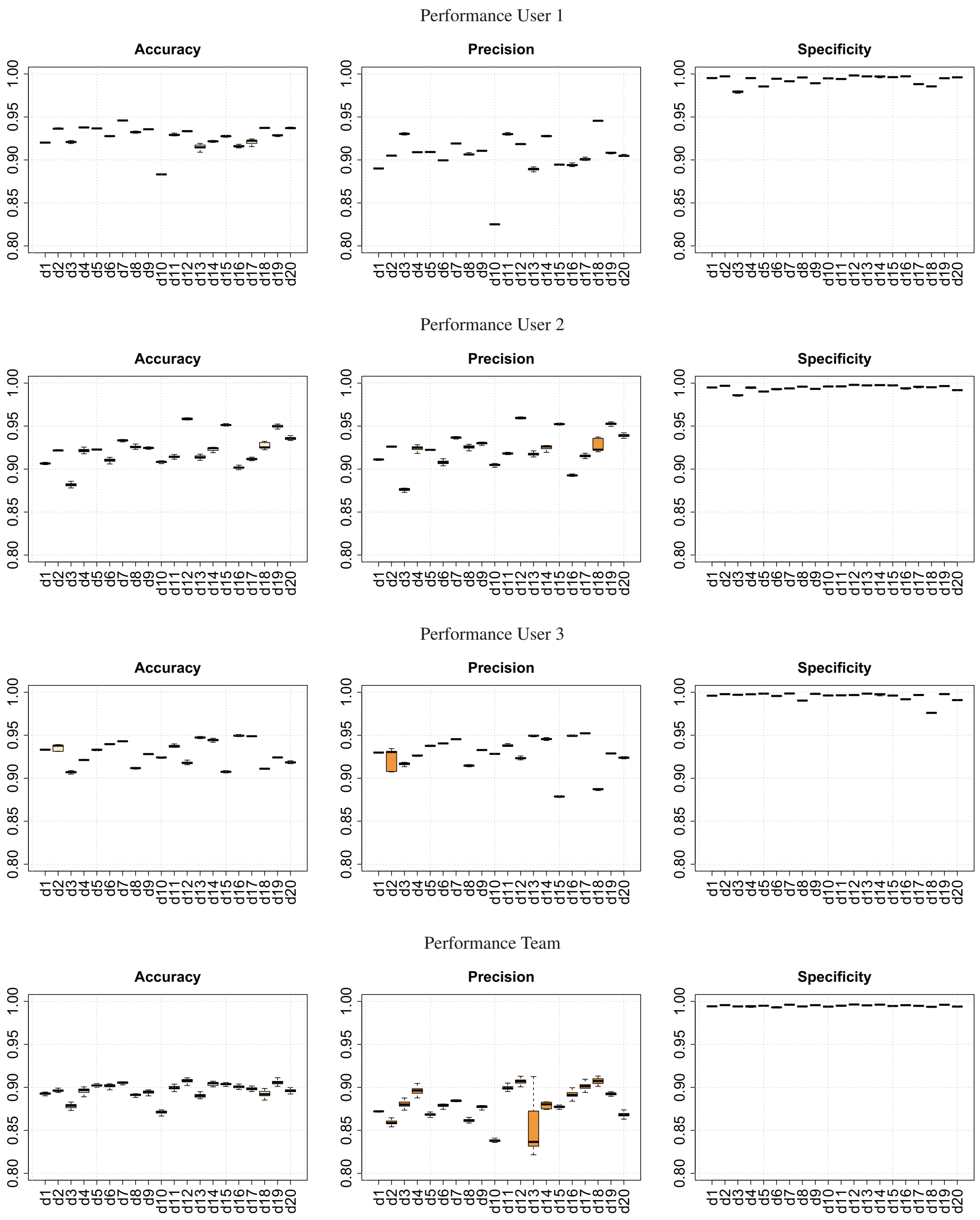


Figure G.3: The figure shows the improved model performance for the 20 short datasets. The x-axis shows the 20 meetings, and y-axis the accuracy. From top to bottom, the first row contains the accuracy, precision, and specificity for user 1, the second row for user 2, the third row for user 3, and the fourth row for the team behaviour.

G.3 Performance evaluation of the cooking task model

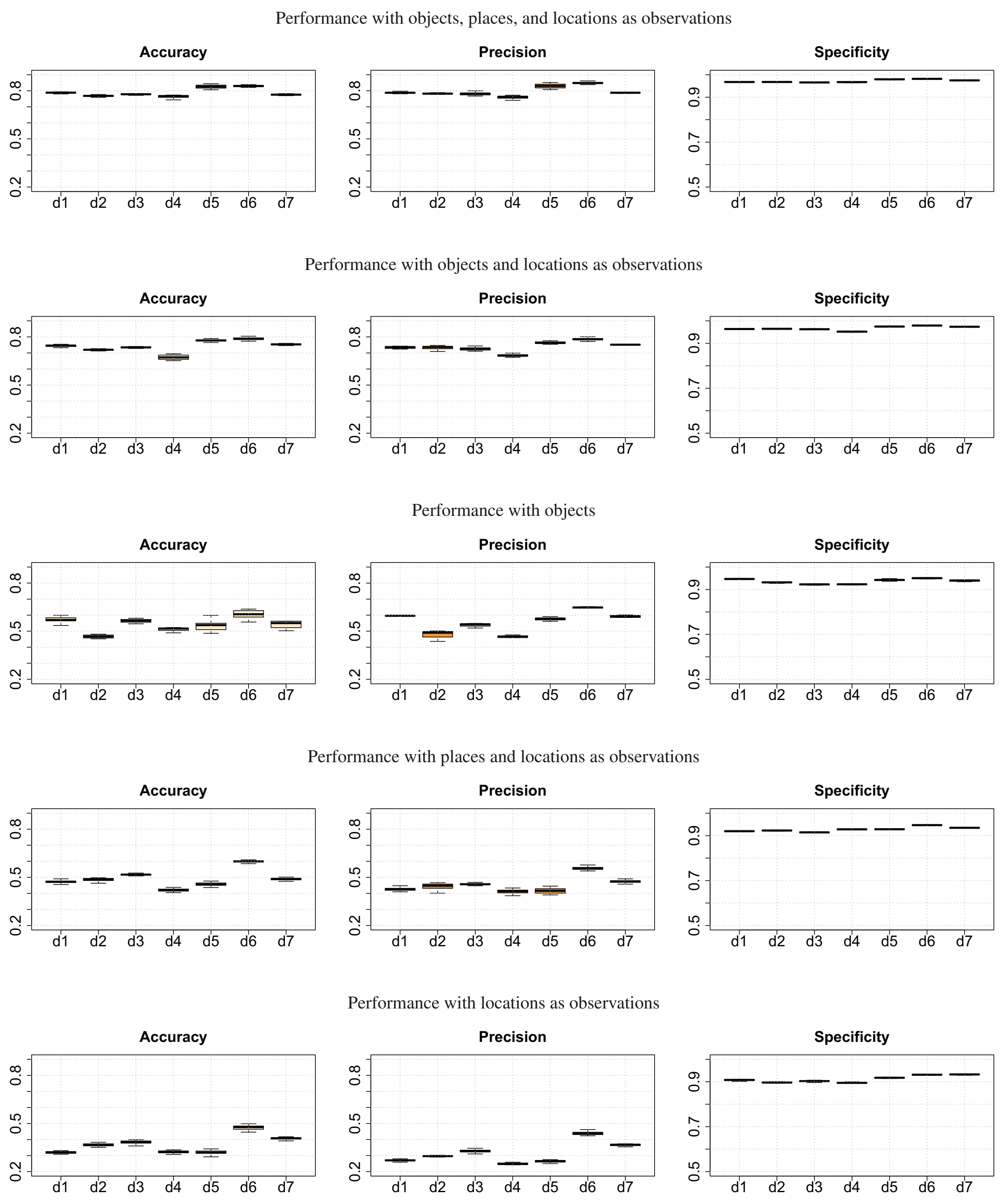


Figure G.4: The figure shows the model performance for the cooking task. In it the x-axis represents the 7 experiments, and the y-axis shows the performance. The first column of plots shows the accuracy, the second the precision, and the third the specificity. The results for different combinations of sensors are plotted.

Appendix H

Friedman Test Results

H.1 Interpreting the results of a Friedman test

The Friedman test is a non-parametric statistical test that is used to detect differences in behaviour across multiple test attempts. It was first introduced by Milton Friedman in his work *The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance* [49]. It is applied by ranking each row together, then considering the values of ranks by columns. In the case of activity recognition, the test can be used to test whether models differ significantly in their performance. That is done by comparing the results for accuracy, precision, recall, and specificity of the different models by applying the Friedman test.

Given data $\{x_{ij}\}_{n\times k}$, that is, a matrix with n rows (the blocks), k columns (the treatments) and a single observation at the intersection of each block and treatment, calculate the ranks within each block. If there are tied values, assign to each tied value the average of the ranks that would have been assigned without ties. Replace the data with a new matrix $\{r_{ij}\}_{n\times k}$ where the entry r_{ij} is the rank of x_{ij} within block i. Then the test statistic can be calculated by the formula

$$Q = \frac{n\sum_{j=1}^{k}(\bar{r}_{\cdot j} - \bar{r})^2}{\frac{1}{n(k-1)}\sum_{i=1}^{n}\sum_{j=1}^{k}(r_{ij} - \bar{r})^2}, \tag{H.1}$$

where $\bar{r}$ is calculated by the formula

$$\bar{r} = \frac{1}{nk}\sum_{i=1}^{n}\sum_{j=1}^{k} r_{ij} \tag{H.2}$$

and $\bar{r}_{\cdot j}$ is calculated by the formula

$$\bar{r}_{\cdot j} = \frac{1}{n}\sum_{i=1}^{n} r_{ij}. \tag{H.3}$$

It is used in deciding whether two models are significantly different and if the difference in the produced results is due to chance or due to external factors. The test is calculated according to Formula H.1 and as a result it outputs three values – the test degree of freedom, the chi square value (denoted as χ^2), and the test p-value. One can then ask the question, what are these statistics and what do they tell us.

The first statistic is associated with the number of categories involved in the test. It is calculated as the number of categories minus one.

The second value is the χ^2 which is an approximation of the Q value in Formula H.1. It summarises the degree of dependence between row and column variables in contingency tables [26, p. 36].

The last value is the p-value which provides the probability that the deviation of the observed results from the expected results is due to chance and not to external factors. The standard used as a threshold is $p > 0.05$ [26, p. 123]. This indicates that any deviation to be due to chance in 5% of the cases or less. According to Cohen [26, p. 123] one usually does not reject the null hypothesis unless $p < 0.05$. Of course, one could choose another p-value as the threshold for the test significance. Table H.1 shows the corresponding χ^2 value for given

Table H.1: Distribution of χ^2 for different degrees of freedom and p-values. Table adapted from [42, Table IV].

Degrees of Freedom (df)	Distribution of χ^2											
	0.95	0.90	0.80	0.70	0.50	0.30	0.20	0.10	0.05	0.01	0.001	Probability
1	0.004	0.02	0.06	0.15	0.46	1.07	1.64	2.71	3.84	6.64	10.83	
2	0.10	0.21	0.45	0.71	1.39	2.41	3.22	4.60	5.99	9.21	13.82	
3	0.35	0.58	1.01	1.42	2.37	3.66	4.64	6.25	7.82	11.34	16.27	
4	0.71	1.06	1.65	2.20	3.36	4.88	5.99	7.78	9.49	13.28	18.47	
5	1.14	1.61	2.34	3.00	4.35	6.06	7.29	9.24	11.07	15.09	20.52	
6	1.63	2.20	3.07	3.83	5.35	7.23	8.56	10.64	12.59	16.81	22.46	
7	2.17	2.83	3.82	4.67	6.35	8.38	9.80	12.02	14.07	18.48	24.32	
8	2.73	3.49	4.59	5.53	7.34	9.52	11.03	13.36	15.51	20.09	26.12	
9	3.32	4.17	5.38	6.39	8.34	10.66	12.24	14.68	16.92	21.67	27.88	
10	3.94	4.86	6.18	7.27	9.34	11.78	13.44	15.99	18.31	23.21	29.59	
	Nonsignificant								Significant			

p-value and degree of freedom. Assuming any difference is significant only when $p < 0.05$, Friedman test values that are placed to the left of this threshold confirm the null hypothesis, while those to the right confirm the first hypothesis. A detailed table of the χ^2 distribution values can be found in [42, Table IV].

Generally, the two hypotheses for the Friedman test are the following.

- *Hypothesis 0:* the distributions of the different categories are the same and any deviation of the expected value is due to chance.
- *Hypothesis 1:* the distributions of the different categories are different and any deviation from the null hypothesis is due to external factors.

More concretely, for the three modelling problems the hypotheses will be the following.

- *Hypothesis 0:* the model performance does not differ significantly given the available datasets.
- *Hypothesis 1:* the model performance differs significantly given the available datasets.

In case the null hypothesis is accepted, that will indicate that the model performs similarly for the different datasets and that it is relatively robust to changes in the data. It will also indicate that the model is relatively general compared to models where the difference in performance is significant.

In case the null hypothesis is rejected and first hypothesis is accepted, that will indicate that the model is affected by changes in the data. This will also be an indication that the model is relatively specific compared to models that do not have significant change in performance.

The two hypotheses can be interpreted as indicators for under-fitting (when the null hypothesis is accepted), or overfitting (when the null hypothesis is rejected). This combined with the results from the model performance points out at the bias-variance dilemma where with decreasing the bias (or the difference between the predicted and the real value), the model variance increases (the model's sensitivity to changes in the data).

H.2 Results for the two problems

Below the results for the three modelling problems can be found, both in the case of the intuitive models from Chapter 3 and the same models with some additional patterns in their implementation from Chapter 4. Here in the case of the meeting model, n is represented by the different user roles (user one, user two , user three, and the team), while k is represented by the available datasets (21 datasets). In the case of the cooking task, n is represented by the different observations with which the user state is estimated, while k is represented by the 7 available datasets. In the case of the office scenario there was only one type of observations and due to the varying number of user and their unsynchronised behaviour, each of the users was considered as a separate dataset. For that reason, n was represented by the accuracy, precision, and specificity, while k by the available datasets. The same was also calculated for the meeting problem and the cooking task.

Table H.2: Friedman test results from the two meeting models. The test was performed to check whether the model results differ given the different datasets. It was performed for the multi-agent model from Chapter 3, as well as for the multi-agent model from Chapter 4. The significance was tested for accuracy, precision, and recall.

	chi squared			degree of freedom			p-value		
	accuracy	precision	specificity	accuracy	precision	specificity	accuracy	precision	specificity
intuitive	41.23	29.65	40.66	20	20	20	0.003476	0.07574	0.004115
with patterns	35.97	32.46	38.61	20	20	20	0.01549	0.03856	0.007451

Table H.3: Friedman test results from the two cooking models. The test was performed to check whether the model results differ given the different datasets. It was performed for the model from Chapter 3, as well as for the model from Chapter 4. The significance was tested for accuracy, precision, and recall.

	chi squared			degree of freedom			p-value		
	accuracy	precision	specificity	accuracy	precision	specificity	accuracy	precision	specificity
intuitive	7.2	13.71	11.48	6	6	6	0.3027	0.033	0.07448
with patterns	18.85	21.08	6.85	6	6	6	0.004412	0.001771	0.3343

Table H.4: Friedman test results for the three modelling problems with combined accuracy, precision and specificity. The test was performed to check whether the model results differ given the different datasets. In it the accuracy, precision, and specification for the different user roles (in the three-person meeting), or the different sensors (in the cooking task), were taken as rows, while the different datasets were taken as columns.

	chi squared		degree of freedom		p-value	
	intuitive	patterns	intuitive	patterns	intuitive	patterns
meeting	55.65	63.90	20	20	3.277e-05	1.742e-06
cooking task	11.88	37	6	6	0.06457	1.761e-06
office scenario	13.95	20.88	8	8	0.08293	0.007449

Appendix I

Wilcoxon Test Results

I.1 Interpreting the results of a Wilcoxon test

The Wilcoxon Rank Sum Test is a non-parametric test which is used in situations where the underlying distribution is non-normal or when the distribution is not known. It was first introduced by Frank Wilcoxon in his work *Individual Comparisons by Ranking Methods* [154]. It is used in deciding whether two models are significantly different and if the difference in the produced results is due to chance or due to external factors. The test is calculated based on the sum of sample ranks and as a result it outputs two values – the Wilcoxon statistic value (denoted as W), and the test p-value. One can then ask the question, what are these statistics and what do they tell us.

The first value is the rank assigned to each value in the sample. In the case where the observations are tied together, or have ties they get the average rank calculated by $\frac{\sum_{i=0}^{n}(rank+i)}{i}$ with n being the number of observations that are tied. When all ranks are calculated, they are then summed up for each of the two samples and significance is assigned on the sample with the smaller sum of ranks. More details about the calculation procedure in R can be found in *Statistics: An Introduction Using R* [28, p. 79].

The second value is the p-value which provide the probability that the deviation of the observed results from the expected results is due to chance and not to external factors. The standard used as a threshold is $p > 0.05$ [26, p. 123]. This indicates that any deviation to be due to chance in 5% of the cases or less. According to Cohen [26, p. 123] one usually does not reject the null hypothesis unless $p < 0.05$. Of course, one could choose another p-value as the threshold for the test significance.

To decide on the significance on the difference in results (or with other words, whether to reject the null hypothesis), on can use lookup tables for the W values. Or alternatively, in the case the p-value is smaller than the significance threshold value of 0.05%, the null hypothesis is rejected and the alternative hypothesis is accepted.

In the case of the Wilcoxon test the two hypotheses are the following.

- *Hypothesis 0:* there is no significant difference between the performance of the two models and any differences are due to chance.

- *Hypothesis 1:* there is significant difference between the performance of the two models and any differences are due to external factors.

I.2 Results for the three problems

Below the results from the Wilcoxon test comparing the models from Chapter 3 and the corresponding models with additional patterns from Chapter 4 are given. The test compared the accuracy, precision, and specificity for each of the given users in the case of the 3-person meeting; for each of the different observations in the case of the cooking task; while in the office scenario there was only one type of observations and unsynchronised users so, only one test per accuracy, precision, and specificity were performed.

Table I.1: Wilcoxon test results for the meeting scenario. The test compares the accuracy, precision, and specificity for the multi-agent meeting model in Chapter 3 with those of the model in Chapter 4. The table provides the statistical value that gives the rank assigned to the different user roles and the corresponding p-value for each user.

	statistic value				p-value			
	User 1	User 2	User 3	Team	User 1	User 2	User 3	Team
accuracy	231	219	158	231	9.54e-07	6.68e-05	0.00171	9.54e-07
precision	163	108	108	208	1.03e-01	8.12e-01	0.81168	6.07e-04
specificity	224	205	158	201	1.81e-05	1.00e-03	0.14696	1.86e-03

Table I.2: Wilcoxon test results for the cooking task scenario. The test compares the accuracy, precision, and specificity for the model in Chapter 3 with those of the model in Chapter 4. The table provides the statistical value that gives the rank assigned to the model performance for the different observations, and the corresponding p-value for each observation type.

	statistic value					p-value				
	OHL	OL	O	HL	L	OHL	OL	O	HL	L
accuracy	27	28	28	15	25	0.0313	0.0156	0.0156	0.9375	0.0781
precision	27	27	28	28	7	0.0313	0.0313	0.0156	0.0156	0.2969
specificity	28	28	28	10	23	0.0156	0.0156	0.0156	0.5781	0.1563

Table I.3: Wilcoxon test results for the office scenario. The test compares the accuracy, precision, and specificity for the model in Chapter 3 with those of the model in Chapter 4. For each of them the statistical value and the corresponding p-value are given.

	statistic value	p-value
accuracy	28	0.0225
precision	35	0.0209
specificity	35	0.0209

Appendix J

Evaluation Metrics for Activity Recognition

In this appendix we give detailed information about the evaluation metrics typically used in the field of activity recognition and discussed in Chapter 5. The formulas presented here can be found in variety of data analysis and machine learning publications, for example in [113].

Accuracy: According to Fig. 5.22 accuracy for a two-class problem is calculated as

$$Accuracy_M = \frac{\sum tp + \sum tn}{\sum tp + \sum tn + \sum fp + \sum fn} \tag{J.1}$$

where M is the model being tested, tp are the the correctly recognised positive instances, tn are the correctly recognised negative instances, fp are the instances incorrectly recognised as positive, and fn are the instances incorrectly recognised as negative.

When we have a multi-class problem, the accuracy is reduced to calculating the recall for a multi-class problem. Then given a confusion matrix with N classes, the accuracy is calculated by taking the sum of the matrix diagonal and dividing it by the number of classified instances. With other words

$$Accuracy_M \text{ (Multi-class)} = \frac{\sum_{i=1}^{N} e_{ii}}{\sum_{i=1}^{N}\sum_{j=1}^{N} e_{ij}}, \tag{J.2}$$

where e is a matrix cell. The index ii indicates that only the diagonal of the matrix is taken. It is then divided by the number of all instances. The formula could also be represented by using the two-class recall from Equation J.6. In that case it will look the following way.

$$Accuracy_M \text{ (Multi-class)} = \frac{\sum_{i=1}^{N} Recall_i}{\sum_{i=1}^{N}\sum_{j=1}^{N} e_{ij}}, \tag{J.3}$$

Precision: Using the same notations and according to Fig. 5.22 precision in a two-class problem is represented by the formula

$$Precision_M = \frac{\sum tp}{\sum tp + \sum fp}. \tag{J.4}$$

In the case of a multi-class problem, the precision is calculated by summing up all precision values for the individual classes and then dividing them by the number of classified instances.

$$Precision_M \text{ (Multi-class)} = \frac{\sum_{i=1}^{N} Precision_i}{\sum_{i=1}^{N}\sum_{j=1}^{N} e_{ij}}. \tag{J.5}$$

Recall: According to Fig. 5.22, recall for a two-class problem is defined by the formula

$$Recall_M = \frac{\sum tp}{\sum tp + \sum fn}. \tag{J.6}$$

In the case of a multi-class problem, the recall equals the accuracy and is calculated according to Formula J.3.

Specificity: According to Fig. 5.22 specificity is represented by the formula

$$Specificity_M = \frac{\sum tn}{\sum tn + \sum fp}. \tag{J.7}$$

In the case of a multi-class problem it is then calculated by summing up all specificity values for the individual classes and then dividing them by the number of classified instances.

$$Specificity_M \text{ (Multi-class)} = \frac{\sum_{i=1}^{N} Specificity_i}{\sum_{i=1}^{N} \sum_{j=1}^{N} e_{ij}}. \tag{J.8}$$

Hypothesis testing: To test a given hypothesis in a statistical manner one can use different statistical tests. The process of choosing what kind of test to apply is illustrated in Fig. J.1.

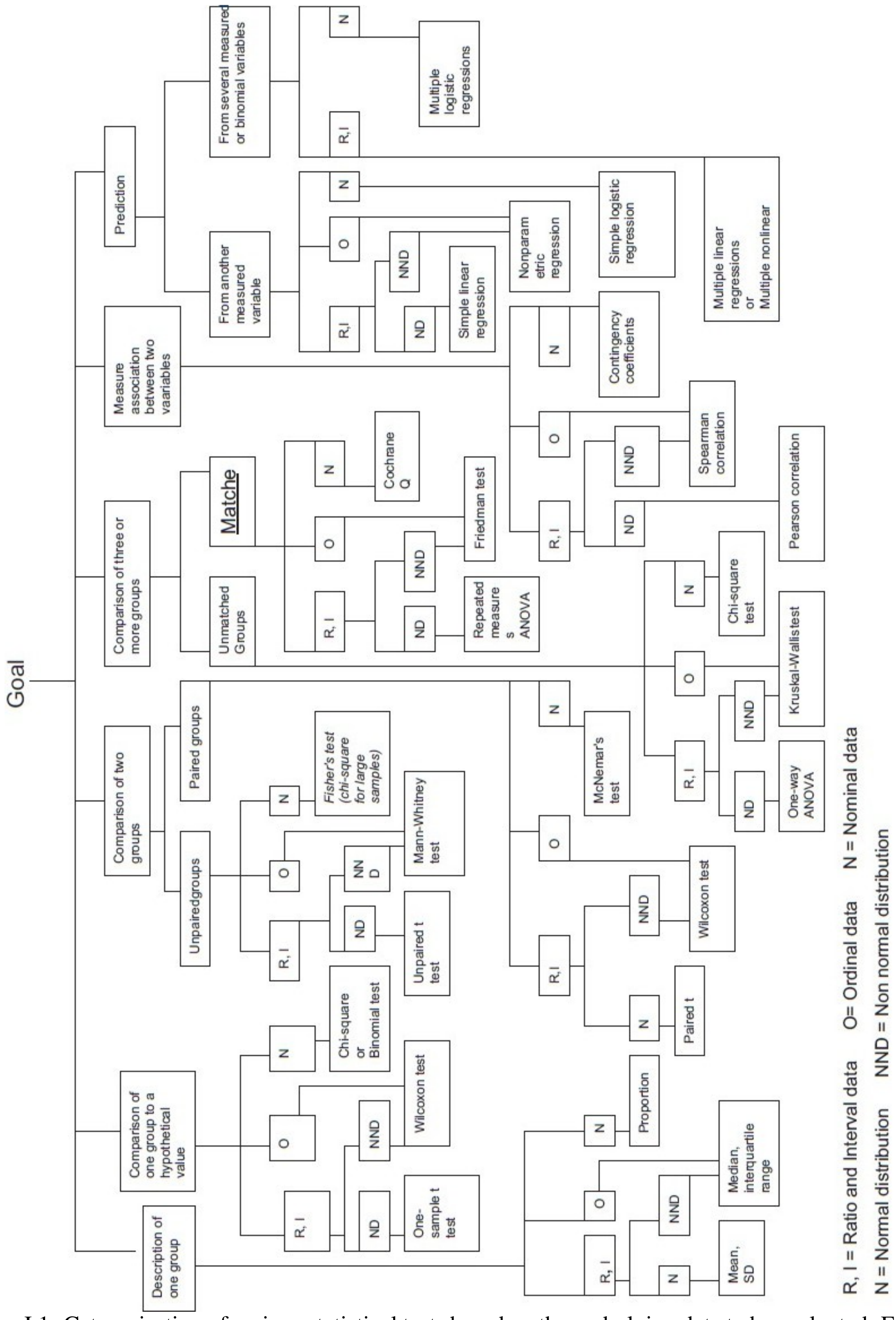

Goal
Description of one group
Comparison of one group to a hypothetical value
Comparison of two groups
Comparison of three or more groups
Measure association between two vaariables
Prediction
Unpairedgroups
Paired groups
Unmatched Groups
Matche
From another measured variable
From several measured or binomial variables
R, I
O
N
ND
NND
NN D
Mean, SD
Median, interquartile range
Proportion
One-sample t test
Wilcoxon test
Chi-square or Binomial test
Unpaired t test
Mann-Whitney test
Fisher's test (chi-square for large samples)
Paired t
McNemar's test
One-way ANOVA
Kruskal-Wallistest
Chi-square test
Repeated measures ANOVA
Friedman test
Cochrane Q
Pearson correlation
Spearman correlation
Contingency coefficients
Simple linear regression
Nonparametric regression
Simple logistic regression
Multiple linear regressions or Multiple nonlinear
Multiple logistic regressions
R, I = Ratio and Interval data   O= Ordinal data   N = Nominal data
N = Normal distribution   NND = Non normal distribution

Figure J.1: Categorisation of various statistical tests based on the underlying data to be evaluated. Figure adapted from [73].

Appendix K

Example Directory Structure for Results Reproducibility

In order to yield reproducible results, one has to also provide a reproducible directory structure where all data, models, experiments, and evaluations are saved. Below is an example structure that ensures separation between these elements as well easy traceability.

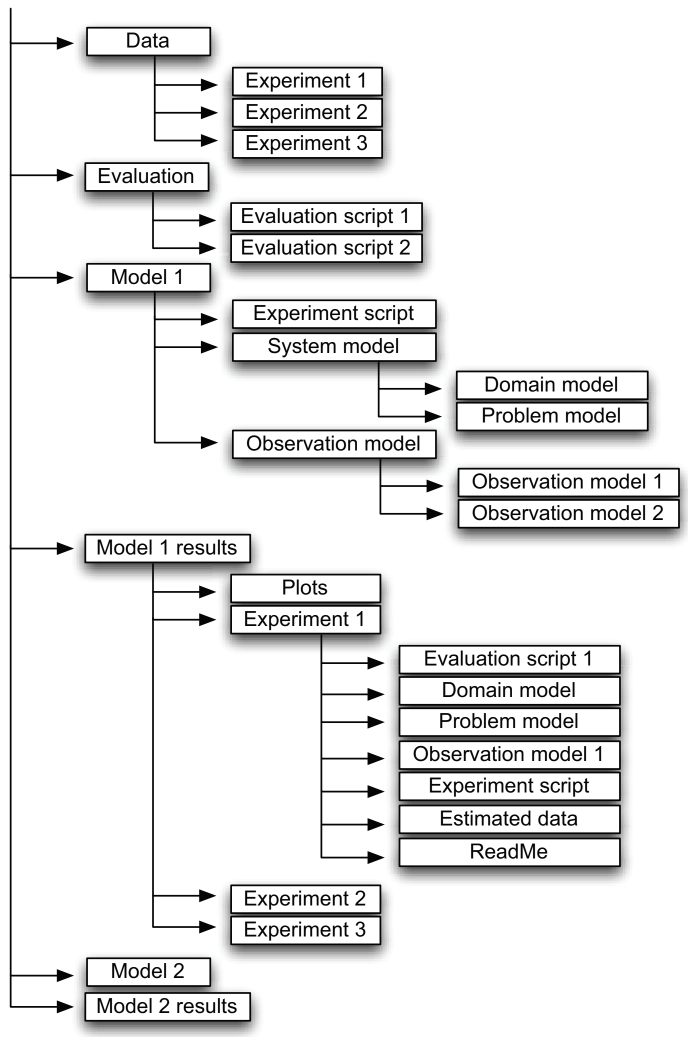


Figure K.1: An example directory structure for better results reproducibility.

Theses

1. Symbolic modelling of user behaviour allows incorporating prior knowledge in assistive systems. This reduces the need of training data and provides rich context information about the user state.

2. For a formalism to be suitable for symbolic modelling of user behaviour, it has to satisfy a set of requirements concerning the behaviour dynamics to be modelled as well as the targeted model application.

3. Generative rule-based models are suitable for modelling user behaviour for activity recognition, because they rely on compact algorithmic representation for the transition model of the underlying dynamic system. They allow coping with large state spaces and behaviour variability.

4. Generative rule-based models, combined with Bayesian filters, allow utilising the ability to model behaviour variability with the ability to reason in probabilistic manner about the user state, histories and goals. This provides a means for compact representation of complex domains, and interchangeable observation models making the behaviour structure independent of the underlying sensor infrastructure.

5. The process of creating symbolic models for activity recognition, that make use of Bayesian filters, is a complex task caused by the combination of symbolic and probabilistic representations. This is also true for seemingly trivial problems.

6. To ensure high model performance, traceability of modelling solutions, and results reproducibility, a structured development process for symbolic human behaviour models for activity recognition needs to be introduced. This is an aspect of activity recognition that is often neglected while the stress is put on run time modelling.

7. There is no state of the art development process for symbolic human behaviour models for activity recognition that discusses the problem of combining symbolic and probabilistic modelling aspects.

8. The introduced development process for symbolic human behaviour models for activity recognition combines state of the art software engineering development processes and data analysis techniques, thus bridging the gap between symbolic and probabilistic model development. It also introduces custom procedures for developing the models' probabilistic elements. The process ensures model traceability and model and results reproducibility. It also provides a guideline for building and evaluating symbolic models for activity recognition.

9. Symbolic models that rely on causal relations build up complex structures. In order to be able to evaluate the correctness of such structures, the corresponding ground truth has to

correctly represent them. This indicates that the evaluation of such approaches requires that the corresponding ground truth is represented not only by a set of labels but also that the relations between these labels are causally correct.

10. Provided there are no well performing action selection mechanisms, there is a need for modelling mechanisms to reduce the model complexity. Such mechanisms can be expressed in terms of modelling patterns applicable to different problem domains.

11. The modelling toolkit for Computational Causal Behaviour Models provides mechanisms for reducing the model complexity in terms of available actions that can be executed from a given state, number of reachable states, and number of valid plans. These parameters are controlled through the model predicates and the objects' type system. Models that employ these patterns perform better than models developed in an intuitive manner.