MSc Thesis

User modeling for Complex Personalization Problems

A general methodology and BDI-agent-based framework

Universiteit Utrecht

TNO innovation for life

Author:
C.C.M. van Rooij
ICA: 3412326

Supervisors:
MSc. M.M.M. Peeters
Prof. Dr. J-J. Ch. Meyer (UU)
Dr. K. van den Bosch (TNO)

submitted in fulfilment of the requirements for the degree of
Master of Science

in the programme
Technical Artificial Intelligence

at the
Department of Information and Computing Sciences
Utrecht University

January 16, 2014

Abstract

The research field of user modeling is concerned with the complex process of adding personalization to an application with the use of a user model. In this thesis, to support developers and researchers with the employment of personalization, a user modeling methodology and a framework for BDI-agent-based user modeling are proposed. The methodology is meant as a dynamic approach to bridge the current gap between domain-specific and (too) general user modeling methods. The framework is developed to employ the modular, dynamic approach of the BDI-based intelligent agent paradigm to develop a user model for complex personalization problems. By means of a thorough review of the user modeling field, a more abstract overview is created and used to construct both the methodology and the framework. To validate and test the resulting methods, both are applied to the problem of adding personalization to an adaptive educational game in First Aid. The resulting application is verified by means of use cases analysis. The methodology proved to be a thorough and useful method of designing a user model, by making a clear distinction between the conceptual ideas underlying the personalization and the technical implementation of the user model. In this way, a systematic and gradual construction of the model could take place. The framework proved to be a useful way of capturing the processes and features of a user model, especially through the strict and clear architecture imposed by the BDI paradigm. With the framework, developers can utilize the flexibility, robustness and pro-activeness of intelligent agents for the implementation of their user model. Due to the extensiveness of both methods, they are especially useful for more complex personalization problems and scientific research on user modeling. Future work is mainly focused on applying the methodology and framework to additional personalization problems and domains, to further validate and possibly expand them. We hope that this thesis will serve as a (first) step for researchers, designers, and developers to get a better grip on their user modeling problem.

Contents

Chapter 1

Introduction

In the past few decades, due to the advancements in computer technology, computer systems have become increasingly more important and omnipresent. With the rise of the computer for personal use, also the personalization of computer applications has become important. Personalization within computer applications aims to provide users with experiences that fit their specific background knowledge and objectives, and thus to make applications more usable and useful (Fischer, 2001).

Nowadays, it is hard to imagine everyday life without personalization in computer applications. For instance, web shops and social media constantly present you with personalized offers, topics you might like or persons you might know; the fitness application on your smart phone provides you with tips for a healthier life based on your recent activities; computer games keep you occupied by presenting suitably challenging opponents; when using a search engine such as Google, the results are ordered and altered according to your incentive; and even in your house a smart energy meter registers your energy usage and presents you an overview and tips for energy savings.

Personalization of a computer application can be done in two ways. The first way is to equip the user with the possibility to customize the application, for instance through the graphical user interface. However, this requires a certain effort and knowledge from the user. The second way solves this problem by equipping the application with the possibility to automatically adapt to the user's need: the application is personalized without explicit efforts from the user. Automated personalization will be the subject of this thesis.

Especially in the early days of employing personalization, the implementation of automated personalization used to be application-specific. Because the personalization was constructed implicitly in the code, the method of personalization was only usable within that single application, and therefore difficult and expensive to use in other applications or domains. To solve these problems, the research field of user modeling evolved, concerned with the process of constructing a separate module responsible for the personalization of the application to the user's needs. By clustering the different aspects related to personalization within one model, the process of personalization and the resulting user model became adjustable, extendible and reusable (McTear, 1993; Kobsa, 2001).

1.1 Problem description

When implementing automated personalization with the use of a user model, several problems and difficulties arise. Often, developers can only anticipate on the tasks and contexts in which the application will be used. Nonetheless, developers are burdened with the task to construct a user model at design time, that should be able to provide the experience of a tailor-made product at use time (Fischer, 2001).

Considering the user modeling research field as a whole, there are many domains in which user modeling for personalization can be applied. Each domain has its own requirements, possibilities and problems, making the field broad and complex. For instance, when designing the personalization process of a web shop, you will have to look at completely different aspects than when looking at the personalization process of an educational application. Besides, due to the broad applicability of personalization within applications, many problems associated with user modeling are also part of other research fields, such as knowledge representation, information retrieval, and human-computer interaction, each associated with its own challenges and open questions in their own research domain.

To make the process of user modeling easier, researchers have proposed various methodologies, frameworks and architectures to aid the developer. However, due to the problems mentioned above, researchers agree that it is hard to develop a single method to solve every user modeling problem (McTear, 1993; Kobsa, 2001; Fischer, 2001). This observation has resulted in a few general approaches for user modeling

and a wide variety of domain specific user modeling methods (Webb, et al., 2001; Lorenz, et al., 2005; Deepa, et al., 2012), focusing only on one application domain or one specific way of personalization.

1.2 Thesis focus

The goal of this thesis is to help developers and researchers with the employment of personalization in their application, by providing support for the process of user modeling. This is done by investigating two related user modeling subjects: a user modeling methodology and a framework for BDI-agent-based user modeling.

The general user modeling methodology is meant to bridge the gap between the high-level, conceptual user modeling methods, and the low-level domain-specific user modeling methods. Due to the difficulty of developing a method applicable to each user model problem and domain, the aim is not to solve every possible generic and domain-specific problem that can arise, but to formalize the general questions and an extendible list of possible answers that should lead to a correct and optimal user model.

Supplementary to the methodology, to help developers with the actual implementation of a user model for a personalization problem, a framework based on the intelligent agent paradigm is proposed. An intelligent agent is an autonomous software entity, capable of performing goal-oriented actions in a certain environment (Wooldridge, 2009). Intelligent agents possess interesting features, such as the possibility of flexible behavior, robustness and pro-activeness, which make them an interesting implementation technique for a user model. Agents have been used before for creating a user model (Kobsa, 2007; Razmerita, 2009), but an extensive investigation to this combination has not been done before. To enforce more structure to the agent-based user model framework, we look at the Beliefs-Desires-Intentions (BDI) architecture for intelligent agents. The BDI paradigm provides a model for formalizing the way in which humans (think they) reason, forming goals and using their knowledge and beliefs to construct a plan to achieve these goals.

In order to construct the methodology and framework, the user modeling field is extensively studied. Through this review, the different concepts, properties, and functional components related to user modeling are formalized and combined to compose the methodology. In addition, the review is used to combine the field of user modeling with the field of intelligent agents, resulting in the agent-based user model framework. To investigate the use and usefulness of the resulting methodology and framework, both are applied to an adaptive educational game (AEG) in First Aid (Peeters, 2014a). The resulting application is verified by comparing the behavior of the implementation and the desired personalization, formalized in use cases.

Summarizing, in this thesis, we present the design of a general user modeling methodology and an agent-based user model framework. We evaluate the use of this methodology and framework to address the following main question: "*How can the process of adding personalization to an application with the use of a user model be supported?"*, with the sub-questions:

1. *How can the gap be bridged between high-level and low-level user modeling methods?*
2. *How can the intelligent agent paradigm be used to develop a user model, how effective is a user model based on a BDI-agent, and in what type of applications would a BDI-based user model be useful?*

1.3 Thesis outline

Chapter 2 presents an overview of the user modeling field. The general technical requirements, main application domains and methods are discussed. In Chapter 3, this overview is used to construct a more abstract view of user modeling, addressing the general concepts, properties, and functional components. This abstract view is used to construct the general user modeling methodology. In Chapter 4, the intelligent agents and BDI paradigm are introduced. In Chapter 5, the concepts from Chapter 3 are combined with the intelligent agent paradigm, which results in an agent-based user model framework. Next, to investigate the use and usefulness of both the methodology and framework, a user model is constructed for an Adaptive Educational Game in First Aid. Chapter 6 discusses Adaptive Educational Games, and Chapter 7 describes the implementation and verification of this user model. Lastly, Chapter 8 describes the discussion and conclusion of the thesis.

Chapter 2

User Modeling Review

User modeling can be described as the process of creating a model of the user to personalize a system. The user model is usually constructed as the user is working with the system. An example is an educational application that teaches students a certain skill: according to the rules and knowledge in the user model, the difficulty level of the exercises in the application is altered as the user progresses. We formally define user modeling according to McTear (1993, p. 158): (u*ser modeling is) the process of gathering information about the users of a computer system and of using the information to provide services or information adapted to the specific requirements of individual users (or groups of users)*. The resulting user model consists of a module containing the *operations* that are needed to personalize the system, and the user *profile*, which contains the personal data from the user (Mohamad, et al., 2013).

System personalization through user modeling is tightly related to the research field of adaptive systems. Adaptive systems are systems that automatically sense or track information from the environment, task, system, or user and adapt to help applications to be more effective despite changing conditions (Feigh, et al., 2012). In contrary to this more general field, user modeling is solely focused on systems in which there is interaction with the user, for example through a graphical user interface. Because of this focus on the human user, user modeling is a very cross disciplinary research topic, comprehending the domains of artificial intelligence, computer science, and social science. Ideas have been co-opted from a large range of sub domains, such as human–computer interaction, e-learning, information science, social computing, machine learning, data mining, cognitive science, and so on (Kay, et al., 2012; Kobsa, 2001). New insights and techniques developed in one of these areas, have led to quick advancements in the user modeling area as well. An example of this is machine learning, a sub field of artificial intelligence, where developments led to improvements in speed and efficiency of user preference prediction. In opposite direction, the user modeling application area is also a tough crucible for any technique, making it a useful application and testing area. There is interest in user modeling from both a scientific and commercial perspective (Razmerita, 2009).

In the literature, the term user modeling is also used for two other things. First, the term can be used for the process of defining a user when designing a system. The resulting model is used to create and construct a system, and to make sure that the system serves the end user's means. Second, user modeling can refer to simulating different kinds of users within a new system. Instead of using real humans for broadly testing and evaluating the system, a 'user model' can be created, which is a computational representation of a certain type of user, such as a novice or an expert. In contrary to the definition of user modeling in this thesis, both static user models become useless after the design phase is over.

In this chapter, an overview of the user modeling field is given. The most important requirements and application domain aspects will be discussed, according to the authors. In Section 2.1, the technical requirements are discussed that are independent of the different application domains of user modeling. In Section 2.2, an overview of the current major fields of appliance of user modeling is given. In Section 2.3, the methodologies, architectures and frameworks used for user modeling will be discussed.

2.1 Technical software development requirements

In early attempts for personalization, adaptation of the system to the user was performed by the application itself. No clear distinction was made between system components that served user modeling purposes and components that performed other tasks (Kobsa, 2001). Around the mid-eighties, this separation was made more clearly by using a modular approach to personalization, and thus the field of user modeling emerged. To ease the process of user modeling, strong technical principles and requirements were defined, independent of the specific demands a certain application domain might require. These principles and requirements have their

origin in paradigms and ideas from the software development research field (Fischer, 2001; Kobsa, 2001; McTear, 1993). The most important requirements are:

- **Software decomposition**, meaning the strict separation between components belonging to the user model and components belonging to the rest of the system, as well as the clear distinction between the different kinds of data within the system. Considering decomposition, another thing that must be taken into account is that the input and output of the different components should still correspond to each other
- **Modifiability**, meaning the possibility for future developers to modify the user model, either for maintaining or extending the system. This implies that the user model must be transparent and clear for programmers
- **Software reusability**, meaning the possibility for programmers to use (part of) the user model in future applications, possibly by constructing subsystems on which other systems can build on. User models are desired to have a certain degree of generality, enabling the models to be usable in multiple application and content domains, by extending the given structure.

To satisfy these different requirements, the user model must contain a certain level of abstraction. This also gives rise to the desire that the user model must be minimal, making the model effective, efficient with data and easier to comprehend.

Because the application domain and the desired functionalities also greatly influence the user modeling process, it is not always possible to obtain all of the desired technical requirements in the user model. For example, when using certain machine learning techniques such as a Bayesian network for implementing a certain user model functionality, it might not be possible to make a clear distinction between the data gathered and the functionalities for gathering this data.

2.2 Application Domains for user modeling

There are several major research and application domains in which personalization and thus user modeling plays an important role. This section presents an overview of these domains. In order to bring some structure in this overview, the different topics are divided into three general categories:

- supporting a user during a task
- giving a user a specific personalized experience
- training and educating a user

The categories especially differ in the kind of user data that is used. For each domain, the general purpose of the domain and the more specific purpose of the user model are discussed. Additionally, the data associated with the user model, the exact adaptation that takes place and other important details are discussed. The aspects that are identified in this way, will be used in the next chapter to create a more abstract and general view of the user modeling domain.

2.2.1 User models for providing task support

'Task support systems' is the generic term of systems that support a user during a task by either helping the user perform the task or by completely taking over this task (Nurmi, et al., 2007; Brun, et al., 2010). An example is an application that automatically categorizes the incoming emails of the user.

The general goals of the user model in these applications is to improve the efficiency of interactions with the user, to simplify these interactions and to make complex systems more useable (Razmerita, 2009; Fischer, 2001). To accomplish this personalization, data is gathered through observations of the user. This data is related to the user's goals and needs, but especially to the task that the user currently is performing, like the user's task knowledge and background.

Much research has been done in the domain of task support systems, but because there are many isolated research projects focusing on a specific task or topic (Sannes, 2011), it is hard to make generalizations or to

identify one delimited research topic. Broadly discussed research topics are **Decision Support Systems**, **Adaptive Hypermedia** and **Adaptive Ubiquitous Systems**, each having their own specific domain and way of personalization.

Decision support systems:

Decision support systems are systems that help a user with making a decision in a complex, often professional environment (Nurmi, et al., 2007). An example is a system used at a pharmacy for automatically checking valid combinations of medicine. The system can be used to help the pharmacist in prescribing the right combinations and to give information for making a decision when a problem occurs.

- The *purpose* of the user model in decision support systems is to present the user with the right and appropriate information, giving different feedback or applying different decision steps according to the characteristics of the user.
- The *data* that is used is often associated with the user's task and background knowledge.
- The *adaptation* takes place by adapting the amount and the content of the feedback provided by the system.

Decision support systems are traditionally rule - or logic based systems, in which all the relevant information is represented in a knowledge base. This means that the content of the user model itself is also highly dependent on the way the rules and knowledge are represented.

Adaptive Hypermedia

Strictly speaking, a hypermedia system is a system that allows users to freely scan an information network, structured by means of nodes and links, in order to retrieve items of information (Nurmi, et al., 2007; Deepa, et al., 2012). An example of such an application is an Internet website.

- The *purpose* of the user model is to make the interface and structure of the system dynamic. This enables the application to adapt to the user and to make it easier for the user to search for and retrieve relevant information.
- The *data* that is used in the user model is related to the user's abilities, knowledge, and goals in the application.
- The *adaptation* takes place by adjusting the structure and the presentation style to the assumed needs of the user. For example, by enhancing web search: promoting pages that might better correspond to the user's characteristics, or by providing navigation support, through highlighting certain elements of a page (Razmerita, et al., 2012).

Adaptive ubiquitous systems

The research field of ubiquitous systems is concerned with information processing applications integrated into everyday objects and activities (Nurmi, et al., 2007). An example is the smart energy meter, recording the energy usage in a household through small devices distributed in a house, helping the user with regulating this energy usage (Hargreaves, et al., 2010).

- The main *purpose* of the user model is to enhance the system, facilitate the user's preferences and thus make the overall use easier.
- Because the personalization can take place in every situation and location, the *data* is focused on the user state and context. For example to enable the contextualization to a current environmental change.
- The *adaptation* takes place by changing the behavior of, and the feedback given by the whole system.

In general, user modeling in ubiquitous systems builds on requirements and functionalities already seen in other supportive systems, but because the personalization can take place in every situation and location, it

does introduce some interesting (technical) challenges (Specht, et al., 2006). In the network of small applications surrounding the user, an open client-server network is often used. Traditionally, all data from the user is centralized on the server side for analyzing and making a decision on adaptation, but because of the distributed nature of these systems, the isolated parts also need a *local presentation* of the user and a certain degree of *autonomy* to make adaptations on their own. This makes performing the user model purposes with minimal means - both computationally and considering memory, important. On a higher, centralized and more abstract level, user personalization also takes place. Balancing out this collaboration is an important and delicate matter in ubiquitous systems.

2.2.2 User models for providing a personal experience

User models for providing the user with a personal experience have the goal to enhance the user experience while using the system. This kind of user modeling is especially focused on more commercial fields, such as e-commerce, marketing and computer games, and became popular with the rise of the Internet.

The data that is used from the user in this main domain is mostly focused on the information that defines the user, such as the **user's preferences and interests**. Because this information is often sensitive, **privacy** is a big issue (Toch, et al., 2012). While in other domains the privacy of the user data is also important, in this domain it is even a bigger point of discussion because the incentive of the application developers is often contradictive to the incentive of the actual user, considering gaining and sharing the user´s personal information. For example, user profiles are often shared between different components of the same application, or even with other applications (Brun, et al., 2010; Karam, et al., 2012), which introduces additional weaknesses and possible unwanted data sharing. Making sure personal information is not accessible to all people, in addition to defining strict privacy policies, is thus essential in these user models.

Popular research topics in this domain are **Recommender Systems** and **User Adaptive Computer Games**.

Recommender Systems

Recommender systems are concerned with presenting the user with relevant information and recommendations. They are mostly used on the Internet, for example on websites such as Facebook, to provide the user with personalized news, targeted advertisements and possible new friends (Brun, et al., 2010).

- The *purpose* of the user model is to provide the system with information that is assumed to be relevant for the user.
- The *data* that is recorded for this purpose is associated with the preferences of the user to certain objects, like products, music or people. To gain a classification of these objects, the interaction history of the user is recorded, or the user is explicitly asked to rate certain objects.
- The content of the system is eventually *adapted* by presenting the newly inferred objects. These objects can be inferred by looking at the properties of the objects in the user profile, or by looking at the objects in other user profiles that are similar to the user (Kobsa, 2001; Kay, et al., 2012). Because of the predominantly commercial goal of these systems, the adaptations often take place in a very intrusive way, to make sure the user notices the change.

Most recommender systems are based on the Internet, which means that some typical technical difficulties are associated with these kinds of user models. First, the user profile is often only saved during the user's visit, which means that fast and efficient adaption is important. Recommender systems often become more precise when the user spends more time with the system. Second, the system's structure is often split up in a client and in a server side, where the client side solely gathers user information and sends it to the server, where the actual computation takes place.

User Adaptive Computer Games

User Adaptive computer games are games that focus on increasing the perceived value by providing a strongly individualized experience (Brisson, 2012). An example is a first-person shooter that adapts the behavior of the opponents according to the shooting accuracy of the player.

- The main *purpose* of the user model is to identify or classify the user, so the appropriate adaptation in the computer game can take place.
- The *data* that is used addresses the preferences and progress of the user, such as the user´s current difficulty level or even the employed strategy. This data is often gained through the interactions of the user with the game, and therefor first needs to be interpreted and formalized before it can be used to infer conclusions on a higher level.
- The *adaptation* that takes place in the game concerns changing the content and behavior of the game, such as the game difficulty, the behavior of non-player characters or even the background music (Bakkes, et al., 2012)

Due to the focus on the user, user adaptive computer games have relatively a lot of processing power available for personalization. This makes the user adaptive computer games domain a very interesting research domain. Large amounts of (raw) data can be recorded and interpreted, resulting in the possibility for complex adaptation and a large range of personalization possibilities.

2.2.3 User models for educational purposes

Educational systems are systems developed with a teaching purpose. They are broadly applied in e-learning, where electronic media and ICT technologies are used for education. However, in most educational systems, user modeling and adaptation plays a minor role. Content is simply presented, and only simple things such as the student's progress in the course are registered. By adding (more) personalization to these applications, the learning value can greatly increase, ensuring that every learner achieves and reaches the highest standards possible (Heller, et al., 2006). Also the experience of the teacher or supervisor can be enhanced through personalization, for example through inferring and employing the preferred teaching style. However, here we consider especially the student as the user to which the system will be personalized.

The data in user models for educational purposes generally concerns the student's current knowledge state and other individual characteristics, such as the student's learning style and preferences. The data in the user profile should also be accessible for humans, such as the student's supervisor. Therefore, it is preferable that the information saved in the user profile is interpretable by humans. When looking at the time of adaptation in educational systems, we can make a clear distinction between adaptation while the student is performing an exercise, which we will refer to as online adaptation, and adaptation that takes place afterwards, which we will refer to as offline adaptation.

The most important research domains that do use extensive user modeling constructions are **Intelligent Tutoring Systems** and **Adaptive Educational Games**.

Intelligent Tutoring Systems

Intelligent Tutoring Systems (ITS) are systems that provide students automated step-by-step guidance as the students perform training tasks and/or work on exercises. An ITS has the purpose to complement or even replace the human teacher. An example is a system for teaching students how to program, with the ability to automatically detect common mistakes (Elsom-Cook, 1993).

- The main *purpose* of the user model is to select educational activities and strategies, as well as to deliver individual feedback that is most relevant to the user's level of knowledge (Kobsa, 2001; McTear, 1993).
- The user *data* that is recorded for this purpose is the student's state, knowledge and level of achievement. This information is especially observed through the actions and results of the student,

such as the answers the student gives. After observing this information, it is used to infer higher level properties, such as the student´s learning style and other preferences.
- Using this information, the learning content and behavior of the system towards the student is *adjusted*, for example by presenting easier exercises or specific tips.

Traditionally, just like decision support systems, ITSs are knowledge based systems, using formalized domain knowledge and rules to drive the user adaptation. So-called stereotypes are widely used in ITSs (Kay, 2000) and represent a collection of default attributes that often co-occur in people or in a certain group of people. The different stereotypes that have been developed differ in granularity of detail and complexity. The user model can use these stereotypes to make a large number of plausible inferences on the basis of a substantially smaller number of user observations. This means that in ITS, categorizing the user in a stereotype and maintaining the consistency of the user data, are also important roles of the user model.

Adaptive Educational Games

Adaptive Educational Games (AEGs) are complex educational games that combine ideas from several research areas, to enhance the student's learning experience (Peeters, et al., 2012a). AEGs are especially based on serious games: computer games with an educational purpose, where things are taught to students by using a playful approach (Korteling, et al., 2011; Johnson, et al., 2005). An example of an AEG is a training application for fire fighters, letting the fire fighters train their skills and knowledge in a safe, virtual environment.

- The *purpose* of the user model in an AEG is to optimize the learning process and outcome.
- The user *data* is considered with the progress and knowledge of the student, but also with the student's mental and cognitive characteristics. To construct this high level student information, the student's behavior and performance in the application is tightly monitored. In some cases additional physical data is gathered, like the student's heartbeat and skin conductance.
- The gained information can be used to *adapt* the content, presentation, and system behavior to the student's need, for example, by adjusting the content, tone, or amount of presented feedback.

Just like in adaptive computer games, AEGs have a lot of processing power available for personalization, making AEGs a complex and interesting domain for user modeling. In Chapter 6, we will further elaborate on AEGs.

2.3 Methods for user modeling

In addition to the research that was done to develop user models suitable for the application domains discussed above, researchers have also proposed more general design methods and frameworks to guide the developer in the process of user modeling. These general methods are useful in research projects, where the knowledge can be reused to tailor the user model to the system's needs. Also in commercial applications, these general methods have proven to be useful (Brun, et al., 2010), because they make it easier and more feasible to implement personalization into a system.

In early work, the process of user modeling was mostly based on the intuition and experience of the developer or researcher. As the user modeling research field evolved, there has been put much effort in creating a general way for designing and constructing a user model, by basing decisions on more empirical grounds and by defining methods applicable to the whole field (Kobsa, 2001; Durrani, 1997). Frameworks, methodologies and architectures have been developed, defining strict steps, restrictions and choices on how to design and construct a user model.

In the early days of user modeling the focus was put on developing one method applicable to the user modeling field as a whole. However, as mentioned earlier, user modeling is a very cross-disciplinary research topic. Therefore, throughout the decades, the user modeling research field has been influenced by the important research topics and trends of their time. For instance, when ITS became a major topic in the early nineties, user modeling methods were also mostly focused on the application of stereotypes, knowledge

bases, and logic to define a user model. With the rise of the Internet, the focus of the user modeling field shifted to web oriented applications and all the specific problems that arise herewith. Due to this connection, also the general user modeling methods that were developed, were focused on the popular research domains of their time (Kay, et al., 2012). The general ideas behind user modeling did not really change, but the specific filling-in of the user model, such as which technology to apply, did change.

Because of this ongoing development of user modeling as a whole, most researchers eventually agreed that one method to solve all problems is not possible (McTear, 1993; Kobsa, 2001). Instead, a wide variety of generic user modeling methods has been developed (Fischer, 2001); each of which supports only a few of the very different manifestations of personalization. In the rest of this section, the general user modeling architecture and the most interesting general and domain specific methods will be shortly discussed.

2.3.1 General architecture

The general architecture of a user model especially captures the software development requirements identified in Section 2.1. Figure 2.1 shows this general architecture. The user model is defined as an isolated component of the system, performing tasks when indicated by the other system components (McTear, 1993). Within the whole system, the user model is often seen as a black box, meaning that only the input and the output should correspond to the requirements of the other components. The dialogue manager or graphical user interface is responsible for the interactions with the user, and fuels the user model with data about this user. The user modeling component itself consists of a management component, a maintenance component, and a user profile. The management component is responsible for the construction of the user model, utilizing the user information and supplying the other components of the system with relevant information. The maintenance component is responsible for processing and updating the user information in the model. The user profile is a database that contains the actual user data. Instead of a single user profile, the user modeling component can also contain a database of user profiles. In this general architecture, many details should be filled in by the developer, such as how to acquire and how to represent the necessary user information.

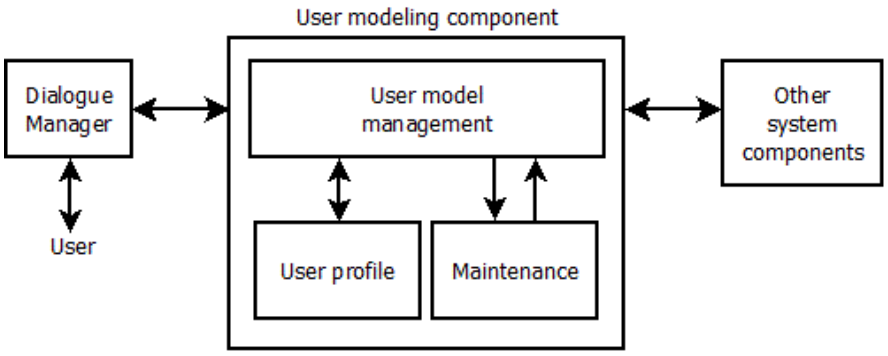

Figure 2.1: The general user model architecture, adapted from (McTear, 1993)

For specific domains such as ITS (Yang, 2010; Nwana, 1990), hypermedia systems (Deepa, et al., 2012), web-based systems (Razmerita, et al., 2003), and ubiquitous systems (Specht, et al., 2006), more precise user model architectures have been developed, all originating from this abstract model.

2.3.2 General methods

Throughout the literature multiple methodologies for user modeling have been constructed. An early, but interesting user modeling methodology was constructed by (McTear, 1993). McTear identifies several problems that need to be solved for creating a user model:

- Select the user characteristics you want to use in your model
- Determine the way in which these characteristics can be represented and measured or observed
- Determine the way in which this user information can be maintained

- Determine the way in which the personalization circle works, considering the needed functionalities and desired inputs and outputs
- Integrate the user model with the rest of the system components
- Determine how to validate the user model

In his general summary of user modeling, McTear also identifies several important choices one has to make. The user model is either:

- *individual* (made for an individual user) or *canonical* (made for a class of users)
- *static* (not possible to alter) or *dynamic* (altered during the course of interaction)
- *short term* (discarded at the end of a session) or *long term* (maintained for future usage)
- *explicit* (user data is gained directly) or *implicit* (user data is gained indirectly)

These terms can be used to classify the user model and thus simplified the user modeling process. However, when looking at all the different properties of user models we saw in Section 2.2, this classification hardly captures all the different possible aspects. Additionally, the user modeling steps McTear identifies are broad, yet not specific enough to appropriately guide the developer.

Another interesting and more abstract general methodology is composed by (Nurmi, et al., 2007). In this methodology, the whole process from constructing, to evaluating the user model is captured.

1) Choose the domain and perform an analysis
   I. Define the requirements: analyze the technical requirements, the tasks that need to be performed, and how a user would perform these
   II. Preliminary Evaluation: verify that the system performs the tasks that it is supposed to do
2) Create the user model:
   I. Identify the users
   II. Collect data about the users, which can be done:
      - Manually, by observing and building a cognitive model of the users
      - Automated, by using machine learning techniques to learn the model
3) Apply the model and perform adaptation
4) Evaluate the user model
   - Expert evaluation, by performing a usability and performance analysis
   - Manual evaluation, by letting the users give feedback

This methodology might be all-comprehending, but is also far too general for direct application. Again, compared to the overview of user modeling given in Section 2.2, many general user modeling steps and problems are left undefined. It is useful as a coordinating plan, but not for dealing with the specific subtleties of individual user models.

2.3.3 Domain specific methods

In addition to the general methods for creating user models, there has been research in each individual research domain for creating a more concrete method. For instance in ITS (Elsom-Cook, 1993) and web-based systems (Razmerita, 2011). Each method and framework attempts to catch the data flow, choices and problems an application in the domain should take into account.

An interesting user modeling survey has been performed by (Brun, et al., 2010), looking closely at web based systems and especially to recommender systems. In their pursuit to classify and compare different user models in this domain, the authors created a user modeling topic map. Figure 2.2 shows an overview of the topic map. A distinction is made between four different components of user modeling:

1. Input, representing the characteristics of the data used to build the user model

2. Accessibility, representing the way the model can be interpreted and what it represents
3. Type, representing the user model features that are dependent of the whole system
4. Approach, the general characteristics that should be chosen when building a model

Although the emphasis lies on recommender systems, there are many terms that are applicable to the user modeling area as a whole. Unfortunately, no specific instructions or approaches are introduced to use the topic map in the process of user modeling itself, which in the article is discussed as future work.

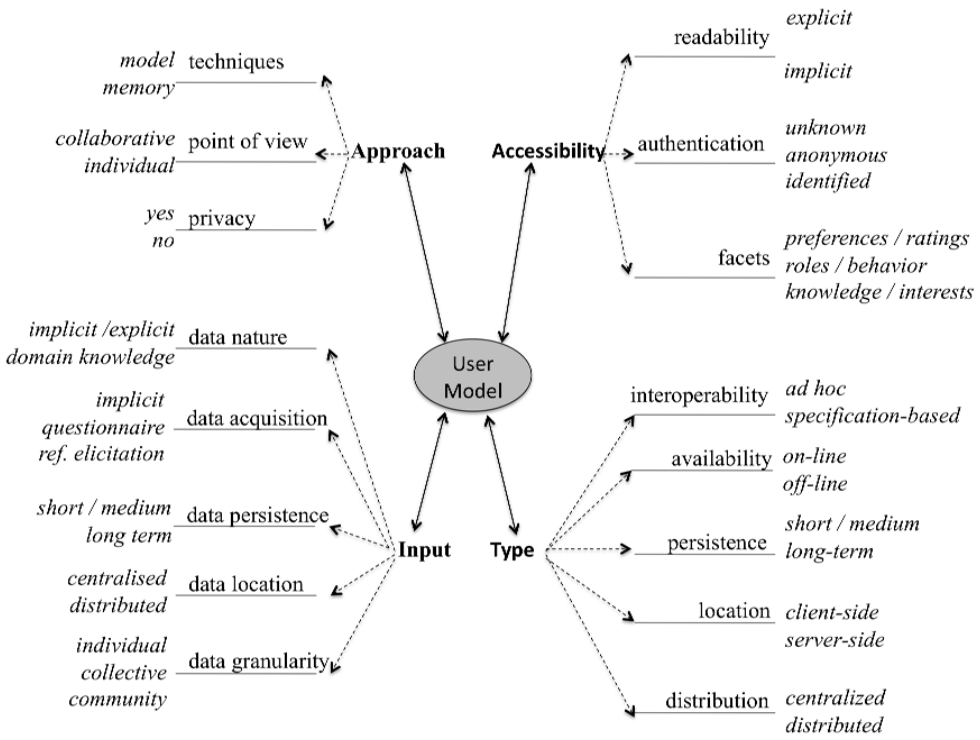

Figure 2.2: A topic map for user modeling in recommender systems, based on (Brun, et al., 2010)

Another, very specific user modeling framework is created by (Sannes, 2011). Focus is put on user models for performing task support. An overview of the framework can be found in Figure 2.3, and consists of the following components:

1. System: the whole application, which interacts with both the user model and the user;
2. Models: pieces of logic that convert data into a higher level of abstraction, such as task models and cognitive models;
3. Current User Context: registers the raw data about the current situation of the user;
4. Current System Context: registers the raw data about the current situation of the system;
5. Current User State: the current information available about the user, such as the emotions the user is feeling or the user's stress level;
6. History: a log of all occurred user states;
7. User Fact Base: all knowledge about the user that holds for a longer period of time, such as the user's preferences, skills, and knowledge;
8. Query Engine: the gateway of all the user knowledge stored in the framework, for both the system as the user model itself;
9. User State Determiner: responsible for determining the current state of the user at the current point in time. To infer this state, models (2), the previous user information (6, 7), and the current user information (5) is requested through the query engine;

10. Learning Engine: responsible for learning facts about the user, over a longer period of time

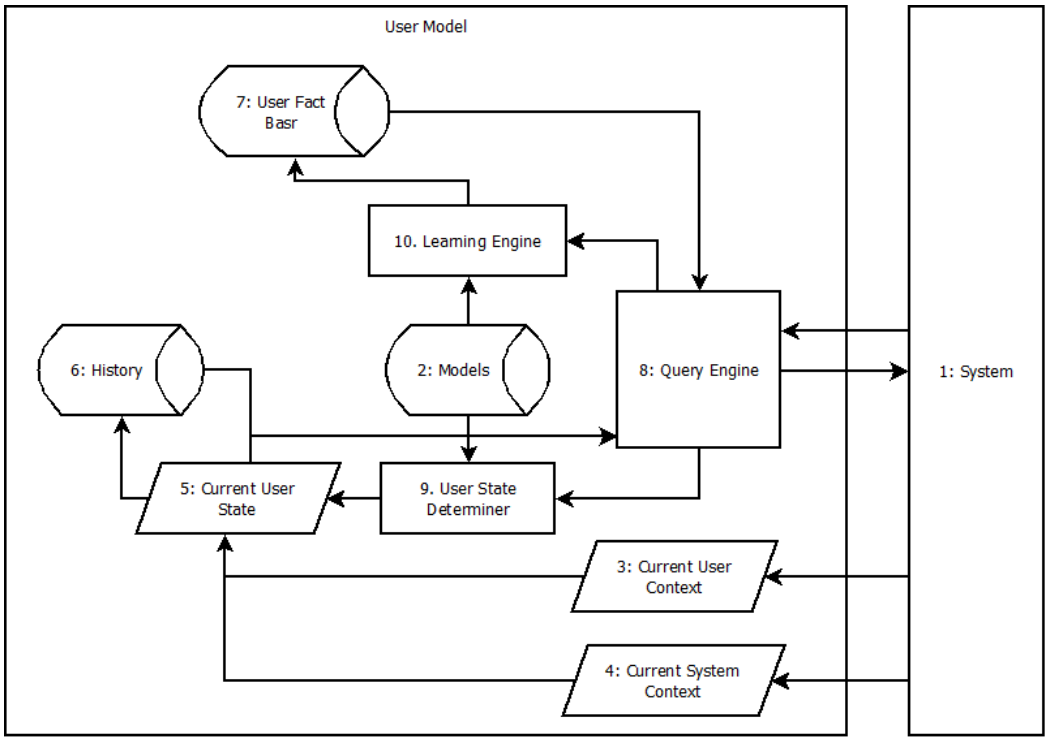

Figure 2.3: A user modeling framework for task support systems, based on (Sannes, 2011)

This very elaborate architecture for user modeling captures many interesting concepts and relations between these concepts. However, when looking at the user modeling domain in general, many components are not necessary or can be depicted in a less complex manner. Additionally, hardly any information is provided on how to design a user model with this framework, making the framework not directly applicable.

2.4 Conclusion

In this chapter, a review has been given of the user modeling field. The general technical requirements have been defined, an overview of the current important user modeling application domains has been given and multiple methods for user modeling have been discussed.

User modeling is not a straightforward process. Each application domain has its specific requirements and demands on the user model, especially differing in the general task it should perform, the kind of data that is used, and the way how the adaptation takes place. When looking at the technical requirements of a user model, ideally it should be as modular as possible, enabling decomposition, modifiability and reusability. Depending on the actual user modeling domain, it is not always possible to obtain all the desired technical features in the resulting model. Multiple methodologies, frameworks and architectures have been developed to help the developer with the process of user modeling. There are a few general user modeling methods for capturing user modeling as a whole, and a multitude of user modeling methods focused on a specific application domain or problem. However, the high level methods are often too unspecific to capture many of the important user modeling aspects, while the domain specific methods are often too detailed and narrow, which makes it hard to apply them for more general user modeling problems.

Concluding, according to this review there is no general method to bridge the gap between the domain specific methods and the (too) general methods. Therefore, in the next chapter, the findings presented in this chapter will be used to construct a more abstract overview of user modeling, resulting in a general user modeling methodology.

Chapter 3

General User Modeling Methodology

When adding personalization to a system with the use of a user model, many general and application domain specific problems must be solved. As was concluded in Chapter 2, a multitude of methods have been proposed to help developers with the process of user modeling, but the major flaw of these methods is that they are either far too broad, and therefore too unspecific, or far too (domain) specific, and therefore hard to apply to more general problems. Additionally, existing methods are often dependent on the major research topics of their time, which makes it hard to use them for present-day problems, let alone future problems.

To bridge the gap between unspecific and specific user modeling methods, this chapter proposes a general user modeling methodology. This methodology consists of three phases: the conceptual, functional and technical design phase. The conceptual design is concerned with the general purpose and behavior of the user model, whereas the technical design is concerned with the actual implementation. In general, the more analytical, broad, and unspecific user modeling methods are focused on the conceptual design, whereas the domain specific methods are focused on the technical design phase. The functional design phase that is introduced in between is concerned with explicating the intermediate steps between the high-level conceptual design and the low-level technical design. The resulting method can be used to couple the user modeling problems that need to be solved, to the possible technical solutions. To ensure that this methodology is also useable for future research domains, the aim is not to solve every possible generic and domain-specific problem that can arise, but to formalize the general questions and an extendible list of possible answers that should lead to a correct and optimal user model.

To construct the different phases of the methodology, the information discussed in Chapter 2 will be used. The conceptual design will be constructed according to the existing general user modeling literature, as discussed in Section 2.3. However, in order to construct the functional and technical design, first the underlying user modeling properties making up these design phases must be further refined and discussed. Therefore, in Section 3.1, the user modeling application domain exposition as presented in Section 2.2 will be used to abstract the general functional components, functional properties and technical properties applicable to user modeling, resulting in the possible requirements and options for implementing a user model. In Section 3.2, the complete general user modeling methodology will be presented, in addition to guidelines for using the methodology.

3.1 General user modeling properties

Before the methodology can be constructed, first the user modeling domain must be depicted in a more abstract and general way. When developing a user model, the operations the user model has to perform and their associated properties are mostly determined by the application domain itself. However, when taking a closer look at the overview of the user modeling application domains as set out in Section 2.2, one can conclude that there are also a multitude of more abstract and independent overlapping operations and properties. As discussed in Section 2.3.1, one of the strict distinctions that can be made is between the data used in the user model to actually apply the personalization (i.e. the user profile), and the operations that are necessary to perform the personalization of the system. This distinction will be used throughout this chapter to structure the discussion.

In this section three general user model elements are identified and discussed. Subsection 3.1.1 describes the general personalization cycle that defines the functional components that make up the user model. Subsection 3.1.2 describes the abstract functional properties that can be related to the functional components. These properties are concerned with the desired behavior of the user model and further refine

the problems that have to be addressed and solved. Subsection3.1. 3 describes the technical properties, concerned with the actual options that are available for implementing the desired behavior in the user model.

The formalization of these functional components and properties is especially influenced by the topic map constructed by (Brun, et al., 2010), previously described in Section 2.3.3. To keep track of the different functional and technical properties and their possible values, an overview can be found in Appendix A.

3.1.1 Personalization Cycle

When looking at the operations that are necessary in the automated process of personalization, an adaptation cycle can be identified (Sannes, 2011; Elsom-Cook, 1993). The three steps of this cycle are: 1) observe the user 2) predict the user 3) adapt the system. A similar cycle can be distinguished in the broader domain of adaptive systems (Feigh, et al., 2012). The main difference is that in adaptive systems, observations and adaptations that do not affect the user are also taken into account. A visual overview of the personalization cycle can be found in Figure 3.1.

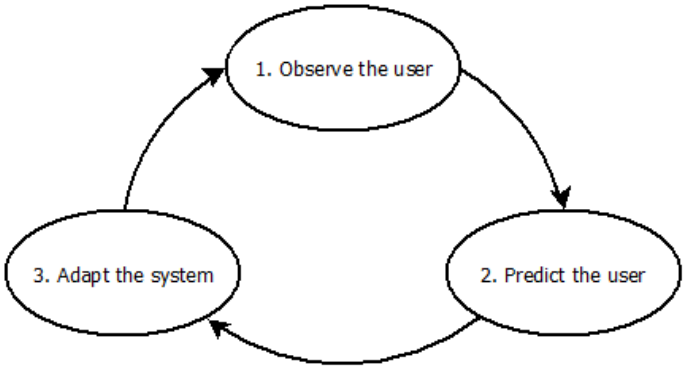

Figure 3.1: User model personalization cycle

This cycle will be used in the next section to discuss the necessary general functional components and their associated properties related to user modeling. The different steps in the cycle are:

1. **Observing the user**: observations of the user are the core of the personalization provided by the user model. This step is concerned with obtaining data about the user for constructing the user profile and for keeping it up to date. Observations may also be required that do not directly include the user, but address the context of the user at that time.
2. **Predicting the user**: once the user has been observed, the data can be used to construct assumptions about the user. These assumptions can be very broad; they might involve the identification of products in a web shop the user might like, but they can also say something about the current process of the user, or about the expected responses the user will exhibit when an adaptation is made. Due to the assumptive nature of these pieces of information, we call this step the *prediction* of the user. The prediction step can be divided into two interrelated steps: *processing user data* and *reasoning with user data*.
    i. **Processing user data**: it is not self-evident that the (raw) data that results from observing the user can be used immediately. In some cases, the data might first need to be processed before it can be used by the reasoning component. This processing step consists of two main sub steps: *amending* the incoming user data and *maintaining* the existing user data. Amending the incoming user data refers to the formalization and normalization of the incoming user data. Maintaining the existing user data refers to making sure that the user profile stays consistent and up to date (Fischer, 2001; Kobsa, 2001).
    ii. **Reasoning with user data**: after the data of the user has been processed, it is ready to be used for the creation of assumptions about the user. Whereas the processing step is more concerned with straightforward actions and procedures, the reasoning step is concerned with constructing

completely new information. Different kinds of reasoning techniques can be used to construct this new information.

An example of the operations that can take place in the prediction step can be found in an application that monitors the user's heart rate. The raw data obtained from the heartbeat sensor first needs to be corrected to delete possible noise (amend by normalize). Next, the average is calculated, and the old value in the user profile is replaced by the new value (maintain). Finally, this value is extrapolated by the reasoning component to infer that the stress level of the user exceeds the upper limit (reason). Hence, it is decided that an adaptation is required.

3. **Adapting the system**: finally, the system is adapted to the assumed needs of the user. This step entails the presentation of the adaptation to the user and communicating this adaptation to the rest of the system. In most systems, there is only one way to adapt the system. For example in recommender systems, where only the content of the recommended objects can be changed. In more complex systems such as adaptive educational games, the adaptation can take place in different ways, for example by changing the overall difficulty or changing the provided feedback. This means that in the prediction step, also *what* to adapt should be determined.

The specific interpretation of each step in the personalization cycle depends on the application domain and the employed technology. Additionally, the different steps in the personalization cycle can overlap with, and ensue from, each other. An example of this cycle can be found in a computer game that adapts its difficulty level to the shooting accuracy of the user:

1. The user's behavior and results are observed, like the amount of shots fired and enemies hit
2. According to the user data, the shooting accuracy of the user is calculated and used to infer the proper difficulty level of the game
3. The difficulty level of the enemies is changed according to the newly inferred difficulty level

An example of this cycle in a less complicated system is a recommender system that recommends books to a user in a web shop:

1. The behavior of the user is observed, like the books the user views and/or buys
2. The observations are used to infer which other books the user might like
3. The application is adapted, by showing the inferred books to the user

3.1.2 Functional user modeling properties

As can be inferred from the personalization cycle, a user model consists of several functional components, where each component is responsible for supporting a step in the cycle, i.e. observe the user, predict the user's behavior, and adapt the system to the user's (future) needs. In this section these functional components and their associated concepts and properties are discussed. Additionally, the general functional properties related to the user model as a whole and the user profile are discussed.

Observing the user

The first step involves the reception of new information, i.e. observations about the user. When looking at the functional component (or functionality) that is responsible for observing the user, there are three properties that can be distinguished: the type of observation, the level of intrusiveness of the observation, and the reliability of the resulting data.

Type of observation. Most observations done in a user model are directly related to the user. Often, there is additional information needed to further interpret the observations related to the user. For instance, the state of the environment. In addition to user-based observations, there exist system-based, environment-based, task-based, and spatiotemporal observations (Feigh, et al., 2012).

Intrusiveness. The intrusiveness of the observation is concerned with the amount of intrusion the observation inflicts upon the user. For instance, in educational applications the intrusiveness should often be very low, whereas in applications such as recommender systems it is the other way around.

Reliability. The reliability of the resulting data is concerned with the importance of the truthfulness of the observed data.

Predict the user

The prediction step of the personalization cycle induces the need for a functional component that contains one or more processing and reasoning functionalities that are able to predict the desired information from the user. To process and predict this information, a conceptual model from the human factors literature is normally used, for instance, to interpret the user behavior. Additionally, the predicted data can be versatile, which means that the functional component for predicting the user is often the most complex and important part of the user model.

The necessity of additional processing functionalities for amending and maintaining the data depends on the reasoning functionality. There are many ways in which the reasoning functionality can be filled in. Independent of the method that is used, there are three important functional properties that can be abstracted for this step: granularity, accuracy, and trigger type.

Granularity. Granularity refers to the level of detail of the user model. The data in the user model could be optimized for a single user or it could be defined for a (sub) group of people, such as students with a certain knowledge level.

Accuracy. Another property that can be distinguished is the amount of data required before the output produced by the reasoning functionality becomes accurate.

Trigger type. The most important type of trigger is based on the user data itself, which can be obtained either by observing the user, or by inspecting the data already available in the user model. Besides the user data, the different types of observations done in the observation step, such as changes in the environment, can also trigger the reasoning functionality to adapt the system.

Adapt the system to the user's need

The functional component that is responsible for adapting the system to the user's needs is concerned with the way how to use and present the newly obtained assumptions about the user in the system. There are two properties associated with this functionality: the form and the intrusiveness of the adaptation.

Form of adaptation. The resulting adaptation can be done in four general ways (Razmerita, 2009; Feigh, et al., 2012). Multiple types of adaptations can take place in the same system:

1. Adjust the application content: dynamically tailor the presented information to the user, such as in recommender systems;
2. Adjust the application structure: change the information structure to the user. For example through highlighting, hiding, or sorting certain pieces of information. Especially adaptive hypermedia systems use this type of adaptation;
3. Adjust the data presentation: change the way in which information is presented to the user. For instance, through adjusting the layouts, skins, and fonts of an interface. The information modality can also be changed, which is concerned with changing the information medium, while still expressing the same content. This type of adaptation is often used in task support systems;
4. Adjust the application configuration: change the behavior of the system to the needs of the user. Especially in complex systems like Adaptive Computer Games, these adaptations can be used to adapt

the non-visible things of the systems, such as the general difficulty. Taking over tasks of the user also belongs to this kind of adaptation.

Intrusiveness. The intrusiveness of the adaptation is concerned with the amount of intrusion the adaptation of the system inflicts upon the user.

Overall user model functional properties

When considering the user model as a whole, there are two general properties that can be distinguished that are independent of the different functional components: the operability and the incentive of the user model.

Operability. The user model can be used *online*, during the interactions the user performs while using the system, or *offline*, where the gathered user information is used afterwards to adjust the system. A combination of online and offline is also an option.

Incentive. The initiative of the personalization cycle can be *passive*, where the user model only performs adaptations on designated moments or when other components of the system ask for it. It can also be done in an *active* way, which means that the user model adapts to the user according to its own instigation. Again, a distinction can be made between two types: *reactive* and *proactive* behavior. *Reactive* behavior means that the user model initiates adaptation when certain observations are made. *Proactive* behavior means that the user model actively searches for the need of adaptation whenever it feels necessary.

User profile functional properties

Both the user data and the functional components define the user model, therefore the general properties associated with the two components are tightly intertwined. Most of the general properties that define the user data can be inferred from the different functional components, such as the data's reliability. When looking at the general properties about the data in the user profile, four additional properties can be abstracted: the user data meaning, privacy, user data readability, and user profile persistence.

Expressiveness. Each data entry in the user profile represents a certain user characteristic, depending on the purpose of the user model. The data can be behavior related, like the actions performed in the interface, task related, like the competence level concerning a certain skill, or it can be related to giving the user a personalized experience, like the user's previously viewed products. The term *expressiveness* (Webb, et al., 2001) expresses the requirement of a user model to display and save all these different types and kinds of user data.

Privacy. When considering whether or not to save data about a user, privacy always plays an important role. For instance, in web-based applications extra emphasis is placed on ensuring that the information is kept *private*, whereas in educational systems, the user data should be *public*, or *partly public* in order for people such as the tutor to see it.

User data readability. In some applications, it is important that the data is *readable* by a human, possibly after some data processing. When the data is *readable*, it means that the human can read and understand the data; he can know what the characteristics of the user are. When the data is *not readable*, a human cannot interpret the model and cannot deduce the characteristics of the user. For example, in educational applications, the data about the student's progression with the learning material should be readable for the human tutor, as this enables the tutor to anticipate the students' competences beyond the context of the application. In other applications, such as recommender systems, the readability of the user data is less important, because the data is not directly used by a human being.

User profile persistence. The total lifespan of the user model as a whole. User models on the Internet often last as long as the user is visiting the website, whereas it is necessary in other domains such as educational application to save the user profile as long as the user uses the application.

3.1.3 Technical user modeling properties

Whereas the functional properties define the functional components and their possible desired properties, the technical properties are concerned with the actual options that are available for implementing the user model. As was the case for the functional properties, the technical possibilities of implementing a user model can also be presented in a more abstract way. This section provides a concise classification of how the different functional components can be technically realized.

Observing the user

As inferred from the general personalization cycle, there is a need for a functional component that can observe the user and takes into account the intrusiveness, reliability, and type of observation. When implementing this functional component, the literature on user modeling distinguishes several major technical solutions. Each technical solution can be categorized according to the **origin** of the data - the data is either obtained directly or indirectly from the user, and according to the data **format -** the obtained data is either raw or (partially) interpreted. The three major technical solutions for obtaining user information are:

1. Through **user interactions** with the system. For instance, by monitoring the amount of correct answers in an educational system. In this method, data is obtained *indirectly* from the user. The resulting data is *raw*, thus normally needs additional interpretation. The observed interactions do provide an objective assessment of the user's actions.
2. Through **direct input** from the user. For example, by the product ratings the user gives in a recommender system. This method provides a *direct* way of obtaining user information and results in information that is already (partially) *interpreted*.
3. Through the use of (external) **sensors**. For example by monitoring the user's heartbeat. These methods especially provide an assessment of the user's physical state. This way of obtaining user data is commonly known as psychophysiological measurements (Grootjen, et al., 2006). Again, the data is obtained in a very *indirect* way and the resulting data is *raw,* thus often needs additional interpretation.

Predicting the user

The functional component for predicting the user is responsible for inferring a wide range of desired information. It consists of functionalities that are able to process the user data and to reason about this user data. Moreover, it takes into account the granularity and accuracy of the prediction model, and the type of trigger to use in the prediction model. When implementing this functional component, the most important and complex task is to implement the functionalities capable of reasoning with the user data. Within the user modeling domain, three main approaches can be distinguished:

1. The **rule-based approach**. In this approach, the reasoning functionality consists of a knowledge base of predefined facts and inference rules. The inference rules are used to deduce new facts and are triggered by the observed and interpreted user data (Kobsa, 2001). Common techniques using a rule-based approach for reasoning are logic-based systems, stereotypes and decision trees (Fischer, 2001; Solinger, et al., 2005). The rules and initial knowledge are explicitly predefined by the developers of the user model or by domain experts. Due to the declarative and static nature of using predefined rules to predict the user, rule-based systems are often not very dynamic and thus restricted in complex and uncertain environments. This approach is commonly used in knowledge-based systems such as intelligent tutoring systems and decision support systems.
2. The **computational approach**. With this approach, according to the user data obtained from the individual user or from a community of (similar) users, correlations, patterns, and connections can be

discovered and a model for prediction can be trained. For instance, through analyzing the behavior of costumers in a web shop, the correlation between products can be discovered. Within this approach, techniques are used from the machine learning and data mining (statistical methods) domain, such as (un)supervised learning, graphical models and Bayesian networks (Webb, et al., 2001; Solinger, et al., 2005). The resulting prediction functionalities are especially useful for handling uncertain or incomplete user information. This approach is widely used in recommender systems and adaptive hypermedia systems.

3. The **(cognitive) model approach**. This approach uses (cognitive) models to understand and predict the user's behavior, such as the user's goals. These models are based on different theories and methods from psychology and cognitive science, such as the Beliefs-Desires-Intentions (BDI) paradigm and the SOAR architecture (Heuvelink, 2009). This approach is used in adaptive computer games and adaptive educational games.

The different approaches can also be used to complement and strengthen each other.

Adapt the system to the user's need

The last functional component is responsible for adapting the system while taking into account the form and intrusiveness of the adaptation. When multiple ways of adaptation are possible, the best way of adaptation is determined by the prediction functionality. The implementation of the components that can be adapted is not a part of the design process of the user model itself: choices regarding the implementation of these components depend on the structure of the system as a whole. As such, this functional component of the user model (adapting the system) is only responsible for informing the other components to adapt in the way that was inferred during the prediction step.

Overall user model technical properties

The technical properties of the *overall* user model mainly depend on the technical properties of the system as a whole. The properties that can be distinguished are: the user model location and the available computational power.

Location. The architecture of the complete system determines the location of the user model components. The common approach is to centralize the user model, which permits multiple uses of the user model by different components in the system. Another possibility is to use a distributed approach, in which the components are spread throughout the application. In this case, the functionalities can be placed at the client – or at the server-side. When looking at network based systems such as web applications and adaptive ubiquitous systems, the user model is often distributed.

Computational power. The overall system also determines the amount of available computational power for the user model.

User profile technical properties

The technical properties that can be associated with the *data* that is used in the user model (the user profile) are mostly determined by the implementation of the functional components responsible for observing and predicting the user. The additional properties that are relevant for the user model as a whole are: the data type and the data storage location.

Data type. The data itself is described in very concrete data types and data structures, such as integers, Booleans, lists, or strings. Each data entry can also be categorized according to its persistence, which can range between static, such as the language of the user, and temporal, like the user's average heartbeat.

Storage location. Additionally, just like the location of the user model functionalities in the complete system, also the storage location of the user data can be distinguished. Again, either a central spot is used

to keep the database, or multiple spots are used to keep the different parts of the user database (Karam, et al., 2012). When considering a distributed approach, a distinction can be made between storing user information at the client or at the server side of an application.

3.2 General User Modeling Methodology

In this section, the general user modeling methodology is proposed. This methodology is meant to bridge the gap between broad conceptual and domain-specific user modeling methods. It is especially focused on the different questions that need to be answered, in order to map the general ideas behind personalization (Section 2.3), to the necessary functional components and their desired properties (Section 3.1.2), and to the technical possibilities for implementing a user model (Section 3.1.3). Combined, they should guide the development of a correct and effective user model. To keep track of the definitions defined in the previous section and reused in this section, a summary of the properties associated with each functional component can be found in Appendix A.

Before the methodology can be applied, certain information about the system needs to be available. When developing a user model, the most important things to look into are the application domain and, if applicable, the existing system. Therefore, a domain and task analysis is essential before employing the user modeling methodology (Kobsa, 2001). The design process of the user model itself is split up into three main phases: the conceptual, the functional and the technical design phase. These different main steps of the methodology are visualized in Figure 3.2. Each phase builds upon the choices made in the previous phase, and adds more details and constraints to the resulting user model. To illustrate each phase, an example about an educational application for teaching mathematics is used at the end of each section. After describing and explaining the methodology, several guidelines are given to employ the methodology.

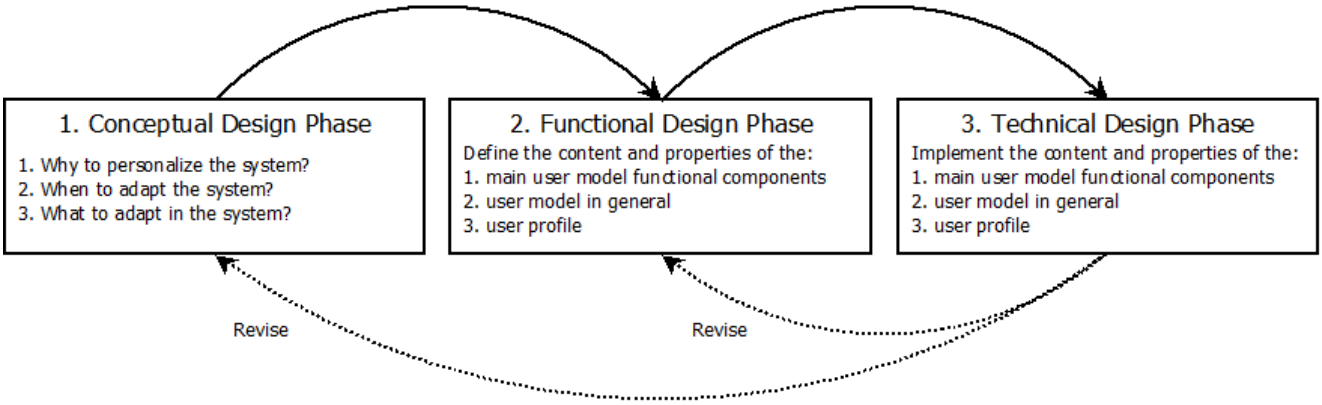

Figure 3.2: visualization of the general user modeling methodology

3.2.1 Conceptual Design

Any user model requires a conceptual design that captures the purpose and requirements of the desired user model, independent of the implementation concerns. In this phase the general questions are asked that need to be answered when adding personalization to an application. The three main questions are:

1. Why will there be adaptations? This question is concerned with the reason of adding personalization to the application. What is the overall goal of the application, and what is the utility and reason of adding personalization to the application?
2. When will the system be adapted? In order to determine when to adapt, the developer should look at the conditions for adaptation, meaning the abstract observations that are necessary to *trigger* this adaptation. For instance, certain behavior displayed by the user, or a predefined time interval
3. What will be adapted? This step is concerned with the concrete application specific components that should be changed, such as the content or the configuration of the application

These different concepts are related to problems in the Human-Computer Interaction domain: you want to adapt the system to the user, but the question remains how "to say the 'right' thing at the 'right' time in the 'right' way" (Fischer, 2001). Depending on the desired system, a theory or method from the human factors literature is often used to make sure that the adaptation that should take place makes sense, and thus is effective. Most of the information necessary for these steps should already have been formalized through the prior task and domain analysis. Often, it is sensible to define these concepts consecutively. Determining ***why*** to adapt in the application is the main incentive to the answer of the other two questions. When you know ***when*** you want to adapt, you will have to find out ***what*** to adapt to achieve the personalization goal.

Example: An educational application in mathematics

When looking at the example about an educational system in mathematics, the steps of the methodology can be filled in as follows:

1. Why will there be adaptation? The overall goal of the system is to efficiently teach a student mathematics. We want to personalize the application to make the learning more challenging and dynamic.
2. When will the system be adapted? The application should adapt when it detects that the student's competence level improved or that the student is having difficulties with the current level. The general trigger for this adaptation is the user's performance, expressed in the amount of mistakes.
3. What will be adapted? The application should adapt itself to the competence level of the student, thus change the difficulty level of the exercises that are presented to the student.

3.2.2 Functional Design

In the functional design, the conceptual definitions of *why*, *when* and *what* to adapt are used to fill in the details and desired properties of the three main functional components, the overall behavior of the user model, and the user profile. In this way, the ideas and the purpose behind the user model are made more concrete.

Main user model functional components

The three main functional components are responsible for: observing the user, predicting the user, and adapting the system. There is a direct mapping between these functional components and the answers why, when and what to adapt the system. However, a developer should take into account that filling in the details of these components is a flexible process, in which the different questions and components influence each other. The mapping can be done as follows:

1. **Predicting the user**. The question **why** to adapt the system results in the necessity and content of the functional component responsible for predicting the user. Personalization should result in the improvement of the system, and to adapt the system in the right way, the user should be correctly predicted. The desired properties that can be used to further refine this component are the *granularity* of the user model, the *accuracy* and the *type of trigger* that should be used.
2. **Observing the user**. The question **when** to adapt the system determines the content of the functional component responsible for observing the user. Depending on the identified general triggers, the *type of observation* can be determined. Additionally, it is important to determine to which extent the observations should be *intrusive* and how important the *reliability* of the resulting data is.
3. **Adapting the system.** The question **what** to adapt in the system determines the content of the functional component responsible for adapting the system. Within this functionality, the *intrusiveness* and the *type of adaptation* should be determined.

For each functional component, it might be necessary to define the sub-operations that lead to the goal of the component.

User model in general

Apart from the content and properties of the general functional components, the properties of the overall user model also need further refinement. The ideas behind **when** to adapt result in the need to determine the *incentive* and *operability* of the user model as a whole. With the definition of the incentive, operability, and the necessary functional components, a high-level user model deliberation cycle can be defined that serves as a blue print for the user model.

User profile

Finally, the user profile can be further defined by looking at the general purpose of the system and the user model. Herein, the *readability* of the data in the user profile, the persistence of the user profile, and the level of *privacy* of the user profile should be further refined.

Example: An educational application in mathematics

When looking again at the general example, the following content and properties can be filled in:

1. The resulting functional components and their properties:
   1.1. Predicting the user: considering granularity, the application is applicable to all students using the application and will not be optimized for the single user. The prediction should be as accurate as possible right away. The trigger used is solely user-based.
   1.2. Observing the user: in order for the student not to lose focus, the observation of data should be non-intrusive. The data obtained must also be as reliable as possible, due to the big influence of the adaptation to the system. The observation that is done, is user-based.
   1.3. Adapting the system: the configuration of the system will be adapted, resulting in the selection of different exercises. This adaptation should also be as non-intrusive as possible.

   No additional sub-operations are necessary for any of the functional components.
2. User model in general: the user should answer a set of exercises, after which the user model determines whether the difficulty should be adjusted. Thus, the user model adapts in a passive way, and the operability is solely offline. The relatively easy high-level deliberation cycle is visualized in Figure 3.3.

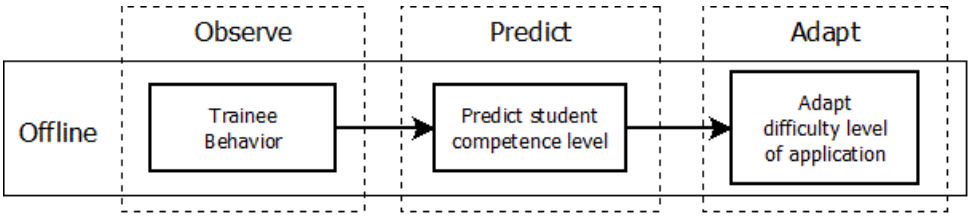

Figure 3.3: High-level user model deliberation cycle of an educational application in mathematics

3. User profile: the user profile should only be available to the student's teacher. For this reason, overall scores and difficulty levels should be readable. The user profile should be saved as long as the student is registered with the system.

3.2.3 Technical Design

The technical design is concerned with the actual implementation of the user model according to the defined concepts, functionalities and desired properties. The conceptual and functional design might have looked straightforward, but when working towards an implementation in the technical design, more delicate choices and problems arise. It is often not possible to exactly match the desired properties from the previous phases with the actual technical possibilities. This means that the technical design also influences the conceptual and functional design, forcing the developer to backtrack and adjust the previous decisions. Often, it is only possible to implement the user model by making payoffs between the desired properties and the actual possibilities.

Main user modeling functional components

The developer has to choose how to implement each separate functionality that was identified during the functional design. Depending on the different properties of the techniques that are chosen, these functionalities may overlap and intertwine. Again, a distinction can be made between the different main functional components, and their technical properties:

1. **Predicting the user**. To implement the prediction functionality, three main *approaches* can be used: the rule-based, cognitive model, or computational approach. Each approach handles specific problems in a different way, like how uncertain data is handled or how the model is constructed.
2. **Observing the user**. To implement the functionalities for observing the user, three main methods for observation are possible: user interactions, direct input, or sensors. When choosing a method, the *data origin* and the resulting *data format* of the method should be taken into account. Depending on the user characteristic that will be measured, it might be necessary to operationalize this characteristic. Operationalization involves defining a characteristic in such a way that it becomes usable in the system (Gazzaniga, et al., 2006).
3. **Adapting the system**. The functionality for adapting the system is especially concerned with communicating the desired adaptation to the adaptable components outside the user model. When multiple types of adaptation are necessary for the personalization goal, the predicting functionality is also responsible for choosing between these different types.

With the introduction of the method(s) and approach(es) to observe and predict the user, also additional operations for processing the user data might be necessary. After the definition of the conceptual, functional, and technical properties for each necessary (sub-) operation, their **general algorithm** can be defined. Each algorithm describes the steps that should be taken to reach the desired output, given a certain input.

User model in general

When considering the implementation of the user model as a whole, additional technical properties must be taken into account, influenced by the architecture and properties of the surrounding application. These technical properties are the *location* of the user model and the available *computational power*.

User profile

Finally, the technical properties of the data used in the user model can be determined. The details about the data mainly depend on the implementation choices of the separate functionalities. However, often the main *data types* and *data structures of the user model* can be further refined. Lastly, the *storage location* of the user profile must be determined.

With the complete definition of the functional components, the user model in general, and the user profile, the actual implementation can now take place.

Example: An educational application in mathematics

The user model of our example can be implemented in the following way:

1. The resulting technical properties of the different functional components:
   1.1. Predicting the user: a rule-based approach is used to predict the user. The simple rules are constructed by the developer himself.
   1.2. Observing the user: the user's progress, consisting of the mistakes made, is observed through the user's interactions in the system. This entails implicit and thus non-intrusive data gathering. The raw data is relatively simple; therefore extra processing is not needed.
   1.3. Adapting the system: when an adaptation is necessary, the system is notified that the presented exercises must have a lower or higher difficulty level.

   No additional operations or data processing is necessary. The general algorithm to predict the level of difficulty can be defined as follows:

Input: the user's progress, consisting of the mistakes made.
Output: the increase or decrease of the difficulty level
Algorithm: when the user made less than 5 mistakes, the presented exercises become harder. When more than 10 mistakes are made, the exercises become easier.

2. General user model: the user model is implemented with a centralized architecture. The location of the user model is at the same point as the rest of the application. The user model does not require much computational power, thus resources will not be a problem.
3. User profile: the user profile is also at the same location as the rest of the system. The user data is expressed in the integer data type.

The information formalized in the different steps can be combined to implement the user model. Due to the informative nature of the example, the complete implementation is not constructed.

3.2.4 Methodology Guidelines

Because of the different relations that exist between the design phases of the user modeling methodology, it might still be hard to build a coherent user model. For this reason, in this section several guidelines and tips are presented for using the methodology. Most of the information necessary for the conceptual and functional design phase has already been formalized through the task and domain analysis that was done before the user modeling began, making these two phases fairly intuitive. The technical design is a more complex process, in which the selected technologies are influenced by the previous choices, and might also redefine them. Therefore, this section places emphasis on the relationship between the different concepts and properties chosen in the first two phases and the actual implementation of these properties in the implementation phase. Again, the guidelines are presented separately for each functional component of the user model, for the model as a whole, and for the user profile.

Observing the user

When choosing the technique for implementing the observation functionality, the developer must take into account the desired *reliability* and *intrusiveness* of the data gaining method. Intrusiveness and reliability are often contradictory to each other. The most reliable way of gaining data is to obtain it directly from the user him/herself, but this also means that an intrusive method must be used, such as asking direct questions to the user. When the data gathering has to be done in a non-intrusive way, the obtained data often needs additional interpretation, in which case the reliability of the observation depends on the reliability of the used interpretation model. Thus, for both functional properties, the data *origin* property of the chosen observation technique is essential.

The *data origin* often influences the resulting *data format*. A direct way of obtaining user data normally results in (partially) interpreted data that is directly usable for predicting the user. This means that the operations in the prediction functionality can be easier, computationally less expensive, and faster. The raw data that results from data that is gathered indirectly generally demands more effort and resources.

Predicting the user

Implementing the prediction functionality is one of the most challenging parts of the user modeling process, because each separate reasoning technique handles specific problems, like problem solving or learning, in a different way. When choosing the main approach for predicting the user, the developer must take into account the desired *granularity* and *accuracy* of the reasoning component.

In general, when the granularity of the user model increases, also the complexity of the reasoning component increases. The *way of construction* influences the accuracy of the reasoning component. When it is important that the reasoning component becomes accurate instantly, it is often a good idea to use a reasoning component that must be predefined by the developer or by experts, like a rule-based approach, or a validated (cognitive) model. When the user model has more time to learn the preferences or behavior of a user, a computational approach is often fruitful. The effort the developer wants to put into the construction of the

reasoning component also must be taken into account when choosing the reasoning approach. For instance, when a validated model from the human factors literature is used to interpret the user's behavior, it normally costs less effort than when the method is defined and validated by the developer himself.

Adapting the system

The components that will be adapted are either already part of the system, or have to be created according to the specifications of the user model. Because these components are not a part of the user model itself, not many guidelines can be given for the implementation of this functionality in the user model. The most important task of the developer of the user model is to make sure that the different (resulting) functionalities communicate in the right way with each other and with the rest of the system. Therefore the desired inputs and outputs must be attuned to each other.

Overall user model

When implementing the user model as a whole, especially the technical properties of the complete application play an important role: the system's *architecture* and the available *computational power*.

The general system architecture influences the *storage location* of the user model. This location is normally centralized. The available computational power influences the possible complexity of the functional components for observing and predicting the user, such as the amount of data that can be obtained and processed. The computational power can also influence and restrict the *incentive* and *operability* of the user model. Regarding the choice for the incentive of the system, both passive and active user models have their benefits and disadvantages. When information is obtained passively, unnecessary information might be sent to the user model, while certain specific information might still be missing. When the information is gathered actively, due to this additional task, the load on the user model and the communication load within the rest of the system might both become very high. Also regarding operability, both online and offline operating user models have their advantages and disadvantages. When a user model is used online, much computational power is often necessary to keep it up to date. When considering a user model that is only used offline, the user model is not always available and up to date when the user is using the application, but because there are no computational constraints, more sophisticated and time-consuming algorithms are possible.

Additionally, as discussed in the technical requirements of user modeling (Section 2.1), one of the main guidelines for constructing the user model as a whole, is that a developer should always remember to make the user model as minimal as possible; only capturing the strictly needed data and functionalities, making the system easier to maintain and expand.

User profile

When defining the technical properties of the data in the user profile, the desired readability should be taken into account. For this reason the possible data types and structures could be restricted. Especially the privacy of user data is a very delicate feature and a much discussed subject. A very profound overview of the risks and solutions for handling user information, can be found in (Toch, et al., 2012) and (Mioch, et al., 2013).

When considering the user profile, the developer should always make sure to use and save as less data as is possible, while still keeping it up to date. In this way the user model operates in an efficient way, and the different components of the user model are not overloaded with useless information.

3.3 Conclusion

In this chapter, a general user modeling methodology has been presented to bridge the gap between generic conceptual methods to user modeling and domain-specific user model implementations. To construct this methodology, a generalized and more abstract view of user modeling has been constructed. Both the general functional components of a user model, and their functional and technical properties have been presented.

After the user model is constructed according to the methodology, it is important that the user model is validated, to ensure that it displays the desired behavior. Validating the behavior of the resulting user model is a research field on its own (Fischer, 2001; Toch, et al., 2012), and has become more and more important ever

since the user modeling domain emerged. For further reading about these subjects, we kindly refer to the broad range of existing literature.

In Chapter 7, to validate the use and usefulness of the proposed general user modeling methodology, it is used to construct a user model for an adaptive educational game in First Aid. First, in the next chapters, the research domain of intelligent agents will be introduced and combined with the domain of user modeling, resulting in an agent-based user model framework. This framework will also be used to implement the user model in Chapter 7.

Chapter 4

Intelligent Agents and the BDI paradigm

This chapter introduces the intelligent agent paradigm. Different concepts and approaches will be explained and discussed, where the emphasis is put on Beliefs-Desires-Intentions (BDI) based agents. In the next chapter, these concepts and terms will be combined with the concepts and aspects related to user modeling as discussed in Chapter 3. Together they will form a framework for user modeling with BDI-based intelligent agents.

4.1 Intelligent Agents

An agent is an entity that exists in a world, and that can observe, reason and perform actions in this world (Russell, et al., 2003). A typical example of an agent, would be a human being. However, also a thermostat that automatically perceives the temperature of the room and adjusts the heating satisfies this definition. The general agent deliberation cycle is visualized in Figure 4.1. The agent paradigm is especially useful for solving problems and creating programs in complex systems with parallel processes, such as distributed or multi-threaded systems (Jo, 2011a). Because an agent can act independently in its pursuit of its design objectives, the agent has the ability to exhibit robust but flexible problem solving behavior. When combining multiple agents, even more complex problems can be solved. It is also possible to handle parallel processes by using other paradigms, such as object-oriented or imperative programming, but due to the nature of these methods, it often takes more effort to gain the same result.

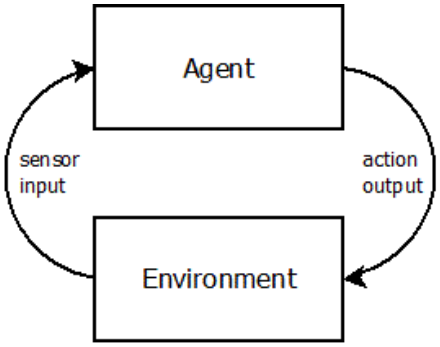

Figure 4.1: The general intelligent agent deliberation cycle, adapted from (Wooldridge, 2009)

There are many domains in which the agent paradigm has been applied, such as artificial intelligence, computer science, economics and social science. Its applicability ranges from the modeling of autonomous systems such as cognitive robots, to the simulation of certain social phenomena such as crowd behavior. For each domain, the range of attributes associated with agency is of varying importance. This has resulted in multiple definitions for the term agent. In this chapter, we will keep to the following definition of an *intelligent* agent by (Wooldridge, 2009): *an intelligent agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives*. Four important agent properties are associated with this definition (Jo, 2011b):

- Autonomy: the ability to operate without the direct intervention of humans or other agents and the ability to have a certain level of control over one's own actions and internal state
- Reactivity: the ability to perceive the environment and respond in a timely fashion to perceived changes in the environment

- Pro-activeness: the ability to exhibit goal directed behavior by taking the initiative
- Social ability: the ability to interact with other agents

A multi-agent system is a group of agents within a shared environment. Within this environment, the agents can perform tasks individually, but also cooperatively.

4.2 Intelligent agent architectures

The intelligent agent paradigm can be implemented in a large variety of ways. To make the development of an intelligent agent easier, several architectures have been proposed. The different architectures are especially concerned with how the agent should decide which action it should perform to satisfy its goals. According to (Russell, et al., 2003), the complexity of this decision-making process is influenced by the environment in which the agent is based. They present a terminology for categorizing the different environments, by making a distinction between the following properties:

- Fully observable vs. partially observable: the possibility to obtain complete, accurate, up-to-date information about the environment's state
- Deterministic vs. stochastic: the certainty about the resulting environment state after performing an action
- Episodic vs. sequential: the influence of an agent's actions on the current state, or future states
- Static vs. dynamic: an environment can only be changed by the agent, or is changed by multiple processes
- Discrete vs. continuous: whether or not the environment has a fixed, finite number of possible actions and percepts
- Single agent vs. multi-agent: the amount of agents in the environment

As one might expect, the most difficult environments are the ones that are only partially observable, stochastic, dynamic, continuous, and contain multiple agents. According to these properties, a developer can choose from several architectures for constructing the intelligent agent. These architectures can be classified in three categories (Russell, et al., 2003), each category increasing in complexity:

1. **Simple reflex agents,** or purely reactive agents, are agents that can only select actions on the basis of the current percept, ignoring the rest of the percept history. The agent's behavior is completely determined by the so-called condition-action rules: taking the according action when the right conditions are perceived.
2. **Model based reflex agents** are agents that keep track of the world around them, saving percepts to use for reasoning in a later stadium. This requires a model about the world that captures the rules of the world and the way the agent's actions are expected to influence this world. In this way, partially observable environments are easier to handle.
3. **Goal-based agents** are agents that also keep track of their goals, next to keeping track of the state of the environment. Goals are meant to describe situations that are desirable. Often, it is not possible to achieve goals with only direct actions, therefore planning and searching mechanisms are important parts of the agent. A utility function can be used to determine the usefulness of certain actions, thus providing preferences between different possible actions. The goal-based approach provides agents with the ability to handle complex environments in a flexible way.

On top of these different agent architectures, the ability to learn can be implemented, enabling the different architectures to handle uncertain information and to adapt to changing environments.

4.3 Intelligent agents with beliefs, desires and intentions

Due to their ability to handle complex environments, goal-based agents are the most interesting and expressive class of intelligent agents. Common architectures in this class are the Beliefs-Desires-Intentions

(BDI), Act-R and Soar architecture (Heuvelink, 2009). Each of these architectures approach behavior from a cognitive perspective. In this section we will look at the most intuitive architecture: the BDI model. The BDI paradigm provides a model for formalizing the way in which humans (think they) reason, forming goals and using their knowledge and beliefs to construct a plan to achieve these goals. This type of reasoning is more commonly known as *folk psychology* (Norling, 2003).

The BDI paradigm is based on the philosophical tradition of understanding practical reasoning – the process of deciding, moment by moment, which action to perform in the advancements of our goals (Bratman, 1987). Bratman formalized the relations between the mental states associated with the beliefs, desires and intentions of an agent. Intentions play a crucial role in the process of choosing what goal we want to achieve and how we want to achieve them: according to the desires and beliefs someone has, certain intentions are selected, which will focus the actions that are performed in order to achieve the goal and thus constrains the possibilities in the future.

4.3.1 Deliberation cycle

When using the BDI paradigm as an architecture for intelligent agents, an agent's decision making possibilities depend on the manipulation of data structures representing the beliefs, desires, and intentions of the agent. A typical BDI execution cycle contains the following steps (van den Bosch, et al., 2009; Jaques, et al., 2007):

1. according to the sensor inputs from the environment, often called events, the agent's internal beliefs, desires and intentions are updated;
2. select the applicable intentions based on the current desires and beliefs, add them to the intention stack;
3. select an intention;
4. perform the intention if it is an atomic action, or select a new plan if it is a sub-goal.

The agent deliberation cycle extended with the BDI-architecture, is visualized in Figure 4.2. The separate terms can be further refined in the following way:

- **Beliefs:** The agent's beliefs represent the perceived and inferred knowledge of the environment and the agent itself. Together, they form the informational state of the agent. Because the beliefs of the agent contain assumptions, it is possible that the beliefs of the agent are actually not true in the environment.
- **Desires:** The agent's desires can be seen as the goals the agent has in the environment; they form the motivation for the agent to perform certain actions. When an agent has a certain desire, it does not automatically mean that the agent will attempt to achieve it. During the deliberation cycle of the agent, first the feasibility of the different desires is tested, by looking at the existing beliefs and possible intentions to achieve the desire. When a desire is achievable, it will be chosen to be executed and it becomes active. An important property of the active desires, is that they must be consistent, so not contradicting each other.
- **Intentions:** The intentions represent the currently chosen course of action. When a certain desire is selected, the agent devotes its commitment to the intention that is assumed to lead to achieving this desire. More concretely, intentions can be seen as plans consisting of a number of actions, like a recipe for achieving the main desire. These steps can be executable actions in the environment, or sub-goals that have to be achieved first. Plans are initially only partially conceived, with details being filled in as they progress. An important property of intentions is that they are persistent. An agent will not give up on its intentions and will continue trying to achieve them, until either 1) the agent believes it has successfully achieved them, 2) the agent believes it cannot achieve them, or 3) the purpose of the intention is no longer present.

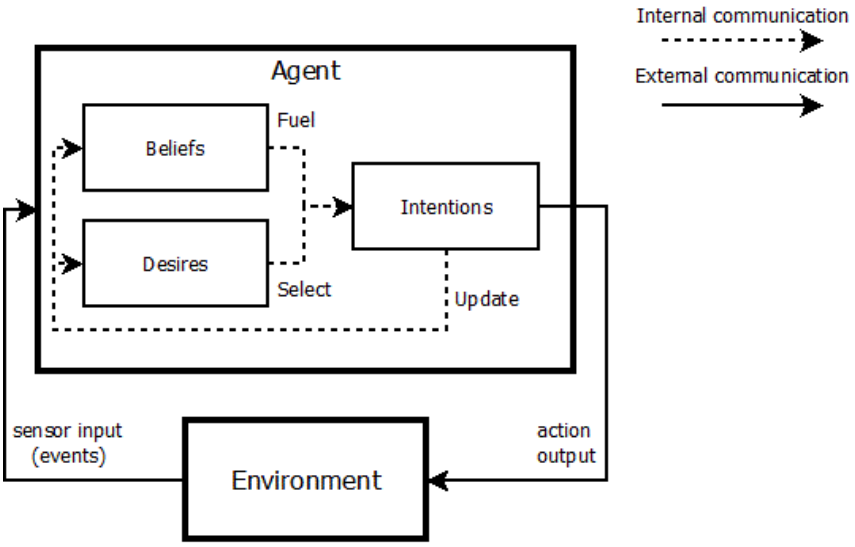

Figure 4.2: The BDI-based intelligent agent deliberation cycle

4.3.2 Higher level properties

The BDI approach for developing an intelligent agent results in several higher level properties and advantages. By using the goal oriented approach, actions are executed while keeping track of why the action is being executed. Whenever something goes wrong, the agent has the ability to backtrack its decisions and thus automatically recover from failures. This results in dynamic problem solving behavior, without explicitly solving each possible problem (Georgeff, et al., 1999).

Additionally, through the BDI paradigm, an intelligent agent has the ability to reason about its goals in a high level fashion. The underlying actions necessary for actually achieving the goals are captured in the different plans, independent of the reasoning process. This means that there is a strict separation between the data that is used in the model (captured in the belief base), the incentive of the model (captured in the different desires), and the actual execution of the different plans in the intentions. This results in a very modular system.

Finally, when looking at developing a BDI-based agent, the biggest advantage of using the BDI paradigm is its root in folk psychology: capturing human knowledge and reasoning for specific tasks (Norling, 2003). When people explain their actions, they tend to explain them in terms of intentions and plans, which in turn are explained in terms of beliefs and goals. Additionally, when people talk about the way in which they try to achieve their goals, they often do this in a hierarchical matter, thus resulting in partial plans that are relatively easy to formalize in a plan library for a BDI agent. It still requires a lot of effort and careful questioning when acquiring the information from people, but the fact that both the model builder and the subject being modeled are referring to the same concepts, does simplify matters.

4.4 Conclusion

In this chapter, the intelligent agent paradigm has been introduced. Due to their ability to act independently in the pursuit of their design objective, intelligent agents possess interesting abilities, such as the possibility of flexible behavior, robustness and pro-activeness. Multiple architecture of intelligent agents have been discussed. One of the most interesting architecture is the BDI paradigm, which employs a strongly goal-based approach.

In the next chapter, the concepts and terms that were introduced in this chapter, will be related to the user modeling field as set out in Chapter 2 and 3. Together they will form a framework for user modeling with BDI-based intelligent agents.

Chapter 5

An Agent-based User Modeling Framework

In Chapters 2 and 3, the user modeling field has been extensively discussed. When creating a user model, many concepts and properties must be taken into account. Depending on the application domain, the user model can become an intricate component. When constructing a user model, the knowledge and procedures are often defined explicitly, resulting in a complex system in which many functionalities and processes interact with each other. A user model has the task to adapt the system that surrounds it, but the user model itself is often not very adaptive. After the user model has been constructed, it remains fairly static and reactive: adaptation takes place according to the rules and decisions made during the implementation phase.

Section 4.2 described a taxonomy to categorize different types of environments in which intelligent agents can operate (Russell, et al., 2003). The same taxonomy can be used to categorize the user modeling problem when adding personalization to a system. The complexity of the desired user model depends on variables comparable to the ones identified by Russell & Norvig. When looking at user modeling in general, there are at least two common features across all user modeling problems: 1) a certain amount of unpredictability and 2) limited control over the behavior of the user. In addition, user models regularly face the challenge of dealing with 3) incomplete data about the user, and 4) an unlimited amount of various percepts about the user and the environment. So in terms of Russell & Norvig's taxonomy, a user model must be able to deal with a moderate to heavily complex environment. As the complexity of the user model's task increases, the need for a flexible, robust, intelligent user model arises. The user model must be able to deal with larger data sets, or must be able to deal with compound tasks and procedures. In such cases, the user model often becomes cluttered and needlessly extensive, no matter the methodology that is used to develop it. To solve these problems, the use of an intelligent agent for user modeling is proposed.

As discussed in Chapter 4, intelligent agents have the ability to act independently in the pursuit of their design objective. This results in a high level of flexibility and robustness which is beneficial in complex environments. Intelligent agents that are based on the Beliefs-Desires-Intentions (BDI) paradigm further refine and expand these properties, by introducing a well-structured, goal oriented, and insightful approach to the internal mechanisms of the agent.

In this chapter, the properties and possibilities of BDI-based intelligent agents will be used to construct a framework that helps developers to create a flexible user model for complex personalization problems. In Section 5.1, the concepts and properties associated with user modeling and intelligent agents will be compared to each other. In Section 5.2, these properties are combined, to construct the framework.

5.1 User modeling with intelligent agents

An agent-based user model is in charge of all the functionalities and data needed to personalize the system. Below, the concepts and desired properties related to user modeling are coupled to the BDI-based intelligent agent paradigm.

5.1.1 Personalization cycle resemblance

When comparing the intelligent agent reasoning cycle with the personalization cycle, there is much resemblance. The cycle identified for personalizing a system consists of three steps, i.e. observe the user, predict the user's behavior, and adapt the system to the user's (future) needs. The deliberation cycle of an agent is to observe the environment, reason about the known information, and then perform an action in the environment. This resemblance shows a direct mapping of the general functional components of a user model to the components of an intelligent agent.

5.1.2 Technical software development requirements

In Section 2.1, the general technical requirements of a user model were discussed. Ideally a user model should be as modular as possible, enabling decomposition, modifiability, and reusability. These properties also require a certain level of abstraction. The technical properties of a BDI-based intelligent agent fit these technical requirements well.

Especially when considering **software decomposition**, BDI-based intelligent agents possess strong features. Within the BDI architecture, the dynamic information about the user can be represented in the agent's beliefs, the incentive for performing the user model's goals and sub-goals can be represented in the agent's desires, and the procedures for actually achieving these goals can be represented in the agent's intentions (action plans). In this way, the data and the actual procedures are strictly separated from each other, resulting in a modular and generic user modeling framework.

Due to the strong decomposition of the different components within a BDI-based intelligent agent, also the **reusability** and **modifiability** of the components are high. Because of the dynamic coupling of the user model goals (desires), user dependent information (beliefs), and procedural plans for executing operations (intentions), not every specific instruction or operation needs to be explicitly written out. This reduces the programming effort and introduces a strong mechanism for dynamically catching and handling problems and errors. Furthermore, normally when adding additional program constructs -such as features, exceptions, or restrictions, to the user model, the procedures to handle these constructs need to be explicitly defined in the right place. Due to the iterative and dynamic deliberation cycle of a BDI-based agent, through adding additional plans and (accompanying) goals it is relatively easy to modify the model.

Additionally, because all the user-related information is strictly separated from the other system components and put into one database - the agent's beliefs, it is easy to make sure the whole system can access the same set of data. Only one knowledge base needs to be updated and maintained, which means that it is easier to keep the user information consistent and non-redundant.

Another major advantage of the intelligent agent approach to user modeling is the high level of **abstraction** of the development method and the resulting model. The BDI paradigm uses high-level concepts for the internal mechanisms of the model, but on a lower level still every single specific technique can be used. For instance, the most appropriate plan can be chosen according to the current beliefs and desires of the model, but within this plan, everything can still be filled in according to the developer's desires. Whether the developer uses simple rules or a heavy computational method for predicting the user, is still up to the developer.

Another possible advantage of using the BDI paradigm for creating the user model is the use of folk psychology. Employing folk psychology leads to a high level and intuitive way of designing the user model. It is especially attractive to model a human user, because it intuitively resembles the way how people make decisions and form plans. This also means that the resulting user model becomes more insightful and intuitive to read and interpret for both the programmer and the end user, without putting much extra effort in it.

5.1.3 Intelligent agents in user modeling literature

Intelligent agents have been used before in several user modeling research projects. In these projects, emphasis has especially been put on enhancing the user model with the necessary autonomous and proactive behavior (Schiaffino, et al., 2008; Heuvelink, et al., 2008; Razmerita, 2009; Jo, 2011c; Kobsa, 2007). The possibilities of a distributed agent design for user modeling have also been further investigated (Lorenz, et al., 2005; Specht, et al., 2006). In addition to intelligent agents in general, the BDI paradigm for constructing a user model has also been used. Especially translating human reasoning processes into system operations (Norling, 2003; Jaques, et al., 2007), and creating believable behavior have been investigated (Ci, et al., 2009; Harbers, 2011).

In these projects, intelligent agents are only used to develop the user model of the specific project without drawing extra conclusions, or generalizing to a reusable method. Unfortunately, none of the mentioned projects discussed a more thorough investigation of the possibilities of combining the user modeling and the

intelligent agent field. In contradiction to these projects, in this chapter we will look at the applicability of (BDI-based) intelligent agents for user modeling on a higher level, thereby utilizing the general structures and properties of intelligent agents to solve problems that arise when modeling the user.

5.2 The BDI-based intelligent agent user model framework

This section presents the framework for agent-based user modeling. A framework captures the standard structure of an application, and can be used by developers to implement their specific system. Part of the user model properties and functional components can already be placed in this framework, while the more specific details defining the application can be filled in by the developer.

The general user model properties, described in Chapter 3, are used to discuss the features and possibilities of the framework. When a certain property is not mentioned, it means that the property should be filled in by the general user modeling methodology, and thus is not important in the framework itself. An overview of the general components and their functional and technical properties can be found in Appendix A. Within the framework, a distinction is made between the user model and the rest of the application components. A visual representation of the framework can be found in Figure 5.1.

5.2.1 Framework components

Within the framework, next to the user model, the other components of the system are considered to be black box components: only their input and output are taken into account, and there is no knowledge of their internal workings. Two specific types of system components are important: components that are able to *obtain user data* and components that can be *adjusted* by the user model. These components correspond to

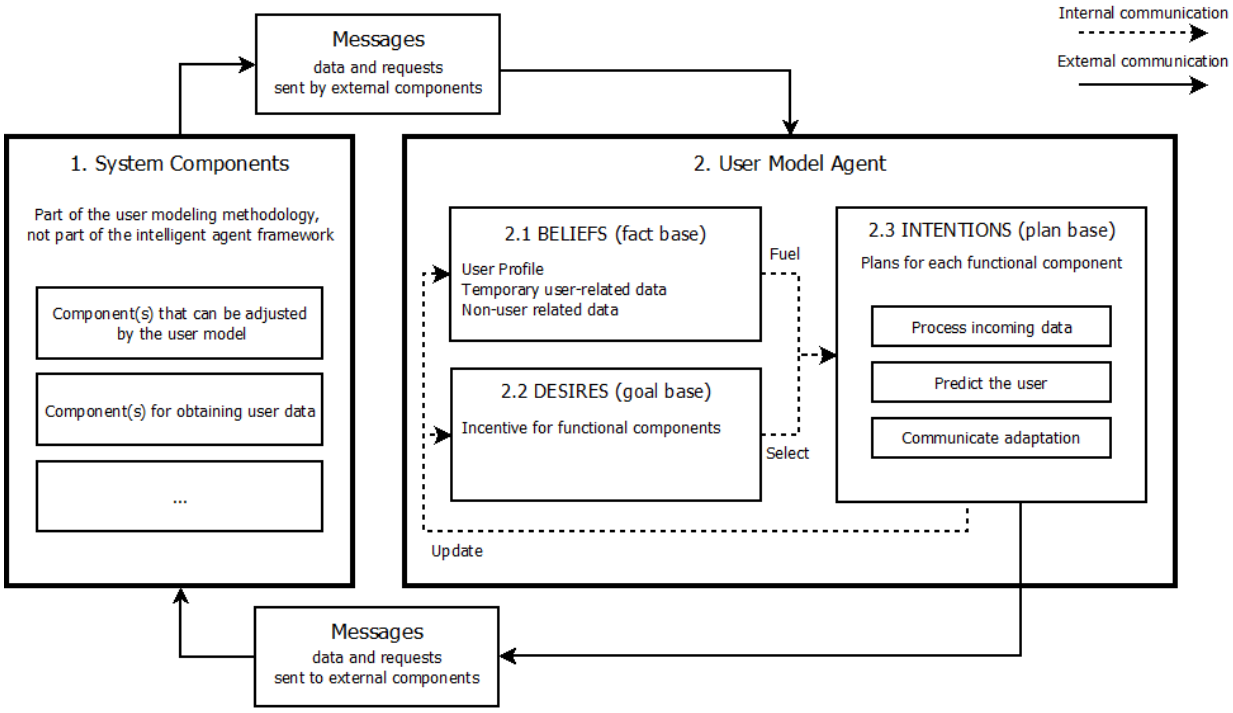

Figure 5.1: A BDI-based intelligent agent user model framework

the steps from the personalization cycle responsible for observing the user and adapting the system. A system component can also belong to both types. The exact content and properties of these components should be defined during the employment of the general user modeling methodology. Only their input and output are important for the internal structure of the user model, and thus for the user model agent part of the framework. From now on, the other system components are called the *external* components.

5.2.2 User model agent

Within the framework, the user model is depicted by a BDI-based intelligent agent. As mentioned in Section 5.1, the personalization cycle seen in user models strongly conforms to the deliberation cycle seen in agents. Normally, an agent receives sensor input from the environment, which serves as a trigger to undertake a certain action (output) in the environment. For an agent-based user model this cycle needs to be slightly refined:

1. Through the external components for observing the user, the user model agent obtains the necessary user data.
2. Within the agent itself, the input is processed and the most appropriate adaptation (output) is determined.
3. The agent communicates the necessary adaptation back to the system, where the selected external component is adjusted.

Depending on the *incentive* of the user model, this cycle can somewhat differ. When the incentive of the user model is *passive*, an external component instructs the user model to infer certain information, or to perform a certain action. When the incentive is *active,* the agent itself infers when action should take place. This can be done in either a reactive or a pro-active way. With the user data obtained from the external components, the agent can determine when to execute certain actions, for instance to intervene in the system, or to obtain additional user information. The *operability* of the user model also depends on the user model incentive. When passive, the external components determine the online- and offline loop of the user model. When active, the user model might infer the start of the online and offline phase by itself.

The user model communicates with the rest of the system through messages. A message from an external component can contain user-related data or a request to the user model. The agent-based user model itself can directly address the external component that should be adjusted, or can send data back to the external component that requested it.

To enable the properties defined above, the BDI-paradigm is used for the agent's internal mechanisms. In the following subsection, the agent's beliefs, desires, and intentions are further refined.

Beliefs

The data used in the user model is saved in the agent's beliefs. The beliefs are updated by the messages sent from the external components, and by the internal actions executed through the agent's intentions. The majority of this data is related to the user profile, but also other data needs to be saved. Therefore, the alternative term *fact base* might be more appropriate. Preferably, within the beliefs, there is a distinction between data related strictly to the **user profile**, like the user's progress, **temporary user-related data**, like the intermediate values used to actually calculate the user's progress, and (possibly temporary) **non-user-related data**, like data related to the system's configurations or predefined inference rules. Within the beliefs of an agent, this distinction could be accomplished by creating separate databases for each type of data. Within the agent's beliefs, also the data of multiple users could be saved.

Desires

Desires, or more often called the goals of the system, are responsible for the execution of the operations in the user model. Within the whole deliberation cycle of the agent, they play a big role in the *incentive* of the user model. Together with the current beliefs of the agent, the desires can *actively* determine when to execute certain action plans from the intention plan base. For instance, to take over a task when it is detected that the user is having difficulties, or to decide to obtain additional information about the user, when it is detected that the agent cannot make a decision. If the incentive is *passive* (procedural), the desires are mostly determined by the external components and thus often are more straightforward. Desires are updated by the messages sent from the external components and by the internal actions executed through the agent's intentions.

Intentions

The agent's intentions consist of a collection of action plans that can be used to satisfy the desires of the agent. Within the deliberation cycle of the agent-based user model, these action plans are responsible for the operations that are necessary to personalize the system. Globally, these operations can be categorized according to the steps defined at the beginning of this section, i.e. process the incoming user data, predict the user, and communicate the adaptation to the external components.

According to the present desires and beliefs of the agent, the most applicable action plan is selected to be executed. For each possible desire, multiple plans can be defined. For each plan, a set of requirements can be specified to define its applicability under various circumstances. Through the definition of slightly different action plans, problems, like invalid input, can be solved in a dynamic fashion. Each action plan has the capability to request and change the values of the agent's desires and beliefs. The exact content and properties of the action plans can be filled in with the use of the user modeling methodology presented in Chapter 3.

5.3 Conclusion

In this chapter, a framework was proposed for user modeling with the use of BDI-based intelligent agents. The framework is meant to help developers with constructing user models for complex personalization problems, such as problems that involve incomplete data about the user, and dynamic and unpredictable user behavior. For systems that demand less of the user model, such as recommender system or simple educational systems, the framework is less useful.

The proposed framework is complementary to the general user modeling methodology defined in Chapter 3. Whereas the methodology covers the general questions that need to be answered for constructing a user model, the framework is more focused on ensuring that the technical and overlying, non-application-specific, properties of user modeling are satisfied. Together, they can be used to add personalization to a system.

The similarities between the user modeling and intelligent agent domains have been discussed, after which the advantages and possibilities have been identified. The framework was constructed by coupling the requirements and properties of user modeling with the advantages of using a BDI-based intelligent agent architecture.

In Chapter 7, the framework is tested by using it, together with the general user modeling methodology, to implement a user model for an adaptive educational game in First Aid.

Chapter 6

Adaptive Educational Games

In Chapter 2, the user modeling application domain of Adaptive Educational Games (AEG) was introduced. In this chapter, the domain is further investigated so it can be used in the next chapter as the application domain for the user modeling methodology. This chapter can be seen as the domain analysis that should be performed before the user modeling methodology can be applied.

AEGs are complex educational games that combine several research areas, including Serious Games, Intelligent Tutoring Systems (ITS), psychology, and didactics, to enhance the student's learning experience (Peeters, et al., 2012a). The emphasis in AEG is put on the development of educational applications using a well-founded psychological background, employing methods and techniques to enhance the effectiveness of the application. AEG are based on the concept of scenario-based training (Peeters, et al., 2011; Peeters, et al., 2014b) , a practical training form in which trainees acquire complex skills by performing a job in a scenario with a short and realistic storyline. A complex skill is a skill that needs practice to master, such as situation awareness or decision making. These skills cannot be learned from simply reading a book and often rely on cognitive as well as physical abilities. Scenario-based training is widely used to train policemen or firemen how to act in dangerous situations, such as a disaster or a bomb alert. However, the major downside of scenario-based training is that its organization often requires a lot of effort, people, and resources (Peeters, CA). The use of an AEG is meant to solve this problem by presenting scenario-based training in a digital form.

AEGs enable a trainee to practice complex skills anywhere and anytime he/she wants. An AEG is an application that uses a virtual world that resembles the professional working environment. This virtual world is often combined with artificial intelligence techniques to create a targeted learning experience, for instance, by adapting the storyline based on didactic considerations (Peeters, CA). AEG can be considered to be the next step in the personalization of educational applications, building on the domains of Serious Games and ITS. Serious games are computer games with an educational purpose. They offer trainees the means to develop their skills and knowledge in a playful, but also meaningful and practical training setting. Fun, knowledge transfer, and reality are carefully balanced out to provide a meaningful, immersive, and motivating learning experience (Peeters, et al., 2012b; Korteling, et al., 2011; Johnson, et al., 2005). The ITS-like part of the AEG is concerned with employing adaptive strategies, grounded in didactic principles derived from training theories and instructional design. Thus, ideally an AEG combines the immersion and appeal offered by games and the didactic structure offered by ITSs.

Many important problems associated with the adaptation of the system depend on the behavior of the user, which means that the process of user modeling in an AEG is a delicate and complex task. Besides, compared to other user modeling domains such as internet applications or ubiquitous systems, AEG have much computational power available for adaptation. The features of AEG mentioned above make AEG an interesting domain for user modeling.

This chapter is structured as follows: in Section 6.1, the specific requirements for an effective AEG, as defined in the literature, are set out. In Section 6.2, the purpose the user model could fulfill in an AEG is discussed and in Section 6.3, the implementation of an AEG in a Multi-Agent System, without the user model, is expounded on.

6.1 Requirements of an Adaptive Educational Game

A (preliminary) list of minimum requirements an AEG must fulfill has been identified by (Peeters, 2014a), with each requirement strengthened by theories from human factors research. These requirements (R01-R17) are expressed in the list below.

R01. The simulated environment should 1) accurately display only the relevant elements of the real-life environment, and 2) provide more abstract representations of the less important elements.
R02. Scenarios must generally contain complete storylines in the form of whole tasks
R03. The amount of new information presented in the scenario should be reduced to the absolute essential
R04. Scaffolding techniques should be employed to ease the learner into independent performance
R05. Learners should receive help on how to effectively employ scaffolds such as hints, prompts, and immediate feedback
R06. Scenarios must be responsive to the learner's choices
R07. Over the course of training the learner must encounter multiple variants of scenarios that address similar learning goals and tasks
R08. Scenarios must address learning goals that challenge the learner yet lie within reach given the learner's current competences
R09. Learning goals must be transparent to the learner and tied to observable performance standards
R10. Scenarios must be selected dynamically based on the learning goals they address and the prior competences of the learner
R11. The events within a scenario should be independent with regard to the learner's outcomes
R12. The learner must not be distracted and/or interrupted during the task performance
R13. When the scenario has finished, the learner's performance must be evaluated by reviewing situation assessments and decisions made during the scenario, and the resulting consequences
R14. After reflection, the learner must receive directed and overarching feedback by looking at both the outcomes of the reflection phase and the performance outcomes compared to the performance standards associated with the learning goals
R15. Scenarios must be designed so that learners can experience the results of particular procedures themselves, thereby discovering the usefulness of standardized procedures and understanding how they were originally constructed.
R16. The learner must receive additional part-task training for routine tasks and/or tasks that cannot be learned within the virtual environment
R17. The instructor must be able to understand the system's processes and control them at a certain level

According to the identified requirements, a functional architecture was created. The schematic overview of the architecture can be found in Figure 6.1. The different components of the functional architecture are:

- The **scaffolding component** controls the support offered to the learner during playtime. The scaffolding component should at least meet requirements R04, R05, R08, R11, and R12
- The **scenario creator** constructs a scenario based on the information contained in the domain representation, the constraints provided by the instructor, and the trainee data provided by the learner model. An extensive study to the scenario creator has done by (Ferdinandus, et al., 2013). Requirements R01, R02, R03, R07, R09, and R15 are important for this component
- The **authoring tool** offers instructors the possibility to interact with the AEG in order to manipulate the scenarios offered to the learner. The authoring tool should at least meet requirement R17
- The **learner model** provides the scenario creator with a learning goal based on the learner's progress, which in turn constructs a suitable scenario. The requirements that are important for the learner model, are R07, R08, and R10
- The **reflection component** supports the reflection and feedback after the scenario has finished. The reflection component should at least meet R13 and R14
- The **virtual environment** is inhabited by non-player characters and the learner's avatar. It is within the virtual environment that the learner can interact with the other characters. R01 and R06 are important requirements for the virtual environment

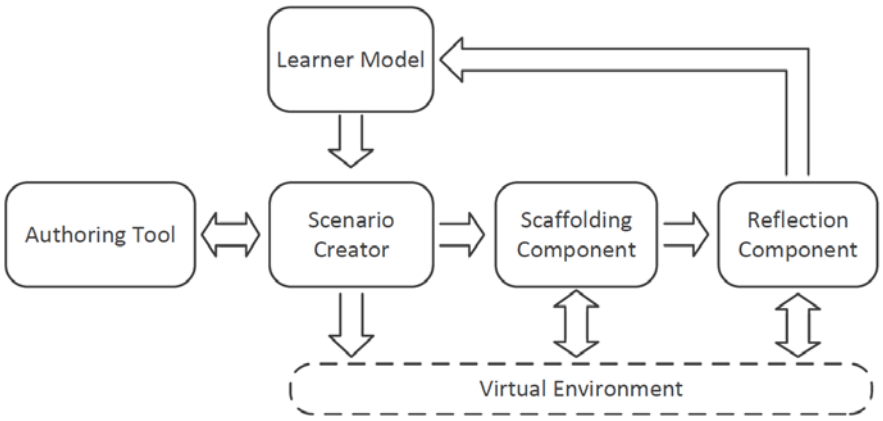

Figure 6.1: the functional architecture of AEG, source: (Peeters, 2014a)

An online and offline part can be distinguished in this functional architecture. The online part, or inner loop, of the AEG consists of the Scaffolding component, Reflection Component, and Virtual Environment. Together they provide the playtime behavior of the AEG. The offline part, or outer loop, of the AEG consists of the Authoring Tool, Learner Model, and Scenario Creator. Together they provide the initialization of the online part and the processing of the results.

6.2 Personalization of an Adaptive Educational Game

To optimize the learning process and learning outcome for the trainee, personalization plays a big role in the AEG. To be more concrete, the experience that the game world provides is personalized in the sense that it offers just the right amount of challenge and the right topic for the trainee to be motivated, and to be able to comprehend the presented problem or situation (Peeters, 2014a). Just like in the overall functional architecture, also in the personalization process there can be made a distinction between an online and an offline part.

6.2.1 Personalization offline

As discussed in the functional architecture, the **learner model** provides the features for the personalization of the AEG in the offline part. Its main purpose is to provide the scenario creator with a learning goal based on the trainee's progress, so the scenario creator can construct a suitable next scenario. The main requirements associated with this task are R08, and R10.

According to these requirements, the learning focus should be challenging, so that the trainee is stimulated to extend his competences. A challenging focus also prevents the trainee from getting bored. As the trainee progresses, the difficulty and complexity of the selected learning focus should also increase. However, it is also important that the learning focus always lies within close reach for the trainee to grasp. This idea is based on the *Zone of Proximal Development* (Vygotsky, 1978; van Merriënboer, et al., 2002), a theory from education psychology that describes the distance between what the trainee is currently able to do by himself, what he can currently accomplish with the help of someone (or something) else, and what he will be able to do independently in the near future. In this way competences are trained in a gradual, constructive way and thus improve the trainee's learning process.

To select a learning focus that confirms to these requirements, the user model needs to have an accurate and up-to-date representation of the trainee. This representation should capture the level of the previously trained competences [R10]. Additionally, because the instructor plays an important role in the AEG [R17], it is important that the instructor can have some insight in the progress the trainee is making. To a certain extent, it is also desirable that the instructor understands the processes that take place in in the user model.

Finally, when the selection of the learning focus takes place, multiple variants of scenarios that address similar learning goals and tasks should be selected, in which the user model can also play a role [R07].

6.2.2 Personalization online

The main purpose of personalization during the online part of the AEG is to help the trainee to perform the scenario. The scenario should be adapted to the trainee and suitable feedback should be provided. The main requirements associated with these tasks are R04, R08, R13, and R14.

While the trainee is playing a scenario, the scenario should be adapted through the employment of scaffolding techniques. Scaffolding techniques regulate the amount of support the trainee receives during playtime. For instance, this support can be the amount of feedback the trainee receives, the amount of time the trainee has to perform a certain action or the tasks the trainee should perform simultaneously. The **scaffolding component** is mainly responsible for this task, but the user model should provide this component with the necessary user information for the most suitable adaptation. At the start of the scenario, the initial amount of support should depend on the trainee´s current progress and preferences. At playtime, however, the level of support should be adjusted depending on the trainee's in-game behavior. Through the employment of scaffolding techniques, the trainee is eased to gradually perform the tasks on his own. Additionally, by not presenting too much information at the same time, cognitive overload is prevented. When employing scaffolding techniques, it is important that the trainee is not distracted and/or interrupted during the task performance [R12], in order to keep the trainee concentrated and in a state of flow (Gazzaniga, et al., 2006).

The second important task of the user model in the online part is to provide suitable feedback. In order for this feedback to be of any help to the trainee, it should be concrete/specific and focused on the trainee's observable behavior. The **reflection component** is mainly responsible for this task. After the scenario is finished, the trainee's performance is evaluated by reviewing the trainee's decisions and actions during the scenario. According to the performance standards associated with the learning focus, the trainee is given directed and overarching feedback about his performance, to gain a deeper understanding of the learning material. To enable this feedback and reflection step, the trainee's (relevant) behavior should be closely monitored and registered.

6.3 A multi-agent system for Adaptive Educational Games

In collaboration with Ruben de Jong and Marieke Peeters, a Multi Agent Director System (MADS) was created, capturing the essence and main features of an AEG (Peeters, 2014a). To implement the MADS, the agent programming language 2APL (Dastani, 2013) and the OperA Conceptual framework and methodology for multi-agent organizations (Dignum, et al., 2011) was used. In Chapter 7, 2APL will be further deliberated on.

To satisfy all the requirements that were set out for AEGs, the AEG needs to be able to perform complex behavior, involving multiple layers of reasoning mechanisms. The AEG relies on a large amount of knowledge about the trainee and the instructor to be able to perform all the tasks it is meant to accomplish, like selecting or generating suitable scenarios, monitoring the trainee, keeping track of the consistency of the story-line and conflict resolution. When using a centralized approach, the main challenge is the scalability of the whole application: keeping track of all the objects in the game world, the learning goals, the different NPCs and their possible behaviors, and the consequences thereof. To overcome these problems, the AEG's knowledge and abilities are distributed over a multi-agent system. In this multi-agent system every component and/or actor is represented by an Intelligent Agent.

6.3.1 Components

The MADS consists of three components: the virtual environment, the graphical user interface, and the multi-agent system.

Virtual environment and graphical user interface

The environment is the meta-representation of the game world in which the scenario takes place. It is used by the trainee and the different agents to interact with each other. For instance, the trainee and a role-playing agent should be in the same location in the environment to be able to perform an action onto each other. The actions performed by the trainee and/or the agents also influence the environment. For instance, when the

trainee picks up a chair, the properties of the environment change. In the graphical user interface the trainee receives information about the events that happen in the environment and the possible actions the trainee can perform. The trainee can perform actions by clicking the appropriate buttons. In addition to the available actions, information can be obtained from the user through pop-up windows that pose direct questions to the trainee. Figure 6.2 gives an impression of the user interface and a possible question in a pop-up window.

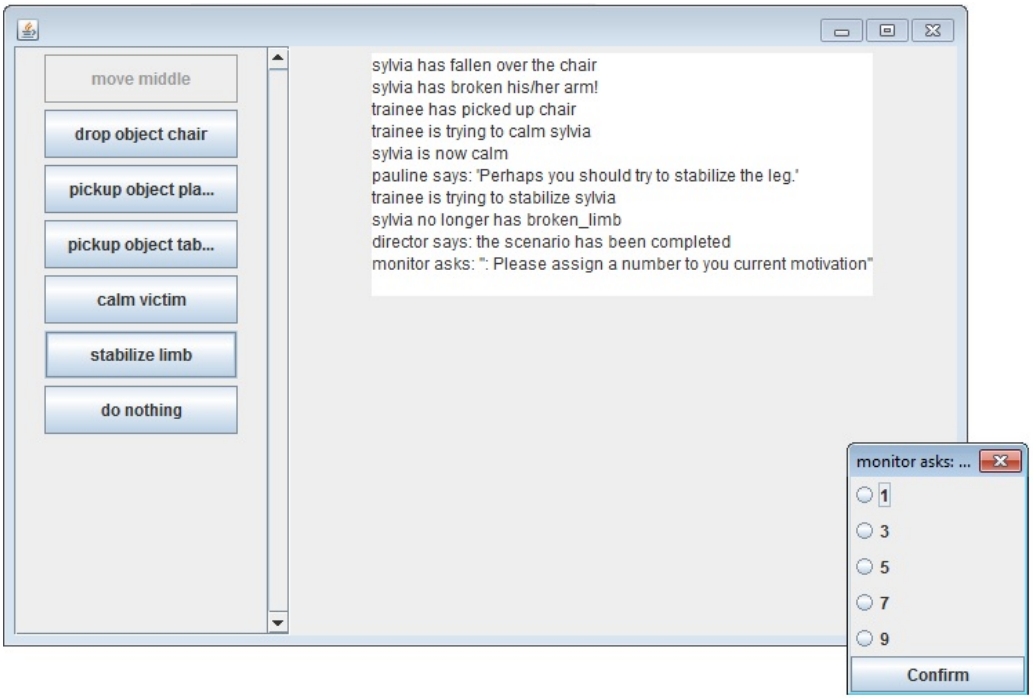

Figure 6.2: the user interface of the MADS

Multi-agent System

The multi agent system consists of several agents representing the different components of the AEG. The different agents are:

- The ***director agent*** can be seen as the representation of the instructor in his absence. The director is the central agent in the MADS, controlling the main flow of the application, and taking care of all the high level communication between the different agents. The director also has the ability to start an auction between the role playing agents, for admeasuring roles and taking on actions before and during the scenario
- The ***planner agent*** is responsible for planning or selecting a suitable scenario plan, based on the learning focus and support level (expressed through the initial level of scaffolding). The planner agent in the MADS contains multiple predefined scenario plans for testing
- The ***monitor agent*** keeps track of the actions that are performed by the trainee in the scenario environment and intervenes when necessary. The monitor agent also has the ability to ask the trainee direct questions through the user interface
- The ***trainee agent*** is the avatar of the trainee in the system. The trainee agent keeps track of the current state of the trainee, such as the trainee's location or the objects in the trainee's inventory. The agent is used to determine the possible actions and interactions the other character agents and the human trainee can perform with each other and in the environment
- ***Role playing agents*** control the non-player characters in the environment. They can either be a victim or a bystander who can help the trainee out. Role playing agents are able to sense and act in the environment, based on their goals. Through interaction with the trainee, they create a series of events that triggers the trainee to perform actions belonging to the training task

6.3.2 Application running cycle

The application cycle of the MADS can be seen in Figure 6.3, and consists of the following steps:

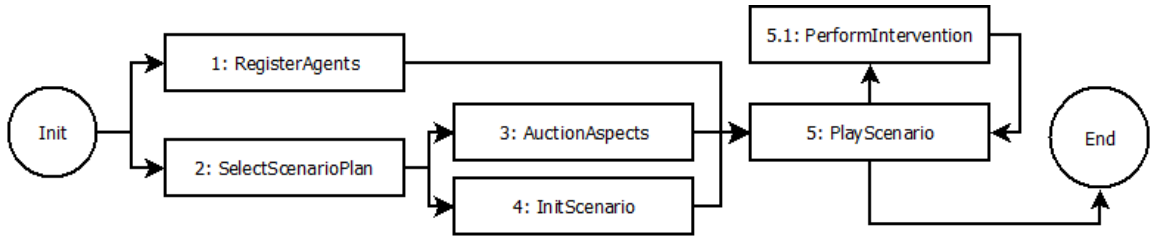

Figure 6.3 The MADS deliberation cycle, made in DIA

1. When the application is launched, all the agents start by registering with the director
2. According to the predefined learning focus and level of scaffolding, the planner selects the most suitable scenario plan and sends it back to the director. This scenario plan contains the action plan and the different aspects that must be assigned to the character agents before the scenario can start, such as the character locations, roles and tasks
3. The director starts one or more auctions to assign the necessary aspects from the scenario plan to the registered character agents
4. At the same time, the director sends the action plan of the selected scenario plan to the monitor, who initializes the next scenario
5. When all the auctions have come to an end, everything is in place and the scenario is ready to start. The director notifies the monitor that the scenario can begin. The trainee plays the scenario, where the action plan determines which action should be performed. The trainee is monitored by the monitor agent
   5.1. Whenever the trainee waits too long, an intervention is chosen, according to the action plan and the level of scaffolding. The director starts an auction to let the most believable character agent perform the intervention (either giving a hint or performing the action).
6. When all the actions of the action plan are performed, the scenario is finishes and the application ends

6.4 Conclusion

Adaptive Education Games are an interesting application domain for applying personalization, due to their complexity and associated challenges. In this chapter, the concept, requirements, and functional architecture of AEG have been broadly discussed, as well as a simplified implementation of an AEG in a multi-agent system.

According to the discussed requirements, the possibilities and necessities of a user model in an AEG have been identified. In the next chapter a user model for an AEG will be constructed, with the use of the general user modeling methodology and the BDI-based user modeling framework.

Chapter 7

Implementing a user model in an Adaptive Educational Game in First Aid

In Chapter 3, a general methodology was proposed to help a developer add personalization to a system. In Chapter 5, an agent-based framework was proposed to help a developer implement a user model in a complex system. In this chapter, the use and usefulness of these two methods will be validated and evaluated by applying them to an adaptive educational game (AEG) in First Aid. Due to the complexity of the personalization problem, and the importance of correct adaptation, an AEG in First Aid is suitable for using the agent-based user model framework.

Adaptive educational games have been extensively discussed in the previous chapter. The different requirements and properties of an AEG that were identified in this chapter can be seen as the domain and task analysis that should take place before the methodology can be used. The multi-agent system for adaptive educational games that was discussed in Section 6.3 will be used and expanded to implement the user model. The construction of the user model focuses on the features enabling the initialization of a scenario, and the features enabling online adaptation. Because of the nature of these features, there is a distinction between the necessary operations taking place online, while the trainee is interacting with the scenario, and the necessary operations taking place offline.

The AEG prototype focuses on the training domain of First Aid. First Aid is concerned with the basic knowledge about how to help people who are suddenly sick or injured. For example, First Aid is used at accidents to help an injured person until professional medical treatment can be provided. It generally consists of a series of simple techniques that an individual can be trained to perform. Scenario-based training is often used to train people to perform First Aid, which makes it an interesting application domain for an AEG. A thorough investigation of the field of First Aid for implementing an AEG has been done by (Peeters, 2014a).

In the different sections of this chapter, the user modeling methodology will be followed. Section 7.1 describes the conceptual design phase, Section 7.2 describes the functional design phase, and Section 7.3 describes the technical design phase, in which also the agent-based user model framework is used. Section 7.4 describes the implementation of the technical design with the use of the 2APL agent programming language. To ensure that the resulting prototype displays the desired behavior, in Section 7.5 the user model is verified by means of use cases analysis.

7.1 Conceptual design phase

In this section the conceptual design phase of the general user modeling methodology will be applied to an AEG for First Aid. The conceptual design phase is concerned with the fundamental and abstract ideas behind adding personalization to the application. In this way, the general purpose and requirements of the user model are defined.

7.1.1 Why will there be adaptation?

As was concluded in Section 6.2, personalization of an AEG can be used to optimize the learning process and learning outcome for the trainee. Through personalizing the experience that the game world provides, the application can offer just the right amount of challenge for the trainee to be motivated, and to be able to comprehend the presented problem or situation. Online, the main purpose of the user model is to help the trainee to perform the scenario. This will be done by inferring the right amount of support, defined in the appropriate level of scaffolding. Offline, the main purpose of the user model is to personalize the next scenario. According to the requirements of an AEG, scenarios should gradually increase in difficulty and

complexity (Zone of Proximal Development); therefore, the user model should select an appropriate learning focus and initial level of scaffolding.

7.1.2 When will the system be adapted?

Online, the adaptation of the level of scaffolding should be determined by the behavior of the trainee. Basically, when the trainee performs badly while playing the scenario, the level of scaffolding should increase. When the trainee performs the right actions, the level of scaffolding should decrease. Offline, the trainee's progress can be used as a trigger to adapt the learning focus. To track this progress, some kind of competence representation should be available. To infer the right amount of challenge, the trainee's motivation needs to be taken into account as well. When the trainee shows good results and a high motivation, the difficulty of the learning focus can gradually increase. Otherwise, a more deliberate adaptation should take place, possibly resulting in a less complex focus. Because the trainee needs to encounter multiple scenarios that address a similar learning focus, a learning focus should be trained multiple consecutive times. The initial level of scaffolding can be adjusted according to the progress the trainee has shown regarding the selected learning focus.

7.1.3 What will be adapted?

To achieve the personalization goal, the behavior of the system, applied through changing the system configuration, and the content that is presented should be adapted by the user model. Online, only the level of scaffolding should be adjusted. The *scaffolding component* that receives this adjusted level of scaffolding is responsible for applying the rest of the adaptation concerned with changing the current content, structure, and presentation of the application. Offline, the content of the application is adjusted indirectly. The user model should determine which learning focus and initial level of scaffolding should be selected, after which the *scenario creator* and *scaffolding component* use this information to create the next scenario.

7.2 Functional design phase

In this section, the functional design phase will be applied to the AEG in First Aid. The definitions of *why*, *when* and *what* to adapt are used to fill in the details and desired properties of the three main functional components, the overall behavior of the user model, and the user profile.

7.2.1 Main user model functional components

The main user model functional components are responsible for observing the user, predicting the user, and adapting the system.

Predicting the user

Online, in order to select the appropriate amount of support to the user, the level of scaffolding should be predicted.

1. **Granularity**. The prediction should be optimized for the trainee. As the trainee interacts with the system, the variables of the functional component should be adjusted to the trainee´s preferences, such as the adaptation speed or the available time to perform an action.
2. **Accuracy**. The adaptation should become accurate as soon as possible.
3. **Type of trigger**. User-based; the adaptation is triggered directly by the actions the trainee performs in the environment and their correspondence to the predefined action plan.

Offline, in order to personalize the initialization of a scenario, the most appropriate learning focus and initial level of scaffolding should be predicted. To infer this data, additional **sub-operations** are necessary. Operations should be available that can determine the trainee's current motivation, determine the performance score of the previous scenario, and update the relevant competences according to this score. Considering the general properties:

1. **Granularity**. The prediction mechanism should be optimized for a single trainee. This means that accurate data about the trainee ´s progress and motivation should be kept.
2. **Accuracy**. The adaptation should become accurate as soon as possible.
3. **Type of trigger**. User based; the learning focus itself should be adjusted according to the behavior and motivation of the trainee during the scenario. The initial level of scaffolding should be determined according to the progress associated with the selected learning focus.

Observing the user

Online:

1. **Type of observation**. User-based; according to the functional component responsible for predicting the trainee, the trainee's actions in the environment should be observed.
2. **Intrusiveness**. The observation should be non-intrusive, because maintaining the concentration of the trainee is important
3. **Reliability**. Changing the level of scaffolding only inflicts small changes in the course of the scenario; therefore it is not necessary for the observations to be completely reliable. Additionally, the observations should success each other relatively fast, so if the level of scaffolding does not appear to be correct, it can be adjusted quickly.

Offline:

1. **Type of observation**. User-based; the adaptation is fueled by the trainee's behavior during the scenario. Also the trainee's motivation should be observed to infer the learning focus.
2. **Intrusiveness**. Since the trainee's behavior is already monitored online, this information can be re-used. Because the observation of the trainee's motivation can take place after the scenario is finished, it can be done in an intrusive way.
3. **Reliability**. The measurement of the trainee's behavior and motivation should be reliable, since both are important characteristics.

Adapting the system

Online:

1. **Form of adaptation.** The system configuration is changed by adjusting the level of scaffolding. The *scaffolding component* should be informed about this newly inferred level.
2. **Intrusiveness.** Maintaining the trainee's concentration during the scenario is important, therefore the adaptation of the scenario should be non-intrusive. Strictly speaking, the user model is not responsible for the adaptation done by the *scaffolding component*, but a low level of intrusion does infer that the level of scaffolding should not be changed too often.

Offline:

1. **Form of adaptation**. The content of the application is adjusted. The *scenario creator* and *scaffolding component* should be informed about the inferred learning focus and initial level of scaffolding.
2. **Intrusiveness.** According to the requirements of the AEG, the trainee should be aware of his progress and the changes and coherence between the consecutive scenarios, so the level of intrusion of this adaptation should be high.

7.2.2 User model in general

In addition to the general functional components, the properties of the overall user model can be further refined. Considering the **operability** of the user model, it operates both online and offline.

Additionally, the **incentive** needs to be defined. Online, the incentive of the user model is *active*. In a *reactive* way, the model should react to the observed behavior of the trainee. Offline, the incentive of the user model is both *passive* as *active*. When the scenario is finished, it is always necessary to process the trainee´s results, update the corresponding competences, and infer the next learning focus. Therefore, this can happen

in a passive and *procedural* way. Keeping track of the motivation of the trainee should happen in an active way. Due to the importance of the trainee's motivation, the user model should *pro-actively* monitor it. Especially when the motivation is low the user model should independently take appropriate action.

With the definition of the operability, incentive, and the necessary functional components, a high-level user model deliberation cycle can be defined. Figure 7.1 presents this cycle.

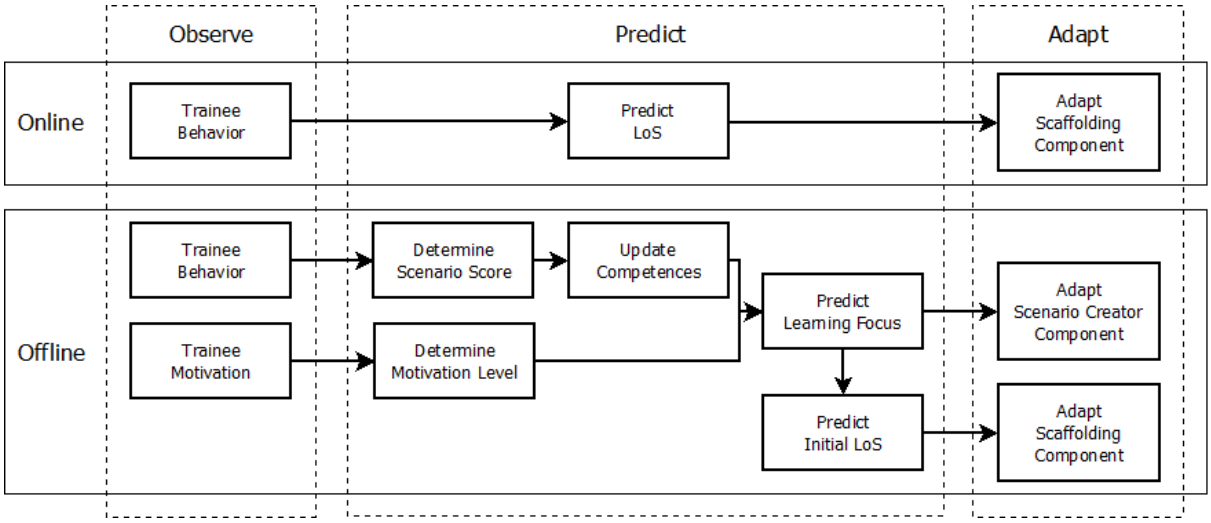

Figure 7.1: Deliberation cycle of the user model in an AEG according to the functional design

7.2.3 User profile

Lastly, the data used in the user model should be expanded on.

1. **User data readability.** One of the requirements of the AEG is that the trainer or supervisor has a certain amount of insight into the user data and general process that is going on in the user model. Therefore, it is preferable that the data in the user profile is easy to interpret.
2. **Privacy.** Data should be available for both the tutor and the student, in order to gain insight into the trainee's progress. Due to the sensitivity of the data, it is important that the information remains secret for other people.
3. **User profile persistence.** The data about the trainee should be saved as long as the trainee is registered with the program.

7.3 Technical design phase

In this section, the technical design phase of the user model will be discussed. The steps of the methodology are used to further refine the individual functional components, the user model as a whole, and the user profile. The techniques and exact algorithms are worked out according to the properties and desired behavior defined in the conceptual and functional design phase. Because the user model extends the Multi Agent Director System (MADS) as defined in Section 6.3, also the integration of the user model in the MADS will be taken into account.

7.3.1 Main user model functional components

Before the three main functional components can be implemented, several technical choices about the main methods and approaches have to be made. Due to interdependencies between the data, the user model as a whole and the different functional components, the information provided in this section was not constructed consecutively.

Predicting the user

Online, the level of scaffolding needs to be predicted. Due to the reactive nature of the desired adaptation a rule-based approach is used. In this way, input can be directly coupled to output. The rules are formalized according to the theory behind scaffolding, as mentioned in the domain analysis (Section 6.2). Although in the functional design phase the desired granularity of this prediction was defined as *optimized for the trainee*, due to the limited amount of computational power, as will be discussed in Section 7.4, the adaptation mechanism itself will not be personalized to the trainee.

Offline, the next learning focus and initial level of scaffolding need to be predicted. A (cognitive) model is used to infer these pieces of data. In the functional design, five consecutive sub functionalities were defined that are necessary to select the learning focus and initial level of scaffolding:

1. Determine the performance score of the previous scenario
2. Determine the trainee's level of motivation
3. Update the relevant competences according to the performance score
4. Predict the next learning focus
5. Predict the initial level of scaffolding

The internal mechanisms of, and the relations between these operations are formalized according to the ideas behind the Zone of Proximal Development (also mentioned in Section 6.2).

Observing the user

Online, the **user interactions** with the system are used as the main **method of observation.** This *indirect* way of obtaining user information results in a non-intrusive and reasonably reliable way of observing the user, as was desired according to the functional design. This method also results in somewhat *raw* user data, but because there are only three possible types of user actions in the MADS, no additional processing is necessary. The types that can be distinguished are:

1. The user performs the right action according to the scenario plan
2. The user performs the wrong action according to the scenario plan
3. The user waits too long before performing the right action and a time-out occurs

Offline, the user interactions with the system that were obtained online can also be used. To reliably measure the motivation of a trainee, a questionnaire is often used (Gazzaniga, et al., 2006). However, for reasons of simplicity and to keep the level of intrusiveness low, the trainee is asked to rate his current motivation by filling out one question. The motivation is measured on a Likert scale ranging between 1 and 9, which is a common way of operationalization. Thus, this information is obtained through **direct input from the user**. This *direct* way of obtaining information results in reliable data, but is also relatively intrusive. However, as defined before, the questioning takes place after the scenario ended, so this is not considered to be a problem.

Adapting the system

Online, whenever the level of scaffolding is adjusted, the newly inferred value should be sent to the *scaffolding component* to be processed and to take the appropriate actions. In the MADS, the level of scaffolding can have the following values: 0 (no support), 1 (some support), 2 (considerable support), 3 (full support).

Offline, the inferred learning focus and initial level of scaffolding is sent to the *scenario creator* and *scaffolding component*, which will use the information as the parameters to construct the next scenario.

General algorithms

Together with the desired properties identified in the conceptual and functional phase, the technical choices for the three functional components lead to the general algorithms that should take place in each (sub-) operation of the user model. In Appendix B, these algorithms are worked out. Due to the experimental nature

of the implementation, three parameters are left open in the algorithms, each related to the interpretation of the values obtained from the user:

1. Classification of a score; when is a performance score, or the progress score associated with a competence considered to be *good*, *reasonable*, or *bad*.
2. Level of motivation; when can the motivation be considered to be *high* or *low*.
3. The *minimum* and *maximum* amount of times a learning focus should be trained consecutively.

In a later stage, additional experiments should be performed to fine-tune the values of these variables.

7.3.2 User model in general

Considering the user model as a whole, its location and the computational power are taken into account.

The architecture of the existing MADS is **distributed**: each component is represented by a different agent that acts autonomously. The whole MADS operates on one system, which means that the available **computational power** is distributed among the different agents. For the operations that have to be performed offline, the user model can basically use as much computational power as necessary. However, online, the computational power is restricted, because the power needs to be distributed over all the agents that are active. Additionally, online, the MADS as a whole needs to act as fast as possible, in order to give adequate responses to the trainee while the trainee is playing the scenario. This further restricts the operations the user model can perform online.

In the MADS design, the *monitor agent* keeps track of the actions that are performed in the scenario environment. Additionally, the MADS requires that all the communication in the system goes through the central agent of the application: the *director agent*. Therefore, to efficiently use the computational power available online, and to avoid an excessive amount of messages between the *monitor agent*, *director agent*, and user model, the user model operations that are necessary online will take place in the *monitor agent*. To avoid any unnecessary messages containing information obtained online, the operation responsible for calculating the performance score, which is conceptually a part of the *offline* user model, will also take place in the *monitor agent*. The rest of the considerably more expensive user model operations that are necessary offline will be represented in a *user model agent*.

Splitting the user model into two parts is contradictory to the software decomposition requirement of developing a user model, which stated that there should be a strict separation between the user model, and the components belonging to the rest of the system. Due to the restrictions imposed by the existing MADS, the splitting of the user model is a reasonable solution.

7.3.3 User profile

Considering the associated user data in the model, the main data types and data structures, and the location of the user profile are taken into account.

Influenced by the existing implementation of the MADS, the main **data type** that is used in the user model are *predicates*. Each predicate captures the category of the variable, and one or more values. Due to this declarative nature, the user data is readable and interpretable for a human. To present the data to external users such as the trainer, the predicates can easily be translated into a more intuitive format such as a graphical display.

The main **data structure** that is used in the user model represents the competences associated with First Aid. These competences are structured in a tree. The tree that is used comprehends a subset of the skills needed to perform First Aid, encompassing social skills required in First Aid. The tree was formalized by (Peeters, 2014a). It can be found in Figure 7.2. Each node in the tree represents a competence and contains a score presenting the trainee's progress. Each node can have zero or more children, where a child represents a sub-competence that should be mastered before the main competence can be trained. For instance, in order to perform the competence 'communicating', the sub-competences 'question' and 'report' should have been acquired.

The main invariant of this tree is that when each child of a node has a score that can be classified as *good*, the main competence has a score that can at least be classified as *reasonable*. This means that the main competence can now be trained. This invariant is used to simplify the operations that can take place on the tree.

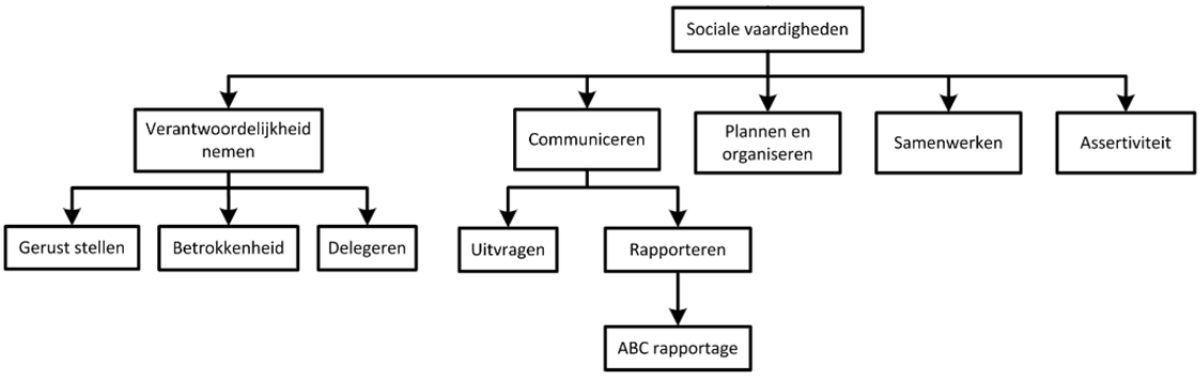

Figure 7.2: data structure representing competences in First Aid, based on (Peeters, 2014a)

Finally, the **storage location** of the data is in a central place. Because the complete system is *closed* - only the trainee and trainer can access its contents, **privacy** is not regarded to be a major concern.

7.4 Implementation using the framework

In this section, the BDI-based intelligent agent framework is used to implement the user model with the use of the agent programming language `2APL'. With the general technical properties and choices fixed, the user model can be implemented. The implementation is guided by the agent-based user model framework. As explained in the previous section, the user model is split up into two components. The *monitor agent* is extended to house the operations necessary to add online personalization, and a *user model agent* is introduced to execute the necessary offline operations.

Since the main structure of the user model is determined by the framework, first the technique chosen to implement the framework, or more specifically the BDI paradigm, will be discussed. Next, the progress is described that leads from the framework to the implementation. Finally, to illustrate the integration of the user model with the MADS, the new deliberation cycle is presented.

Implementation technique

To implement the user model with the help of the framework, the BDI agent programming language 2APL is used (Dastani, 2008; Dastani, 2009). 2APL stands for *A Practical Agent Programming Language.* The language provides an effective and efficient cognitive agent architecture that supports the implementation of both reactive and pro-active agents. 2APL contains important programming principles and techniques, such as recursion, compositionality, abstraction, and exception handling. Additional software tools and environments are available for developers (Dastani, 2013). The belief base of a 2APL agent uses JIProlog, a simplified version of Prolog, to handle data and inference rules (Chirico, 2013). The environment in which the agent operates must be written in Java.

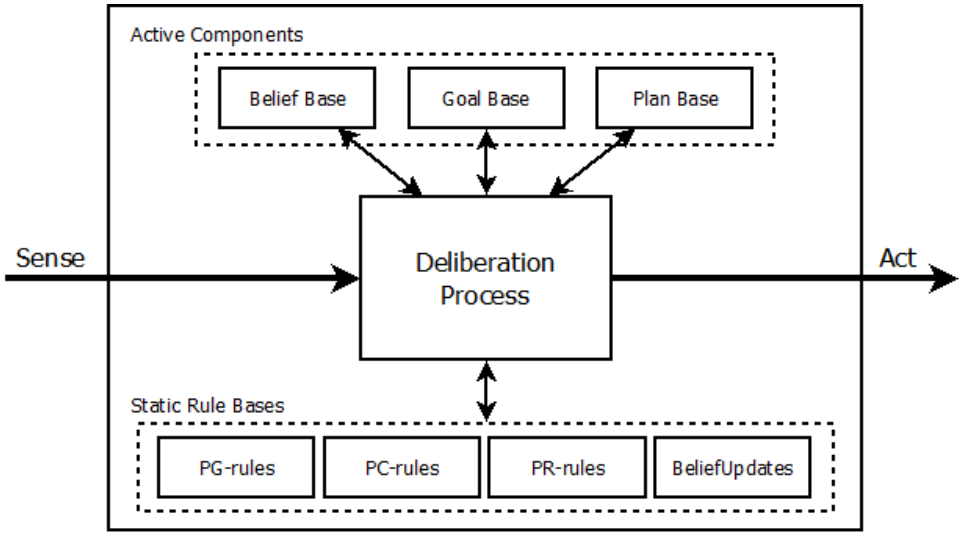

Figure 7.3: 2APL architecture, adapted from (Dastani, 2009)

The 2APL language contains several components that together enable the BDI-mechanism. The different components of the 2APL architecture can be found in Figure 7.3. The **Belief Base** contains the current *beliefs* of the agent, the **Goal Base** contains the current *desires* of the agent, and the **Plan Base** contains the current *intentions* of the agent - or to be more concrete: the exact actions that are going to be executed. Besides these dynamic components, a 2APL agent contains several static rule bases, consisting of plans that can be selected and added to the Plan Base to be executed. Each rule base has its purpose in the realization of the BDI-paradigm:

- **PG-rules (Planning Goal Rules)**. To achieve the desires (goals) of the agent, a planning goal rule can be selected. Through PG-rules, the agent is not so much responding to its surroundings but instead is acting because of an internal drive to act.
- **PC-rules (Procedure Call Rules)**. A procedure call rule can be used for three things:
    - handle an incoming message from another agent
    - handle an event generated by the external environment
    - execute an abstract action: a procedure in which multiple basic actions (see below) are encapsulated

  From the above, it can be established that PC-rules are responsive in nature, are used to handle all the incoming communication, and are used to refine previously adopted (abstract) plans.
- **PR-rules (Plan Repair Rules)**. If the execution of one of the agent´s plans fails, a plan repair rule can be used to replace the failed plan.
- **BeliefUpdates**. A BeliefUpdate rule enables a direct update of the Belief Base. It consists of a function name, a pre-condition and a post condition. When a BeliefUpdate rule is invoked, if the pre-condition can be deduced from the Belief Base, the post-conditions are executed.

Within each plan, there are several **basic actions** that can be performed:

- **Communication action**: send a message to another agent
- **External action**: execute an action in the environment
- **Abstract action**: call upon an abstract action in the PR-rules
- **Belief update action**: call upon a BeliefUpdate rule
- **Goal dynamics action**: update the goal base
- **Test action**: test if an expression is derivable from the Belief/Goal Base

The different components of a 2APL agent work together according to the deliberation cycle visualized in Figure 7.4. Whenever a rule is applied, its plan is added to the Plan Base.

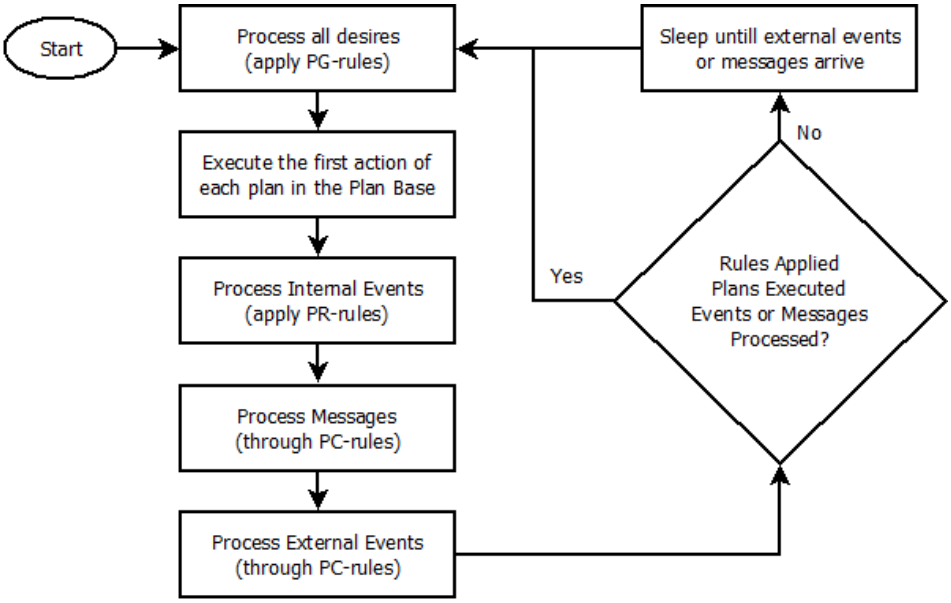

Figure 7.4: The 2APL deliberation cycle, adapted from (Dastani, 2008)

7.4.1 From framework to implementation

With the definition of the language that will be used to implement the framework, the implementation of the user model can continue. In this section, first the general translation of the framework components to the desired components of the user model is discussed, while taking into account the features and restrictions of 2APL. Second, the personalization cycle of both the online and offline user model is discussed, illustrating the resulting implementation. For further reference and more details, the complete implementation of the user model in 2APL can be found in Appendix C. The three components of the BDI-agent are refined in the following way:

Beliefs. The user characteristics that were defined in the methodology, such as the competence tree, are stored and maintained in the agent's Belief Base. In 2APL, it is not possible to define multiple Belief Bases to distinct the different kinds of (user) data. Therefore, in order to distinguish the different types of data specific predicates are used.

Desires. Depending on the incentive of each necessary operation of the user model, the *desires* in the Goal Base define when an operation should be executed. In this implementation, desires are especially used to instigate pro-active behavior, producing a drive for the agent to engage in behavior independent of incoming events (see also *intentions* below).

Intentions. Within the agent-based user model framework, the necessary operations of the user model are defined in the *intentions* of the agent. The possible intentions of a 2APL agent are represented in the plans contained in the rule bases of the agent. In these plans, each *basic action* is possible to execute. This means that the necessary operations from a user model can be expressed in all types of plans. Therefore, which rule base to choose to express the user model operation in, depends on the incentive of the user model:

- When proactive, goal-oriented behavior is desired, a PG-rule is used; PG-rules are directly triggered by the agent's *desires*.

- When reactive or procedural behavior is necessary, a PC-rule is used. PC-rules are mostly used to handle external communication. In this way, incoming requests and user data can be directly processed, without using unnecessary and devious programming constructs.

According to the main translation of the framework and the desired user model, the resulting deliberation cycles are now presented, adapted from Section 5.2.2.

Online: extending the monitor agent

The part of the user model that is integrated into *the monitor agent* is responsible for predicting and adjusting the level of scaffolding. The system components that can be adjusted and that can be used to obtain user information are already integrated into the *monitor agent*. The exact implementation of the online user model can be found in Section C.1 of Appendix C. The personalization cycle looks as follows:

1. The *monitor agent* observes the trainee´s action while the trainee runs the scenario.
2. In a procedural way (through a PC-rule), the incoming actions of the trainee are classified as correct or incorrect, according to the predefined action plan. Time-outs are detected and classified in a pro-active way (through a PG-rule); a desire is constantly active to keep track of the amount of time the trainee uses. The classification is used to predict the correct level of scaffolding, according to the algorithm defined in Appendix B. Whenever an action is performed or a time-out takes place, it is recorded in a logbook for future use.
3. The level of scaffolding is adjusted in the *monitor agent*. In the MADS, the choices available for the scaffolding component are integrated in the action plan of the scenario. Since the *monitor agent* keeps track of this plan, the agent can directly take the appropriate action.

Offline: a user model agent

The *user model agent* is responsible for predicting the learning focus and the initial level of scaffolding for the next scenario. The exact implementation of the *user model agent* can be found in Section C.2 of Appendix C. The personalization cycle looks as follows:

1. Online, the *monitor agent* collects data about the trainee's behavior. This data is saved in a logbook. After the scenario is finished, the *monitor agent* calculates the performance score and asks the trainee to rate his motivation. All this information is sent to the *director agent*, which sends it to the *user model agent*.
2. In a procedural way (through several PC-rules), the *user model agent* updates the competence tree, selects the next learning focus, and infers the initial level of scaffolding. In a pro-active way (through an explicit desire and accompanying PG-rules), the *user model agent* keeps track of the trainee's motivation. In accordance with the resulting level of motivation, a different learning focus can be chosen. Again, the individual user model operations are defined according to the algorithm in Appendix B.
3. The user model communicates the inferred data to the *director agent*, which sends it to the *planner agent*. In turn, the *planner agent* uses the data to construct the next scenario.

7.4.2 Resulting deliberation cycle

With the implementation of the user model, the deliberation cycle of the MADS, as defined in Section 6.3.2, was also adjusted. The new deliberation cycle is visualized in Figure 7.5 and consists of the following steps:

1. When the application is launched, all the agents start by registering with the director
2. At the same time, the director asks the user model to infer the data necessary to initialize the next scenario. The user model infers the next learning focus
3. According to the selected learning focus, the initial level of scaffolding is inferred by the user model. Both the learning focus and the level of scaffolding are sent back to the director

4-6 Step 4 – 6 correspond to step 2 - 4 from the original cycle. The next learning focus and initial level of scaffolding is used to construct the scenario plan. The next scenario is initialized and started as soon as all the components are in place

7. The trainee plays the scenario, where the action plan determines which action should be performed. The trainee is monitored by the *monitor agent*, who also fulfills part of the role of the user model.
   7.1. Whenever the trainee misses an action or waits too long, the scenario is adapted; the level of scaffolding is changed and the director is notified to execute a certain intervention, depending on the level of scaffolding and action plan
   7.2. The director starts an auction to let the most believable *character agent* perform the intervention (either giving a hint or performing the action)
8. When all the actions of the action plan are performed, the scenario ends. The scenario score is calculated and additional information about the motivation of the trainee is asked. All this information is sent to the director
9. The director sends the obtained user data to the user model, which updates the information about the trainee. The MADS is reinitialized to host the next scenario
10. If all competences have been trained well enough, the application ends, otherwise a new learning focus is selected, and the cycle returns to step 2

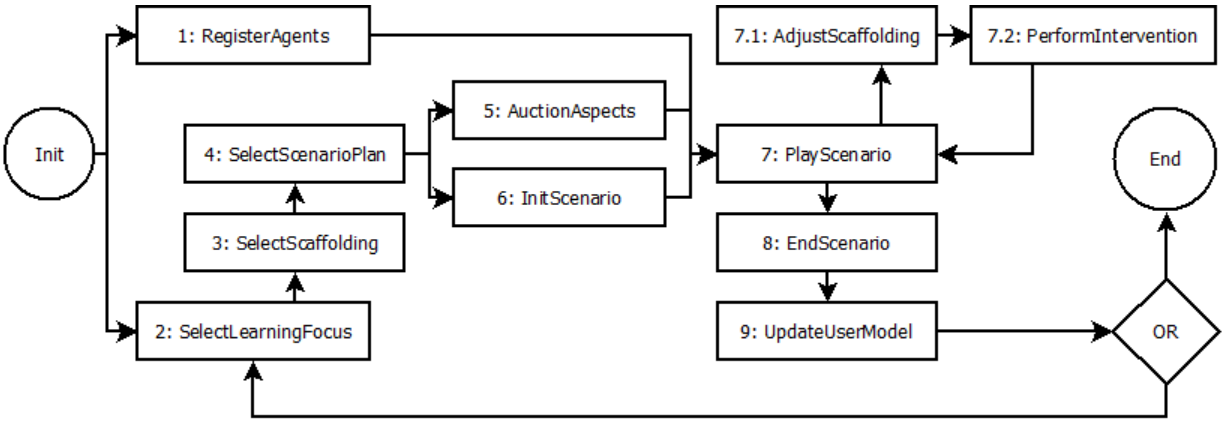

Figure 7.5: The revised MADS deliberation cycle

7.5 Verification

In the previous sections, a user model for an AEG has been designed, with the use of the general user modeling methodology and the agent-based user model framework. In this section, the behavior of the implementation is tested by means of *use cases* to verify that the resulting user model displays the desired behavior as set out in the domain and task analysis.

A use case is a list of steps that define the interactions between an actor and the system to achieve a certain goal. In this case the actor is the trainee and the system is the Multi-Agent Director System (MADS) extended with the user model. The different use cases in which the user model is involved are worked out in detail and executed while working with the system. The resulting behavior of the system is observed and compared to the desired behavior, to verify that the user model works in the desired way.

Since this verification only focuses on the operations the user model should perform, the assumption is made that the rest of the system behaves according to the system specification. For example, it is assumed that the planner agent constructs correct scenario plans, and that an adaptation of the level of scaffolding results in the intended effects.

7.5.1 Use case design

Just as in the rest of this chapter, the use cases are divided into an online and offline part. Especially for the offline part, it is a complex task to construct a thorough walkthrough of all the possible use cases leading to

the different behavior of the user model. However, as seen before, the whole offline process can be divided into multiple (sub-) operations:

1. Determine the performance score
2. Determine the trainee's level of motivation
3. Update the competence tree structure
4. Predict the next learning focus
5. Predict the initial level of scaffolding

Therefore, to simplify the verification, each operation is examined separately. Operations 1 and 2 are focused on the interactions of the trainee and the system, whereas operations 3, 4 and 5 are focused on the processed data given as output by the previous operations. Because the term *use case* is no longer applicable to the walkthrough of the possible combinations of inputs for these operations, the term *parameter set* is used instead.

The exact details and the execution of each use case and parameter set is worked out in Appendix D. The appendix also describes the exact procedure and input that was used for executing each use case or parameter set, so they can be reenacted when necessary.

7.5.2 Examined use cases and parameter sets

For each (sub-) operation of the implemented user model, one or more use cases or parameter sets are constructed. Each use case or parameter set is influenced by the user data that is gathered during the online part of the deliberation cycle, consisting of the user's actions and motivation. The obtained user data is directly used and processed in use case 1, 2, and 3. The resulting outputs of these operations are used as input for use case 3, 4, and 5. For each operation, the use cases or parameter sets that are investigated are set out in the section below.

Adjust the level of scaffolding

The level of scaffolding changes when an action is performed in the correct or incorrect way, or when a time-out takes place. Therefore, the following three use cases are examined:

1. the trainee performs the right action
2. the trainee performs the wrong action
3. a time-out occurs

Determine the performance score

The use cases examined for this operation describe the manner in which particular types of concrete trainee behavior should result in a particular performance score. The overall performance score is calculated by taking the average of the scores obtained for each individual action. To simplify the verification of this operation, in the first three use cases an action plan is used containing only one action, covering all the base cases. The fourth use case examines the calculation of the performance score when the action plan contains more than one action. The examined main use cases are:

1. the trainee performs the action in the right way
2. the trainee performs the action after a time-out
3. a non-player character performs the action
4. the action plan contains multiple actions

For each main use case, also the number of mistakes and whether or not a time-out took place is taken explicitly into account. The resulting performance score is used as input for operation 4 and 5.

Determine the trainee's level of motivation

At the end of the scenario, the current motivation of the trainee is requested. Together with the average of the previously registered motivation score, this concrete evaluation of the trainee's motivation is used to infer the more abstract *level of motivation*. Two use cases are considered:

1. the trainee indicates that his motivation is *low*
2. the trainee indicates that his motivation is *high*

The resulting level of motivation is used as input in operation 5.

Update the competence tree structure

The competence tree is updated based on the performance score obtained in operation 2. The main parameter sets investigated in this verification are categorized according to the possible classes of performance scores:

1. the performance score is *bad*
2. the performance score is *reasonable*
3. the performance score is *good*

Additionally, a distinction is made between *node* competences – competences that have underlying competences, and *leaf* competences - competences that do not have underlying competences.

Select an appropriate learning focus

The learning focus can either remain the same or it can be a new learning focus. The main parameter sets considered in this verification are:

1. the previous learning focus has been trained less than 3 times
2. the previous learning focus has been trained between 3 and 5 times
3. the previous learning focus has been trained 5 or more times

Each main parameter set also explicitly takes into account the trainee's level of motivation and the average performance score obtained for the current learning focus thus far.

Select an appropriate initial level of scaffolding

The performance classification associated with the selected learning focus (good, reasonable, or bad) is used to determine an appropriate initial level of scaffolding.

7.5.3 Results

For each use case and parameter set, the expected and resulting behavior of the system is written out in Appendix D. The expected functional behavior and the observed functional behavior exactly coincide in almost all the use cases and parameter sets. Only in the parameter sets related to updating the competence tree, some minor rounding errors were detected. These are probably caused by the JI Prolog engine that is used in 2APL. Nonetheless, it can be concluded that the implemented system performs exactly as specified in the domain and task analysis, and in the conceptual user model design

7.6 Conclusion

In this chapter, the general user modeling methodology and the agent-based user model framework have been used to implement a user model for an Adaptive Educational Game (AEG) in First Aid. Throughout the sections, the user model was gradually developed, starting at a very abstract level, and resulting in the very concrete implementation. To verify that the resulting system displayed the desired behavior, multiple use cases and parameter sets were constructed and executed. The system acted in the exact same way as was described in the initial system specification, i.e. the conceptual, functional, and technical designs. The worked

out and implemented processes of the user model are expected to result in a progressive competence development loop.

Extended by the implemented user model, the Multi-Agent Director System (MADS) can be seen as a prototype for an AEG in First Aid. In order to establish that the AEG indeed results in effective learning, further research is required. User-based experiments should be conducted to validate the usefulness of the complete system. In the same manner, experiments should be performed to fine-tune the parameters that were defined in the user model (see Section 7.3.1 – General Algorithms), such as the classification of performance scores.

For the user model itself, future research can be conducted to the AEG requirements that were not investigated in the implementation. No focus was placed on the role of the user model in providing tailored feedback (requirements R13, R14) and the influence and control of the trainer on the user model (requirement R17). The user model can also be further improved by extending the defined algorithms. For instance, by taking into account additional user characteristics. To investigate whether more intricate algorithms would improve the behavior of the AEG, more experimentation would be required. Due to the modular structure of the existing user model, the model can be easily expanded to contain the additional requirements and features.

The application of the methodology and framework while developing a user model for an AEG in First Aid also resulted in several findings, observations, and experiences regarding the applicability and usefulness of the methodology and the framework. These results will be discussed in the next and final chapter; the discussion and conclusion.

Chapter 8

Discussion and Conclusion

This final chapter presents the discussion and conclusion of the master thesis. The discussion describes the results and findings of the proposed general user modeling methodology and agent-based user model framework, by looking at the worked out implementation of an adaptive educational game in First Aid. Additional directions for future research are proposed. The conclusion completes the thesis by summarizing the research that was done and the results that were obtained.

8.1 Discussion

In this thesis, adding personalization to a system by using a user model has been investigated. In order to aid developers with the complex process of user modeling, a general user modeling methodology has been constructed. The purpose of this methodology was to propose a dynamic method that captures the essence of user modeling and bridges the current gap between domain-specific and (too) general user modeling methods. Supplementary to this methodology, a BDI-agent-based user model framework has been proposed, to investigate the usefulness and effectiveness of basing a user model on the intelligent agent paradigm.

To construct both methods, first a thorough review of the user modeling field was performed, which was presented in Chapter 2. Based on this review, the technical requirements, main application domains, and existing methods of user modeling were taken into account, and used to construct a more abstract overview in Chapter 3. Although a possible point of discussion, we assume that enough application domains were investigated to identify the most important underlying properties, such as the main functional components, properties and possible solutions of user modeling. Based on the abstract overview of user modeling, both the methodology and framework were constructed in Chapters 3 and 5. Finally, the methodology and framework were validated and tested by applying them to the problem of adding personalization to an adaptive educational game (AEG) in First Aid. The application of both methods resulted in several findings, observations, and experiences regarding their applicability and usefulness.

8.1.1 General user modeling methodology

During the implementation of the user model for an AEG in First Aid, it became clear that several steps and properties were missing in the methodology, while other properties turned out to be useless. In general, for each design phase, it was necessary to add additional (sub-) steps to make the process between the different phases clearer. In this way, the separation between the conceptual ideas and the technical details became stricter. For instance, in the functional design the need for a high-level deliberation cycle was introduced, and in the technical design, a step was added to define general algorithms before moving on to the actual implementation. In addition, several (technical) properties were excluded, such as the *format* and *origin* of the user data, which turned out to be too low-level for the methodology.

The methodology was adjusted to integrate the missing steps and properties. Care was taken to ensure that the changes made to the methodology did not make the methodology too much focused on the problems associated with AEG. We presume that the methodology is still applicable to other domains, but to be sure of this, the methodology should also be applied and tested on other user modeling domains (see Future work).

In addition to the necessary adjustments, several other observations were made during the employment of the methodology. The separation between the conceptual, functional, and technical design phase resulted in a clear distinction between the ideas behind the personalization, and the technical implementation of the user model. By defining the properties that do not specifically belong to either the conceptual or technical design

in the functional design, the transition between both phases proceeded smoothly, in contrast to the existing user modeling methods discussed in Section 2.3.

However, the strict separation also created several problems during the implementation of the user model. Sometimes it was hard to keep an overview of the complete system, causing the usability and clarity of the methodology to dwindle. For example, during the refinement of the general algorithms that were necessary to solve the personalization problems, it sometimes was hard to keep track of the conceptual background and desired properties on which the algorithm should be based. This problem could mean that the methodology still fails to clearly capture the interrelations between the different design phases, and thus the conceptual ideas and technical solutions. Therefore additional refinement of the methodology is necessary.

The methodology is designed in such a way, that it should be useful for both current and future personalization problems. To be more concrete, it should be applicable to the application domains discussed in the user modelling review presented in Chapter 2, but also to domains that were not part of the literature review. For this purpose, the methodology is meant as an *approach* to solve the problem of personalization. Rather than giving all the concrete options and solutions for personalization, an extendible combination of high-level properties, possible options and guidelines was defined. As such, the methodology can be extended ; an investigation of other application domains could lead to the addition of properties and values to the methodology.

Consequently, to keep the methodology organized, for certain problems – such as the delicate matter of privacy, only references were added to additional literature. The drawback of this approach is that the methodology is not directly applicable for a novice. The underlying properties and choices need additional study before they can be applied.

8.1.2 Agent-based user model framework

During the implementation of the user model for an AEG in First Aid, the effectiveness of the agent-based user model framework was tested as well. The framework is designed to be applied to complex personalization problems. Due to the complexity of AEG, the framework proved to be very suitable for implementing the user model for an AEG in First Aid.

Resulting from the implementation, additional, more specific findings can be discussed. The BDI paradigm demonstrated to be effective for the construction of a user model. Because the BDI paradigm imposes the clear distinction between the beliefs (data), desires (initiative), and intentions (operations) of the agent, a strict separation between the data and the functionalities used in the user model was possible. This modular, decomposed design also resulted in a very expandable user model. Additionally, due to the restrictions the BDI paradigm imposed upon the user model, poorly organized constructions and mistakes could be avoided.

Considering the initiative of a user model, the BDI paradigm is especially effective for pro-active (goal-oriented) behavior. However, when simpler, reactive or procedural behavior is sufficient, the BDI paradigm might be too complex, and therefore superfluous. When a combination of pro-active and procedural or reactive behavior is necessary, BDI is a good solution. Folk psychology, the theory behind the BDI paradigm, is often also too complex for building a user model, except when the user model processes are based on a (cognitive) model of human behavior. Altogether, because pro-active behavior is not always necessary in a user model, to make the intelligent agent paradigm applicable for a wider range of personalization problems, other, less complex structures could be investigated in addition to BDI.

Although of less importance, another point of discussion for the framework is related to the agent programming language that was used: 2APL. Due to the interpretation of the BDI-paradigm by 2APL, expressing certain features of the framework was hard and just not very graceful. For instance, it is hard to efficiently add external components to the system to do specific (for example heavy computational) operations. Other BDI-based agent programming languages could have been used, such as GOAL, JACK, or SPARK (Morley, et al., 2004), but the main problem is that none of these languages is optimized to use for user

modeling. Therefore, one of the points of future work (see below) will be concerned with adjusting 2APL, or introducing a new, better suited BDI-based agent programming language.

8.1.3 General remarks

Considering both the methodology and the framework, we have seen that they are especially applicable to more complex personalization problems. Due to their complexity, it is not always necessary to apply them to a user modeling problem. As discussed in the user modeling review, it often happens that the user modeling problems is relatively easy and straightforward, which means that using these methods could be needlessly complex, and therefore even counterproductive. An example of the unnecessary complexity of the resulting methodology is the running example used in Chapter 3. The personalization that is chosen turns out not to be very complex, but the complete working-out of the methodology does become comprehensive.

Nonetheless, the methodology and framework do present a way of creating a user model in a systematic and traceable way, which is often important for scientific research. Considering more common solution-oriented development projects, the proposed methods are especially useful for complex projects where it is hard to get a grip on the problem.

8.1.4 Future work

For both the general user modeling methodology and the agent-based user model framework, several directions for further research can be suggested. As discussed above, the design and implementation of the user model for an AEG in First Aid influenced the properties and features of the methodology. Therefore, the most prominent point of future work is to further validate and possibly expand the methodology and framework by applying them to additional personalization problems and domains, and to compare the methodology to the use of alternative methodologies through user-based studies.

Future work on the general methodology can mostly be related to expanding the methodology. The proposed methodology already captures the – in our view, most important high-level steps, properties, and choices that should be taken into account when creating a user model, but each step, property and option has the potential to be expanded. For instance, additional guidelines can be defined or the many differences between the possible reasoning approaches could be worked out. In addition, the agent-based user model framework is only one way to implement the user model. The methodology can be expanded by taking into account other frameworks and architectures, such as the domain-specific methods discussed in Section 2.3. When expanding the methodology, the usability and clarity of the methodology should always be taken into account.

Another direction for future work on the methodology is the development of a support tool for applying the methodology. As observed during the implementation of the user model for an AEG in First Aid, due to the many steps in the methodology and the interrelations between the different possible properties, applying the methodology can become complex. A tool could be developed that captures the steps, possible choices and resulting restrictions. This tool would also be useful for maintaining a clear overview when expanding the general methodology.

Considering future work on the agent-based user model framework, instead of the BDI-paradigm, there are also other, less complex, agent architectures available for developing a user model. For instance (model based) reflex agents. For complex problems, besides BDI, also other cognitive agent architectures are worth investigating, such as the Act-R and Soar architecture (Heuvelink, 2009). All these approaches have the same advantages that intelligent agents present for a user model (autonomy, pro-activeness, and a distributed design), but different, possibly useful, internal mechanisms.

Additionally, as discussed earlier, the problems that arose during the implementation of the user model were especially related to the 2APL implementation language. Future work can be focused on expanding the 2APL language to more closely match the necessary requirements of the user model framework. A new, more suitable BDI-based agent programming language could also be developed. According to the proposed

framework, several key problems of user modeling can be directly addressed in this new or adjusted language. Requirements and possible features of this language would be:

- The availability of multiple Belief Bases to make a clear distinction between the three types of data (user profile, temporary user-related data and non-user related data). This feature also involves the possibility of saving the data of multiple users in the same agent.
- Subdivided rule bases that represent the possible intentions of the agent, based on the three functional components of an agent-based user model (process the incoming data, predict the user, communicate the adaptation).
- Enhancement of the applicability of the Goal Base (desires) of the agent, in order to make goals more useful in reactive and procedural user model processes.
- A more modular approach, where it is possible to execute certain actions in external components. For instance, the possibility to let computational operations be handled by an external component that is more suitable for computation.

8.2 Conclusion

In this thesis, the process of adding personalization to an application with the use of a user model has been investigated. The main question was:

How can the process of adding personalization to an application with the use of a user model be supported?"

To answer this question two subsequent questions were asked:

1. *How can the gap be bridged between high-level and low-level user modeling methods?*
2. *How can the intelligent agent paradigm be used to develop a user model, how effective is a user model based on a BDI-agent, and in what type of applications would a BDI-based user model be useful?*

To answer the first question, a dynamic general user modeling methodology was proposed, capturing the essence of user modeling. In this methodology, a strict separation was made between a conceptual, functional, and technical design phase. The ideas from the general (high-level) user modeling methods were captured in the conceptual design phase. The ideas from the domain-specific (low-level) user modeling methods were captured in the technical design phase. The functional design phase was introduced to bridge the gap between the conceptual and technical phase, by handling the user modelling properties that do not specifically belong to either the conceptual or technical design.

The methodology was applied to the design of a user model for an AEG in First Aid. Through gradually building the desired user model, the problems and possibilities of personalization were steadily narrowed down. Although further research is needed, the methodology did prove to be a thorough method of designing a user model. The methodology enabled to make a clear distinction between the ideas behind the personalization and the technical implementation of the user model, proving the value of this approach to user modeling.

To answer the second question, an agent-based user model framework, using the BDI architecture, was proposed. The framework combines the user modeling domain with the intelligent agent paradigm. Just like the general user modeling methodology, the framework was validated by applying it to the process of implementing a user model for an AEG in First Aid. Because of the resemblance between the personalization cycle that takes place in a user model, and the deliberation cycle of an agent, the framework proved to be a useful way of capturing the processes and features of a user model. The abstract features, functional components and properties of a user model, as defined in the general user modeling methodology, were all expressible in the framework.

The BDI architecture that was used for the internal mechanisms of the agent-based user model framework, proved to be useful to depict the coherence, and make a distinction between the functional components, the data, and the general deliberation cycle of the user model. Furthermore, the BDI architecture enforced additional restrictions upon the system, which meant that common mistakes and poorly organized constructions could be avoided.

Due to the possibility of autonomous, flexible, and robust behavior of an agent, the BDI-agent-based user model framework is especially useful for complex personalization problems, where difficulties such as incomplete data and unpredictability of the user play a role.

At the start of this project, the main difficulty was to get a grip on the user modeling domain, due to its broad applicability and complexity. Through the investigation of the user modeling domain and the construction of the methodology and framework, the domain of user modeling has been set out more clearly. This resulted in insights and knowledge about the problem of personalization and its complexity. However, it does not mean that the problem is solved: there are still many points of interest for future work. Hopefully, this thesis will serve as a (first) step for researchers, designers, and developers to get a better grip on their user modeling problem.

Appendices

Appendix A : General user modeling properties overview
Appendix B : User model operation algorithms
Appendix C : User model source code
Appendix D : Adaptive Educational Game Use Cases

Appendix A

General user modeling properties overview

This appendix presents a structured overview of the properties associated with user modeling across the different application domains. The functional and technical properties of the following components are provided:

1. User model functional components
   1.1. Observing the user
   1.2. Predicting the user
   1.3. Adapting the system
2. General user model
3. User profile

A.1 User model functional components

A.1.1 Observing the user

Conceptual properties	
Type of observation	The type of observation done by the user model • user-based observation • system-based observation • environment-based observation • task-based observation • spatiotemporal observation • …
Intrusiveness of observation	The level of intrusion the observation inflicts upon the user, ranging between intrusive and non-intrusive
Reliability of observation	The truthfulness of the observed data, ranging between unreliable and reliable
Technical properties	
Method of observation	The method for observing the user information • user interaction in the system • direct input from the user • (external) sensors • …

A.1.2 Predicting the user

Functional properties	
Prediction model granularity	The level of detail of the model • optimized for one individual • optimized for a group of people
Prediction model accuracy	The amount of data necessary to give a good prediction
Type of trigger	The type of trigger that is used in the prediction model: • user-based trigger • system-based trigger • environment-based trigger • task-based trigger

	• spatiotemporal trigger • …
Technical properties	
Prediction approach	The method in which prediction takes place • rule-based approach • computational approach • (cognitive) model approach

A.1.3 Adapting the system

Functional Properties	
Form of adaptation	The way the system is adapted • data structure • data content • data presentation • system configuration
Intrusiveness of the adaptation	The amount of intrusion the adaptation inflicts upon the user • intrusive • intermediate level of intrusion • non-intrusive

A.2 Overall user model

Functional properties	
User model incentive	The amount of initiative the user model undertakes • passive (or procedural) • active (pro-active or reactive)
User model operability	The moment in which the user model is active • online • offline • online and offline
Technical properties	
Storage location of user model	Physical location of the user model • centralized in one location • distributed throughout the application: o client-side of application o server-side of application
Computational power	Amount of computational power available for the user model

A.3 User profile

Functional properties	
Expressiveness (Description of user information)	The meaning and purpose of the saved data
Privacy	The accessibility of the user data • private • public • distinction between different entries or different users/systems
Readability	Interpretability of the saved data for the programmer or end-user • readable • not readable

User profile persistence	The period the user profile is saved • temporary (for one session, as long as the user uses the system, …) • for ever / undetermined • …
Technical properties (applicable to each separate data entry)	
Data type of user data entry	Concrete data type of user information • string • int • Boolean • …
Storage location of data entry	Physical location of data • centralized in one location • distributed throughout the application: o client-side of application o server-side of application

Appendix B

User model operation algorithms

In this section, for each identified (sub-) operation of the user model for an AEG in First Aid, a high-level algorithm is worked out. Each algorithm defines the input and the steps that lead to the desired output. The operations that are worked out, are:

1. Adjust the level of scaffolding (online)
2. Determine the performance score
3. Determine the trainee's level of motivation
4. Update the competence tree structure
5. Select the next learning focus
6. Select the initial level of scaffolding

The exact implementation of these algorithms can be found in the user model program code, as worked out in Appendix C. Operation 4 and 5 are influenced by the competence data structure. More details about this chosen data structure can be found in Section 7.3.3.

B.1 Adjust the level of scaffolding (online)

Input: the trainee behavior in the environment
Output: the adjusted level of scaffolding (LoS)
Algorithm: how the LoS is adapted depends on the type of action the trainee performs, according to the predefined action plan of the scenario:

1. The user performs the right action: the LoS decreases by 1
2. The user performs the wrong action: the LoS increases by 1
3. A time-out occurs: the LoS increases by 1

B.2 Determine the performance score

Input: the list of the necessary actions from the action plan, and for each action: the amount of mistakes made, the time-outs that took place, and the LoS during each time-out
Output: the performance score of the scenario, concretized in a value between 0 and 100.
Algorithm: Whenever the trainee did not make any mistakes and no time-out(s) occurred, the performance score should be maximal (100). When the number of mistakes increases, or when one or more time-outs occur, the performance score decreases. The LoS during the time-out, and thus the chosen intervention, determines the maximum score that is possible for completing a certain action. The exact relation between the number of time-outs/mistakes and the performance score should be determined based on experimentation or with the use of a machine learning algorithm, such as reinforcement learning.

B.3 Determine the trainee's level of motivation

Input: current and previously registered motivation score
Output: the trainee's level of motivation, categorized as either *high*, or *low*
Algorithm: to keep things simple, the average of the old and new motivation is taken and classified according to the classification provided in Section 7.3.1 – General Algorithms.

B.4 Update the competence tree structure

Input: trained learning focus, performance score, competence tree
Output: updated competence tree
Algorithm: the performance score represents the trainee's progress related to the previous trained learning focus, therefore the competence(s) associated with the learning focus in the tree will be updated. The score can be categorized into three classes, each with their distinct consequences for the update that should take place:

- The score is ***good***: proportional to the performance score, the overall score of the tree increases. Especially the overlying competences are affected, making it more plausible that they are selected.
- The score is **reasonable**: proportional to the performance score, the overall score of the tree increases. The learning focus is likely to be trained again.
- The score is **bad**: proportional to the performance score, the overall score of the tree decreases. Especially the underlying competences are affected, and might be needed to be retrained.

B.5 Select the next learning focus

Input: trainee´s level of motivation, number of times the current learning focus has been trained, average of the performance scores gained for the current learning focus
Output: next learning focus
Algorithm: the learning focus stays the same, if:

- The current learning focus is trained less than the minimal designated times it should be consecutively trained
- The current learning focus is trained more than the minimal, but less than the maximal designed times, the trainee's motivation concerning the learning focus is still *high*, and the average performance score is still too low (*bad*)

Otherwise, a new learning focus is selected by inspecting the competence tree. For each node, starting at the root, the user model looks at the progress score of the competence. Thanks to the invariant defined for the competence tree, you can state that when the progress score can be categorized as *good*, the competence and underlying competences can be considered to be under control. When the score is *reasonable*, you can state that the underlying competences are trained well enough, but the competence itself is not. When the score is *bad*, you can state that the underlying competences are not yet trained well enough, and these should be inspected. The tree is run through until a node is detected - with no untrained children, that can be trained.

B.6 Select the initial level of scaffolding

Input: next learning focus, its associated progress score
Output: the initial LoS for the next scenario
Algorithm: to determine the initial LoS, a linear function is used. When someone has a high progress score for the selected learning focus, the LoS is low. Otherwise, this level increases up to total support.

Appendix C

User model source code

In this appendix, the code of the user model is presented. The code responsible for online personalization is integrated into the *monitor agent*. The code responsible for the offline personalization is presented in the *user model agent*.

Throughout the code, comments are placed to illustrate the program cycle and the algorithms that were defined for each necessary user model (sub-) operation (as constructed in Section 7.3.1). The main code for each (sub-) operation can be found on the following page:

1. Adjust the level of scaffolding (page 75)
2. Determine the performance score (page 76)
3. Determine the trainee's level of motivation (page 79)
4. Update the competence tree structure (page 81)
5. Select the next learning focus (page 83)
6. Select the initial level of scaffolding (page 81)

In Section 7.3.1 – General Algorithms, several variable parameters where mentioned that need to be fine-tuned in the application. In this specific implementation, the following values were chosen:

1. A performance score and the progress score associated with a competence can be classified in the following way:
    1. If the score is 80 or higher, it is classified as *good*
    2. If the score is between 50 and 80, it is classified as *reasonable*
    3. If the score is below 50, it is classified as *bad*
2. The level of motivation:
    1. If the average motivation is 5 or higher, it is classified as *high*
    2. If the average motivation is below 5, it is classified as *low*
3. The minimum amount of consecutive training sessions without changing the selected learning focus is 3, the maximum amount of consecutive training sessions without changing the selected learning focus is 5

C.1 Monitor Agent

```
/*
User model implementation - on-line part, concerned with:
        Applying scaffolding, based on the trainee's behavior
        Calculating scenario score
Contains the "observing, processing and adapting" steps of the personalization cycle
*/
```

BeliefUpdates:

```
/* Belief update for saving the Actionplan and initial Scaffolding */
{ not actionplan(ACTIONPLAN) and not scaffolding(_) }
        ActionPlan(ACTIONPLAN, SCAFFOLDING)
{ actionplan(ACTIONPLAN), scaffolding(SCAFFOLDING) }

/* Starting the monitor */
{ not start(monitor) and not scenario(done) }
        StartMonitor()
{ start(monitor) }

/* Ending the monitor */
{ start(monitor) }
        StopMonitor()
{ not start(monitor) }

/* Reinitialising belief base */
{ scenario(done) and scaffolding(Scaffolding) and necessaryActions(Actions) and
        totalScore(Score) and scenarioScore(ScenarioScore) }
        ResetScenario()
{ not scenario(done), mistakes(0), not scaffolding(Scaffolding), not
        necessaryActions(Actions), not totalScore(Score), not
        scenarioScore(ScenarioScore)  }

{ log(A1, S1, M1, T1) }
        DeleteLog(log(A1, S1, M1, T1))
{ not log(A1, S1, M1, T1) }

/* Adding a time constraint */
{ not time_constraint(ACTION) }
        TimeConstraint(ACTION)
 { time_constraint(ACTION) }

/* Maintaining and building up the total scenario score */
{ totalScore(Old) }
        UpdateTotalScore(New)
{ not totalScore(Old), totalScore(Old + New) }

{ }
        UpdateTotalScore(New)
{ totalScore(New) }

{ }
        RegisterScore(ScenarioScore)
{ scenarioScore(ScenarioScore) }
```

```
{ }
        AddNecessaryAction(Actions)
{ necessaryActions(Actions) }

/* Timeouts  result in particular responses from the monitor in case the trainee is not performing
the right action within a certain time frame, the exact response of the monitor depends on the
current level of scaffolding. In case the level of scaffolding is 3 when the trainee time's out, the
action is performed by another (non-player) character. Whenever an action in the actionplan is
successfully executed by the trainee or auctioned off to another character, the monitor goes to the
next action. After an action is performed, a log entry is made, format: log(action(ACTION),
scaffolding(SCAFFOLDING), mistakes(N), (no)timeout) */

/* These 2 timeouts are for the last action in the actionplan, the first rule is applicable when the
level of scaffolding is at its highest (3); in this case the scenario should stop, because the last
action has been performed by the NPC. The second rule is applicable when the level of scaffolding
is anything but 3. In this case, the action has not yet been performed by the trainee, nor by one of
the NPCs. Therefore, the scenario is not done yet and the level of scaffolding is increased by one
because of the learner's failure to perform the action in time. */
{ actionplan([action(ACTION, OPTIONS, TIMEOUT)]) and time_constraint(ACTION) and
                mistakes(N) and scaffolding(3) }
        TimeOut(ACTION)
{ not time_constraint(ACTION), not actionplan([action(ACTION, OPTIONS, TIMEOUT)]),
                log(action(ACTION), scaffolding(3), mistakes(N), timeout), not mistakes(N),
        scenario(done) }

{ actionplan([action(ACTION, OPTIONS, TIMEOUT)]) and time_constraint(ACTION) and
                mistakes(N) and scaffolding(SCAFFOLDING) }
        TimeOut(ACTION)
{ not time_constraint(ACTION), log(action(ACTION), scaffolding(SCAFFOLDING), mistakes(N),
                timeout), not mistakes(N), mistakes(0), not scaffolding(SCAFFOLDING),
                scaffolding(SCAFFOLDING+1) }

/* These 2 timeouts are for any action but the last action in the actionplan. Again there is a
distinction between the two cases regarding the level of scaffolding. Only in this case, the first rule
replaces the current action plan with the action plan that is left after removing the current action.
The second rule is the same as the second rule above: it increases the level of scaffolding by 1.*/
{ actionplan([action(ACTION, OPTIONS, TIMEOUT)|REST]) and time_constraint(ACTION) and
                mistakes(N) and scaffolding(3) }
        TimeOut(ACTION)
{ not time_constraint(ACTION), not actionplan([action(ACTION, OPTIONS, TIMEOUT)|REST]),
                actionplan(REST), log(action(ACTION), scaffolding(3), mistakes(N), timeout),
                not mistakes(N), mistakes(0) }

{ actionplan([action(ACTION, OPTIONS, TIMEOUT)|REST]) and time_constraint(ACTION) and
                mistakes(N) and scaffolding(SCAFFOLDING) }
        TimeOut(ACTION)
{ not time_constraint(ACTION), log(action(ACTION), scaffolding(SCAFFOLDING), mistakes(N),
                timeout), not mistakes(N), mistakes(0), not scaffolding(SCAFFOLDING),
                scaffolding(SCAFFOLDING+1) }

/* TestAction tests whether the trainee's taken action is correct. If it is incorrect it will increase the
level of scaffolding. Whenever an action in the actionplan is successfully executed by the trainee or
auctioned off to another character, the monitor goes to the next action.*/

/* Testaction for correctly executing the last action in the actionplan. Because this is the last
action in the plan, the scenario comes to an end after processing this action.*/
{ actionplan([action(ACTION, OPTIONS, TIMEOUT)]) and mistakes(N) and
```

```
                time_constraint(ACTION)  and scaffolding(SCAFFOLDING) }
        TestAction(ACTION)
{ not time_constraint(ACTION), not actionplan([action(ACTION, OPTIONS, TIMEOUT)]),
                log(action(ACTION), scaffolding(SCAFFOLDING), mistakes(N), notimeout), not
                mistakes(N), scenario(done) }

/* These 2 actions are for correctly executing any action but the last in the actionplan. The first
rule is applicable when the level of scaffolding is 0. This case is handled separately because the
level of scaffolding is already at its minimum and is not decreased any further. The second rule is
for all other cases; in these cases the level of scaffolding is decreased by one because of the
correct performance of the action. */
{ actionplan([action(ACTION, OPTIONS, TIMEOUT)|REST]) and mistakes(N) and
                time_constraint(ACTION) and scaffolding(0) }
        TestAction(ACTION)
{ not time_constraint(ACTION), not actionplan([action(ACTION, OPTIONS, TIMEOUT)|REST]),
                actionplan(REST), log(action(ACTION), scaffolding(0), mistakes(N),
        notimeout), not mistakes(N), mistakes(0) }

{ actionplan([action(ACTION, OPTIONS, TIMEOUT)|REST]) and mistakes(N) and
                time_constraint(ACTION) and scaffolding(SCAFFOLDING) }
        TestAction(ACTION)
{ not time_constraint(ACTION), not actionplan([action(ACTION, OPTIONS, TIMEOUT)|REST]),
                actionplan(REST), log(action(ACTION), scaffolding(SCAFFOLDING),
        mistakes(N), notimeout), not mistakes(N), mistakes(0), not
        scaffolding(SCAFFOLDING), scaffolding(SCAFFOLDING-1) }

/* These 2 test actions are for incorrectly executing any action in the actionplan. The first action is
the case where the level of scaffolding is at its maximum value. In this case the level of scaffolding
is not increased. In the second rule all other cases are handled; in these cases the level of
scaffolding is increased by one because of the incorrect performance of the action. */
{ not actionplan([action(ACTION, OPTIONS, TIMEOUT)|_]) and mistakes(N) and scaffolding(3) }
        TestAction(ACTION)
{ not mistakes(N), mistakes(N+1) }

{ not actionplan([action(ACTION, OPTIONS, TIMEOUT)|_]) and mistakes(N) and
                scaffolding(SCAFFOLDING) }
        TestAction(ACTION)
{ not mistakes(N), mistakes(N+1), not scaffolding(SCAFFOLDING),scaffolding(SCAFFOLDING+1) }
```

Beliefs:

```
/* Initially 0 mistakes are made. */
mistakes(0).
```

Goals:

```
/* Monitor's goals */
keep( time_constraint )
```

PG-rules:

```
/* When the trainee is expected to perform an action, a time constraint is expected. As such, each
action has a Time variable (in seconds). This is then passed on to the environment. A timer thread
is then made in the environment which will throw the event in the specified amount of time. */

keep( time_constraint ) <- not scenario(done) and start(monitor) and
        actionplan([action(ACTION, _, TIME)|_]) and not time_constraint(ACTION) |
```

```
{[      TimeConstraint(ACTION);
        @env( timerEvent( ACTION, TIME ), _ )
]}
```

PC-rules:

```
/* When the director informs the monitor of the actionplan, save the actionplan and scaffolding in
the beliefbase. */
message( director, inform, _, _, actionplan(ACTIONPLAN, SCAFFOLDING) ) <- true |
{
        retrieveNecessaryActions(ACTIONPLAN, []);
        ActionPlan(ACTIONPLAN, SCAFFOLDING)
}

/* Subtract the necessary actions */
retrieveNecessaryActions([action(Action, Options, Time) | REST], SkillList) <- true |
{
        if B( REST = [] ) then {
                AddNecessaryAction([Action | SkillList])
        } else {
                retrieveNecessaryActions(REST, [Action | SkillList] )
        }
}

/* When the director informs the monitor to start, it starts.*/
message( director, inform, _, _, start(monitor) ) <- not start(monitor) and not
        scenario(done) |
{[
        StartMonitor()
]}

/* When the scenario plan is finished, ask the trainee some necessary questions and inform the
director about the scenario results */
message( director, inform, _, _, scenario(evaluate) ) <- scenario(done) |
{[
        // Ask the trainee for its current motivation
        B(VARLIST = [ ["1", "1"], ["3", "3"], ["5", "5"], ["7", "7"], ["9", "9"] ]);
        @env( speechAct( monitor, ': Please assign a number to you current motivation (1 =
        not motivated, 9 = very motivated)', VARLIST), X);
        B(X = [actionresult(Z)] and Y = [Z]);

        // Calculate scenario score
        calculateScore();
        B(scenarioScore(Score));

        // Inform the director of the results
        send( director, inform, performance(score( Score ), motivation( Z ) ))
]}

/* Resetting the different scenario dependent beliefs when the scenario is finished */
message( director, inform, _, _, reset(beliefs) ) <- scenario(done) |
{[
        ResetScenario()
]}

/* Adjust the level of scaffolding
        The level of scaffolding gets adjusted according to the correctness of the trainee's
```

```
        actions, and when a time-out occurs. BeliefUpdates are called upon to change the
        level of scaffolding in the Belief Base */

/* If the trainee performs an action (thrown by the environment), it is tested by the monitor. */
event( trainee_action(ACTION), env ) <- not scenario(done) and start(monitor) |
{[
        TestAction(ACTION);
        checkDevelopement()
]}

/* If the timer event is thrown, the current level of scaffolding is used to determine what action to
take, based on the actionplan. For level of scaffolding 0, no action is taken. For level of scaffolding
1, goal 1 is sent to the director for auctioning, for level of scaffolding 2, goal 2 is sent, etc. */
event( timer(ACTION), env ) <- not scenario(done) and actionplan([action(ACTION, [GOAL1,
        GOAL2, GOAL3], _)|_]) and time_constraint(ACTION) and scaffolding(SCAFFOLDING) |
{[
        if B(SCAFFOLDING = 0) then
        {
                skip
        }
        else if B(SCAFFOLDING = 1) then
        {
                send( director, inform, auction( GOAL1 ) )
        }
        else if B(SCAFFOLDING = 2) then
        {
                send( director, inform, auction( GOAL2 ) )
        }
        else if B(SCAFFOLDING = 3) then
        {
                send( director, inform, auction( GOAL3 ) )
        };
        TimeOut(ACTION);
        checkDevelopement()
]}

/* Check if the scenario has ended due to a time out or a correct action */
checkDevelopement() <- scenario(done) |
{
        StopMonitor();
        send( director, inform, end( scenario ) )
}

checkDevelopement() <- true |
{
        skip
}

/* Determine the performance score
        Recursive function to calculate the score at the end of the scenario, by examining the
scenario log. Log details:
                log(action(ACTION), scaffolding(SCAFFOLDING), mistakes(N), (no)timeout)
An action is logged when a time-out takes place or the action is performed by the trainee or a
character. In the belief base, the log is build up sequentially: logs are ordered according to time.
Mistake details: When a time-out takes place, the amount of mistakes do not directly influence the
maximum score; each mistake causes the LoS to drop, causing MaxScore to drop, resulting in the
```

```
preferred behaviour. When a time-out has occurred before the trainee correctly finishes the Action,
possible mistakes have a relatively bigger influence on the Score */

calculateScore() <- necessaryActions( ActionList ) |
{
        calculateActionScore(ActionList, 100)
}

/* Calculate scenario score: no more actions are left, lets wrap things up! */
calculateActionScore( [], MaxScore ) <- necessaryActions( ActionList ) |
{
        // Calculate the average score of the whole scenario
        B(
                totalScore(X) and
                length(ActionList, Length) and
                is( ScenarioScore, (X / Length))
        );
        RegisterScore(ScenarioScore)
}

/* Calculate scenario score: the action got a time-out */
calculateActionScore( [Action | REST], MaxScore ) <- log(action(Action),
        scaffolding(Scaffolding), mistakes(N), timeout) |
{
        if B(Scaffolding = 3) then {
                // instantly give a 0 performance score
                UpdateTotalScore(0);
                // delete log entry
                DeleteLog(log(action(Action),scaffolding(Scaffolding),mistakes(N),timeout));
                // continue with the next log entry
                calculateActionScore( REST, 100 )
        }
        else {
                // decrease the maximal possible score
                B(is(NewMaxScore, ( 100 - (25 * (Scaffolding + 1)) )));
                DeleteLog(log(action(Action),scaffolding(Scaffolding),mistakes(N),timeout));
                // continue with the next log entry, since Action not finished
                calculateActionScore( [Action | REST], NewMaxScore )
        }
}

/* Calculate scenario score: the action was done correctly by the trainee */
calculateActionScore( [Action | REST], MaxScore ) <- log(action(Action),
        scaffolding(Scaffolding), mistakes(N), notimeout) |
{
        // subtract the penalty (amount of mistakes) made from the maximal possible score (25
                per mistake)
        B( is(ActionScore, max(0, (MaxScore - (N * 25)))) );
        // add the action score to the total score
        UpdateTotalScore(ActionScore);
        // delete log entry
        DeleteLog(log(action(Action), scaffolding(Scaffolding), mistakes(N), notimeout));
        // continue with the next action
        calculateActionScore( REST, 100 )
}
```

C.2 User Model Agent

```
/*
User model implementation - off-line part. especially concerned with initialization and evaluation of
the scenario

        Functionalities for updating the competences, considering competence progress
        Functionalities for selecting a new learning focus
        Functionality for determining the initial level of scaffolding
        Functionalities for keeping track of the trainee's motivation

Contains the "processing" and "reasoning" steps of the personalization cycle
*/
```

BeliefUpdates:

```
/* Register and maintain the learning focus and level of scaffolding */
{ learningFocus(Old, TimesTrained) and learningFocusAverage(Old, Score) }
        UpdateLearningFocus(New)
{ not learningFocus(Old, TimesTrained), learningFocus(New, 0), not learningFocusAverage(Old,
Score) }

{ }
        UpdateLearningFocus(New)
{ learningFocus(New, 0) }

{ currentLoS(Old) }
        UpdateLoS(New)
{ not currentLoS(Old), currentLoS(New) }

{ }
        UpdateLoS(New)
{ currentLoS(New) }

{ competence( Name, Score1, Parent, SkillList ) }
        UpdateCompetence(Name, Score1, Score2)
{ not competence( Name, Score1, Parent, SkillList ), competence( Name, Score2, Parent, SkillList
) }

/* Update counter to keep an eye on the amount of times a competence was trained  */{
learningFocus(Name, Number) }
        IncreaseCounter(Name)
{ not learningFocus(Name, Number), learningFocus(Name, Number + 1) }

/* Keep track of the average score of the current learning focus         */
{ learningFocusAverage(Name, OldScore) }
                        UpdateLearningFocusAverage(Name, Score)
{ not learningFocusAverage(Name, OldScore),
                         learningFocusAverage(Name, (Score + OldScore) / 2) }

{ }
        UpdateLearningFocusAverage(NewName, Score)
{ learningFocusAverage(NewName, Score) }

/* Keeping track of the trainee's motivation*/
{ motivation(Old) }
        UpdateMotivation(New)
{ not motivation(Old), motivation((New + Old) / 2) }
```

```
{ }
        UpdateMotivation(New)
{ motivation(New) }

{ motivationLevel(Old) }
        ChangeMotivation(Motivation)
{ not motivationLevel(Old), motivationLevel(Motivation) }

{ }
        ChangeMotivation(Motivation)
{ motivationLevel(Motivation) }
```

Beliefs:

```
/* Competence tree data structure: competence(Name, Score, Parent, [ Children* ]),
        where Children is of the same format as competence(…)
The initial values in this tree are chosen to work with the specified use cases (see Appendix D). In
case the trainee is new, all values are set to 0 and are dynamically updated based on the
performance scores of the trainee.*/
competence(sociale_vaardigheden, 8.75, root, [verantwoordelijkheid_nemen,
                communiceren, plannen_en_organiseren, samenwerken, assertiviteit]).
        competence( verantwoordelijkheid_nemen, 0, sociale_vaardigheden, [
                        gerust_stellen, betrokkenheid, delegeren] ).
                competence( gerust_stellen, 0, verantwoordelijkheid_nemen, [] ).
                competence( betrokkenheid, 0, verantwoordelijkheid_nemen, [] ).
                competence( delegeren, 0, verantwoordelijkheid_nemen, [] ).
        competence( communiceren, 70, sociale_vaardigheden, [ uitvragen, rapporteren ] ).
                competence( uitvragen, 90, communiceren, [] ).
                competence( rapporteren, 80, communiceren, [ abc_rapportage ] ).
                        competence( abc_rapportage, 80, rapporteren, [] ).
        competence( plannen_en_organiseren, 0, sociale_vaardigheden, [] ).
        competence( samenwerken, 0, sociale_vaardigheden, [] ).
        competence( assertiviteit, 0, sociale_vaardigheden, [] ).
```

Goals:

```
        observe(motivation)
```

PG-rules:

```
/* Determine the trainee's level of motivation
        Keep track of the trainee's motivation, if too low, take appropriate actions */
observe(motivation) <- motivation(M) and M < 5 and not motivationLevel(low) |
{
        ChangeMotivation(low)
}

observe(motivation) <- motivation(M) and M >= 5 and not motivationLevel(high) |
{
        ChangeMotivation(high)
}
```

PC-rules:

```
/* Initial message from the director to initiate scenario          */
message( director, request, _, _, request(userModelData) ) <- true |
{
```

```
        determineLearningFocus();
        B( learningFocus(Focus, Number) );

        B( competence( Focus, CurrentScore, Parent, SkillList ) );
        determineLoS(competence(Focus, CurrentScore));
        B( currentLoS(LoS) );

        // Send the learning focus and level of scaffolding back to the director
        send( director, inform, userData( learningFocus(Focus), scaffolding(LoS) ) )
}

/* Scenario results from scenario are sent from the director, determine parameters to pick new
scenario */
message( director, inform, _, _, update(competence(Name, Score), motivation(Motivation) ) ) <-
true |
{
        // Update the current motivation
        UpdateMotivation(Motivation);

        // Increase Times-Learned-Competence counter
        IncreaseCounter(Name);

        // Update the average score of the current learning focus
        UpdateLearningFocusAverage(Name, Score);

        initiateNewScenario(Name)
}

/* Determine learning focus and level of scaffolding
        If the previous learning focus is not yet trained enough (time trained < 5 and
        average score < 80) and the trainee is still motivated, keep the same focus */
initiateNewScenario(Focus) <- learningFocus(Focus, Counter) and
        learningFocusAverage(Focus, Score) and ((Counter < 5 and Score < 80 and
        motivationLevel(high)) or (Counter < 3 ) ) |
{
        // Keep the same focus, determine a new LoS
        determineLoS(competence(Focus, Score));
        B( currentLoS(LoS) );

        send( director, inform, userData( learningFocus(Focus), scaffolding(LoS) ) )
}

/* Determine learning focus and level of scaffolding
        If the previous learning focus is trained well enough, or for at most 5 times,
        determine a new learning focus and accompanying level of scaffolding */
initiateNewScenario(OldFocus) <- learningFocus(OldFocus, Counter) and
        learningFocusAverage(OldFocus, Score) and ((Counter >= 3 and
        (motivationLevel(low) or Score >= 80)) or (Counter > 4)) |
{
        // Update the competence tree with the new score
        updateTree(competence(OldFocus, Score));

        determineLearningFocus();
        B( learningFocus(Focus, Number) );
```

```
        B( competence( Focus, CurrentScore, Parent, SkillList ) );
        determineLoS(competence(Focus, CurrentScore));
        B( currentLoS(LoS) );

        // Send the learning focus and level of scaffolding back to the director
        send( director, inform, userData( learningFocus(Focus), scaffolding(LoS) ) )
}

/* Select the initial level of scaffolding
        Procedural approach to determine the level of scaffolding when a learning focus
        has been chosen */
determineLoS(competence(Name, CurrentScore)) <- true |
{
        if B(CurrentScore < 25 ) then {
                UpdateLoS(3)
        }
        else if B(CurrentScore < 50 ) then {
                UpdateLoS(2)
        }
        else if B(CurrentScore < 75 ) then {
                UpdateLoS(1)
        }
        else if B(CurrentScore =< 100 ) then {
                UpdateLoS(0)
        }
}

/* Updating the competence tree structure
        Update the recently trained learning focus, and every competence above, and
        when necessary below. Formalized, the following update takes place:
                Parent = OldParent – ((min(100, OldChild/4 * 5)) / 2) / #Children +
                                ((min (100, NewChild /4 * 5)) / 2) / #Children
        When a score is below 50, the underlying children get a penalty:
                ChildNew = ChildOld – (50 – Parent)
        Invariant: When each child of a competence node ranges between 80-100, the
        competence node itself will be 50%     */

/* The competence that should be updated is a LEAF   */
updateTree( competence(Name, Progress) ) <- competence(Name, OldScore, Parent, []) |
{
        // Overwrite previous score
        UpdateCompetence(Name, OldScore, Progress);

        // Update competence score of parent
        updateParent(Parent, OldScore, Progress)
}

/* The competence that is updated is a NODE
        Score >= 50: Competence was trained well enough; general tree score increases
        Score < 50: Competence score is embarrassing, general tree score decreases */
updateTree( competence(Name, Progress) ) <- competence(Name, Current, Parent,  SubSkills) |
{
        if B( Progress >= 50 ) then {
                // Overwrite previous score
                UpdateCompetence(Name, Current, Progress);
                // Recalculate the parent scores
```

```
                updateParent(Parent, Current, Progress)
        }
        else if B( Progress < 50 ) then {
                // Reset own competence score to 50
                UpdateCompetence(Name, Current, 50);
                // Update parents according to the new score
                updateParent(Parent, Current, 50);
                // Determine penalty due to the low score
                B( is( Penalty, 50 - Progress ) );
                // Recalculate the scores of the children. The new competence score is
                determined automatically
                updateChildren(SubSkills, Penalty)
        }
}

/* Reduce the score of each child with the penalty that was given */
updateChildren( [X|REST], Penalty ) <- competence(X, Current, Parent, SubSkills) |
{
        // Reduce the current score, with the penalty
        B( is( ReducedScore, Current - Penalty ) );
        // Recalculate the scores in the tree according to the new value
        updateTree( competence(X, ReducedScore) );
        // Recursively continue updating the rest of the list of children, as long as the list
        is not empty
        if B( not (REST = []) ) then {
                updateChildren(REST, Penalty)
        }
}

/* Update the parent of the current node or leaf */
updateParent( ParentName, ChildOldScore, ChildProgress ) <- competence(ParentName,
        ParentOldScore, GrandParent, SubSkills) |
{
        // Calculate the new competence score of the parent
        B(
                // Adjust the incoming child score, necessary for the mapping of scores
                ranging between 80-100 to score 100
                is(AdjustedChildOldScore, (min(100, (5 * (ChildOldScore / 4))))) and
                is(AdjustedChildProgress, (min(100, (5 * (ChildProgress / 4))))) and
                // Calculating the new parent score by replacing the old childscore with the
                new childscore
                length(SubSkills, Length) and
                is(Minus, ( 0.5 * (AdjustedChildOldScore / Length) )) and
                is(Plus, ( 0.5 * (AdjustedChildProgress / Length) )) and
                is(UpdatedParentScore, (ParentOldScore + Plus) - Minus)
        );

        // Update existing competence with the new score
        UpdateCompetence(ParentName, ParentOldScore, UpdatedParentScore);
        // Update the parent competence
        updateParent( GrandParent, ParentOldScore, UpdatedParentScore)
}

/* The parent node is the tree root */
updateParent( root, ChildOldScore, ChildProgress ) <- true |
{
```

```
		skip
}

/* Select the next learning focus
		Invariant: When each child of a competence node ranges between 80-100, the
		competence node itself will be 50%. This means that when a main skill is < 50,
		there exists a sub-skill that must be trained. This infers that it is not possible to
		look at an empty list of sub skills and thus reduces possible cases */

/* Start with the root skill of the tree
		Score >= 50: pick root skill as learning focus
		Score < 50: inspect the sub skills of the root skill, randomly choose one */
determineLearningFocus() <- competence(Skill, Score, root, SkillList) |
{
		if B(Score >= 50) then
		{
			UpdateLearningFocus(Skill)
		}
		else if B( Score < 50 ) then
		{
			determineRandomLearningFocus(SkillList)
		}
}

/* Look at the underlying competences of the main competence, each a LEAF
		Score < 80: skill must be trained again until score > 80, so skill will be learning focusScore
>= 80: skill is trained well, inspect the skill's siblings, randomly choose one */
determineLearningFocus( [ Skill | Rest ] ) <- competence( Skill, Score, Parent, [] ) |
{
		if B( Score < 80 ) then
		{
			UpdateLearningFocus(Skill)
		}
		else if B( Score >= 80 ) then
		{
			determineRandomLearningFocus(Rest)
		}
}

/* Look at the underlying competences of the main competence, each a NODE
		Score < 50: inspect the sub skills, randomly choose one
		50 >= Score < 80: sub skills are trained well, main skill will be learning focus
		Score >= 80: main skill and subs skills are trained well, inspect the skill's siblings,
		randomly choose one */
determineLearningFocus( [ Skill | Rest ] ) <- competence(Skill, Score, Parent, SkillList) |
{
		if B( Score < 50 ) then
		{
			determineRandomLearningFocus(SkillList)
		}
		else if B( Score < 80 and Score >= 50 ) then
		{
			UpdateLearningFocus(Skill)
		}
		else if B( Score >= 80 ) then
		{
```

```
                determineLearningFocus(Rest)
        }
}

/* Select a random competence to test next */
determineRandomLearningFocus( SkillList ) <- true |
{
        B(
                length(SkillList, Length) and
                is( Index, int(random(Length)) ) and
                nth0(Index, SkillList, Element) and
                delete(SkillList, Element, Abridged_SkillList)
        );
        determineLearningFocus( [ Element | Abridged_SkillList ] )
}

/* Something went wrong in determining the next learning focus - should be impossible */
determineLearningFocus( [] ) <- true |
{
        UpdateLearningFocus(something_went_wrong)
}
```

Appendix D

Adaptive educational game Use Cases

In this appendix, the use cases that were constructed to verify the different sub-operations of the user model are worked out. The sub-operations of the user model that were verified are:

Online:

1. Adjust the level of scaffolding

Offline:

2. Determine the performance score
3. Determine the trainee's level of motivation
4. Update the competence tree structure
5. Select the next learning focus
6. Select the initial level of scaffolding

Each use case gives a detailed description of the possible interactions that can take place between the trainee and the system. When the operation does not get direct input from the user, instead of a description of the possible use cases, the possible sets of parameters are given and tested. These parameters are made concrete and often belong to a certain class of input (parenthesized). Relatively easy subtasks or operations, such as taking the average of a new and an old score, are assumed to be correct and therefore excluded. Each use case or parameter set has the following format:

1. *Input provided to the function*
2. *Expected functional behavior*
3. *Observed functional behavior*

The expected functional behavior is determined according to the algorithms defined in Appendix B, that were later worked out in the implementation (see Appendix C). When the expected and the observed behavior coincide, it means that the user model performs the desired behavior. To reduce the different possible use cases, in some cases multiple slightly different inputs are given. The corresponding expected and observed behavior are systematically written down in the same use case.

To ensure that the different use cases and parameter sets can be reenacted, the general input and the way how the result was observed are specified for each sub-operation. The general input specifies how the input for the use case is constructed, such as the actions done in the systems or the initial content of the belief base. The initial belief base can consist of multiple specific predicates, which can contain variables that change for each use case. These variables are given in italics and start with a capital letter.

D.1 Adjust the Level of Scaffolding

The level of scaffolding (LoS) determines what kind of intervention will take place after a time-out occurs. During a scenario, the LoS is adjusted when the trainee performs an action (either correct or incorrect), or when a time-out takes place.

General Input

The following scenario plan is used:

Learning focus: calm victim

Possible actions in the scenario: 'calm victim', 'stabilize limb', 'do nothing'

Action plan:

calm: Interventions:

LoS 0: do not perform an action

LoS 1: hint: Wow, he/she needs to calm down!

LoS 2: hint: Maybe you should try to calm him/her down first

LoS 3: take over action: calm the victim

The desired LoS is gained by waiting for a time-out during the scenario run. It can range between 0 and 3.

Observation of the result

After the input is processed, the value of the predicate scaffolding(*Level*) in the belief base is inspected.

Use case 1: the trainee performs the right action

Input: the trainee performs the desired action ‘calm’ in the environment.

1. The current LoS is 0
2. The current LoS is 1
3. The current LoS is 2
4. The current LoS is 3

Expected functional behavior: the LoS is decreased by 1, with a minimum value of 0

1. The LoS stays 0
2. The LoS is decreased to 0
3. The LoS is decreased to 1
4. The LoS is decreased to 2

Observed functional behavior:

1. The belief base is not changed
2. The belief base is updated by removing scaffolding(1) and adding scaffolding(0)
3. The belief base is updated by removing scaffolding(2) and adding scaffolding(1)
4. The belief base is updated by removing scaffolding(3) and adding scaffolding(2)

Use case 2: the trainee performs the wrong action

Input: the trainee performs the wrong action (either `do nothing’ or `stabilize limb’) in the environment.

1. The current LoS is 0
2. The current LoS is 1
3. The current LoS is 2
4. The current LoS is 3

Expected functional behavior: the LoS goes up by 1, with a maximum value of 3

1. The current LoS is increased to 1
2. The current LoS is increased to 2
3. The current LoS is increased to 3
4. The current LoS stays 3

Observed functional behavior:

1. The belief base is updated by removing scaffolding(0) and adding scaffolding(1)
2. The belief base is updated by removing scaffolding(1) and adding scaffolding(2)
3. The belief base is updated by removing scaffolding(2) and adding scaffolding(3)
4. The belief base is not changed

Use case 3: a time-out occurs

Input: the trainee fails to perform the action "calm" in the environment before a Time-Out takes place.

1. The current LoS is 0
2. The current LoS is 1
3. The current LoS is 2
4. The current LoS is 3

Expected functional behavior: the LoS is increased by 1 with a maximum value of 3, an intervention is executed. the content of the intervention depends on the previous value of the LoS.

1. The current LoS is increased to 1, nothing happens
2. The current LoS is increased to 2, one of the characters gives the hint "Wow, he/she needs to calm down!"
3. The current LoS is increased to 3, one of the characters gives a hint "Maybe you should try to calm him/her down first"
4. The current LoS stays at 3, one of the characters takes over the action

Observed functional behavior:

1. The belief base is updated by removing scaffolding(0) and adding scaffolding(1), no message is sent
2. The belief base is updated by removing scaffolding(1) and adding scaffolding(2) , a message is sent to the director to start an auction among the characters to perform the hint "Wow, he/she needs to calm down!"
3. The belief base is updated by removing scaffolding(2) and adding scaffolding(3) , a message is sent to the director to start an auction among the characters to perform the hint "Maybe you should try to calm him/her down first"
4. The belief base is not changed, a message is sent to the director to start an auction among the characters to perform the action "calm the victim"

D.2 Determine the performance score

The performance score of the previous scenario expresses the progress of the trainee. The score of each individual action is influenced by the correctness of the action, the amount of mistakes made and whether or not a time-out took place. The first three use cases examine the calculation of the performance score when the action plan contains just one action. The fourth use case examines the performance score when the action plan consists of more than one action.

General input

For use cases 1, 2 and 3, the same action plan as given in Section D.1 is used. For use case 4, the action plan is extended with an additional action:

Stabilize leg: Interventions:

LoS 0: do not perform an action

LoS 1: hint: That leg still looks really bad!
LoS 2: hint: 'Perhaps you should try to stabilize the leg‘
LoS 3: stabilize the victim’s leg

The prerequisite inputs for each use case are gained by performing certain actions in the system. The defined LoS is gained by waiting for a time-out during the scenario run. The defined number of mistakes is obtained by executing the action ‘do nothing’ zero or more times. The performance score itself can range between 0 and 100.

Observation of the result
After the scenario is finished and the performance score is calculated by the Monitor agent, the predicate scenarioScore(*Score*) in the belief base contains the final performance score. Because use cases 1, 2 and 3 only contain one action, the total score is the same as the score for this individual action.

Use case 1: the trainee performs the action in the right way

Input: the trainee performs the right action (calm) in the environment, without any time-outs

1. Number of mistakes: 0
2. Number of mistakes: 1
3. Number of mistakes: 2
4. Number of mistakes: 3
5. Number of mistakes: 4 or more

Expected functional behavior: the resulting score is 100, decreased according to the number of mistakes that were made

1. The resulting score is 100
2. The resulting score is 75
3. The resulting score is 50
4. The resulting score is 25
5. The resulting score is 0

Observed functional behavior: the total amount of points for the scenario should be increased with the points gained for performing the action

1. the belief base is altered by adding the belief scenarioScore (100)
2. the belief base is altered by adding the belief scenarioScore (75)
3. the belief base is altered by adding the belief scenarioScore (50)
4. the belief base is altered by adding the belief scenarioScore (25)
5. the belief base is altered by adding the belief scenarioScore (0)

Use case 2: the trainee performs the action after a time-out

Input: the trainee performs the action (calm) after a time-out,

1. LoS during the time-out is 0
    1.1. amount of mistakes is 0
    1.2. amount of mistakes is 1
    1.3. amount of mistakes is 2
    1.4. amount of mistakes is 3 or more
2. LoS during the time-out is 1
    2.1. amount of mistakes is 0
    2.2. amount of mistakes is 1
    2.3. amount of mistakes is 2 or more
3. LoS during the time-out is 2

3.1. amount of mistakes is 0
3.2. amount of mistakes is 1 or more

Expected functional behavior: when the LoS during the time-out and / or when the amount of mistakes after finally performing the right action increases, the score for the individual action decreases

1. LoS during the time-out is 0
   1.1. the resulting score is 75
   1.2. the resulting score is 50
   1.3. the resulting score is 25
   1.4. the resulting score is 0
2. LoS during the time-out is 1
   2.1. the resulting score is 50
   2.2. the resulting score is 25
   2.3. the resulting score is 0
3. LoS during the time-out is 2
   3.1. the resulting score is 25
   3.2. the resulting score is 0

Observed functional behavior:

1. LoS during the time-out is 0
   1.1. the belief base is altered, by adding the belief scenarioScore (75)
   1.2. the belief base is altered, by adding the belief scenarioScore (50)
   1.3. the belief base is altered, by adding the belief scenarioScore (25)
   1.4. the belief base is altered, by adding the belief scenarioScore (0)
2. LoS during the time-out is 1
   2.1. the belief base is altered, by adding the belief scenarioScore (50)
   2.2. the belief base is altered, by adding the belief scenarioScore (25)
   2.3. the belief base is altered, by adding the belief scenarioScore (0)
3. LoS during the time-out is 2
   3.1. the belief base is altered, by adding the belief scenarioScore (25)
   3.2. the belief base is altered, by adding the belief scenarioScore (0)

Use case 3: a non-player character performs the action

Input: the trainee waits too long before performing an action and the current LoS is 3

Expected functional behavior: the MADS takes over the action and ensures that one of the non-player characters performs the action. The resulting score for this action is 0

Observed functional behavior: the belief base is altered, by adding the belief scenarioScore (0)

Use case 4: the action plan contains multiple actions

Input: For this use case we consider several action plan executions:

1. the initial LoS is 1, a time-out takes place, the right action is performed (calm), a mistake is made (do nothing), and the right action is performed (stabilize)
2. the initial LoS is 2. First, a wrong action is performed (do nothing), then the right actions are performed (calm, stabilize)
3. the initial LoS is 3, a time-out takes place for the first action (calm), the right action is performed (stabilize)

Expected Functional Behavior:

1. the resulting score of the whole scenario is 62.5 ((50 + 75) / 2)

2. the resulting score of the whole scenario is 87.5 ((75 + 100) / 2)
3. the resulting score of the whole scenario is 50 ((0 + 100) / 2)

Observed Functional Behavior:

1. the belief base is altered, by adding the belief scenarioScore (62.5)
2. the belief base is altered, by adding the belief scenarioScore (87.5)
3. the belief base is altered, by adding the belief scenarioScore (50)

D.3 Determine the trainee's level of motivation

The level of motivation influences the selection of the next learning focus. At the end of the scenario, the current motivation of the trainee is requested. This motivation is processed by taking the average of the current motivation and the previously processed motivation score.

General input

To get the desired previous value and level of motivation, additional predicates were added to the belief base of the user model agent:

motivation(*Value*)
motivationLevel(*Value*)

The predicate motivation(*Value*) depicts the previously processed motivation. Both the observed as the processed motivation score can range between 1 and 9. The resulting level of motivation can be low (selected when motivation < 5) or high (selected when motivation >= 5).

Observation of the result

The predicate motivationLevel(*low / high)* depicts whether the level of motivation is high or low. This predicate should change according to the observations made.

Use case 1: the trainee indicates that his motivation is low

Input: the observed motivation is 3

1. the previously processed motivation is 3, the level of motivation is low
2. the previously processed motivation is 5, the level of motivation is high

Expected functional behavior:

1. the processed motivation stays 3, the level of motivation stays low
2. the processed motivation becomes 4, the level of motivation becomes low

Observed functional behavior:

1. the belief base is not changed
2. the predicate motivation(5) is deleted and the predicate motivation(4) is added, the predicate motivationLevel(high) is deleted and motivationLevel(low) is added

Use case 2: the trainee indicates that his motivation is high

Input: the processed motivation score is 7 (equal or higher than 5)

1. the previously processed motivation is 3, the level of motivation is low
2. the previously processed motivation is 5, the level of motivation is high

Expected functional behavior:

1. the processed motivation becomes 5, the level of motivation becomes high

2. the processed motivation becomes 6, the level of motivation stays high

Observed functional behavior: the belief base is changed

1. the predicate motivation(3) is deleted and the predicate motivation(5) is added, the predicate motivationLevel(low) is deleted and motivationLevel(high) is added
2. the predicate motivation(5) is deleted and the predicate motivation(6) is added, the predicate motivationLevel(high) is not changed

D.4 Update the competence tree structure

Before determining the next learning focus, the current competence tree is updated with the performance score of the previous scenario. In the different parameter sets of this section, a distinction is made between the performance score being bad (< 50), being reasonable (>= 50 and < 80) or being good (>= 80).

General input

The competence that was trained can be a *leaf competence* (a competence that does not have underlying competences) or a *node competence* (a competence that does have one or more underlying competences). The competence tree that we will use for these use cases has the following structure (competences that are in the same cluster and have the same level of indentation, are siblings of each other):

sociale_vaardigheden, score = 17.5
verantwoordelijkheid_nemen, score = 0
    gerust_stellen, score = 0
    betrokkenheid, score = 0
    delegeren, score = 0
communiceren, score = 70
    uitvragen, score = 90
    rapporteren, score = 80
        abc_rapportage, score = 80
plannen_en_organiseren, score = 70
samenwerken, score = 0
assertiviteit, score = 0

The scores in this tree have been calculated by hand according to the algorithm worked out in Appendix C. To recap, the update formula for the score of a parent is:

Parent = OldParent – ((min(100, OldChild/4 * 5)) / 2) / #Children + ((min (100, NewChild /4 * 5)) / 2) / #Children

When a score is below 50, the underlying children get a penalty: ChildNew = ChildOld – (50 – Parent)

Additionally, the method updateTree(competence(*competence, score*)) is added to the initial plan base of the user model. This means that when the MADS is started, the tree will be updated at once.

Observation of the result

By updating the trained competence in the tree, the underlying and overlying competences might be changed as well. The updated competences in the tree are returned as output.

Set 1: the performance score is bad

Input: the performance score is 40, and

1. the *leaf* competence ‘delegeren’ was trained
2. the *node* competence ‘communiceren’ was trained

Expected functional behavior:

1. the competence itself and the overlying competence are updated:
    a. delegeren, score = 40
    b. verantwoordelijkheid_nemen, score = 8.3333
        (= 0 – ((min(100, 0/4 * 5)) / 2) / 3 + ((min (100, 40/4 * 5)) / 2) / 3
        = 0 – 0 + 8.3333)
    c. sociale_vaardigheden, score = 18.5417
        (= 17.5–((min(100, 0/4 * 5)) / 2) / 5 + ((min (100, 8.3333 /4 * 5)) / 2) / 5
        = 17.5–0 + 1.0417)
2. the general tree score decreases, and
    a. the underlying competences are updated:
        i. uitvragen, score = 80
        (= 90 – (50-40))
        ii. rapporteren, score = 70
        (= 80 – (50-40))
    b. the overlying competences are updated
        i. communiceren, score = 46.875
        (= 50 – ((min(100, 90/4 * 5)) / 2) / 2 + ((min (100, 80/4 * 5)) / 2) / 2
        - ((min(100, 80/4 * 5)) / 2) / 2 + ((min (100, 70/4 * 5)) / 2) / 2
        = 50 – 25 + 25 – 25 + 21.875)
        ii. sociale_vaardigheden, score = 14,6094
        (= 17.5-((min(100, 70/4*5))/2)/5 + ((min (100, 46.875/4 * 5)) / 2) / 5
        = 17.5 – 8.75 + 5.8593)

Observed functional behavior:

1. the following nodes and leafs of the competence tree in the belief base are updated:
    a. delegeren, score = 40
    b. verantwoordelijkheid_nemen, score = 8.3332
    c. sociale_vaardigheden, score = 18.5417
2. the following nodes and leafs of the competence tree in the belief base are updated:
    a. uitvragen, score = 80
    b. rapporteren, score = 70
    c. communiceren, score = 46.875
    d. sociale_vaardigheden, score = 14.6094

There only seems to be a slight rounding difference between the manually calculated score and the scores given by the user model.

Set 2: the performance score is reasonable

Input: the performance score is 60, and

1. the *leaf* competence ‘delegeren’ was trained
2. the *node* competence ‘communiceren’ was trained

Expected functional behavior:

1. the competence itself and the overlying competence are updated:
    a. delegeren, score = 60
    b. verantwoordelijkheid_nemen, score = 12.5
        (= 0 – ((min(100, 0/4 * 5)) / 2) / 3 + ((min (100, 60/4 * 5)) / 2) / 3
        = 0 – 0 + 12.5)

c. sociale_vaardigheden, score = 19.0625
(= 17.5 – ((min(100, 0/4 * 5)) / 2) / 5 + ((min (100, 12,5/4 * 5)) / 2) / 5
= 17.5 – 0 + 1,5625)

2. the general tree score decreases, the overlying competences are updated
   a. communiceren, score = 60
   b. sociale_vaardigheden, score = 16.25
   (= 17.5 - ((min(100, 70/4 * 5)) / 2) / 5 + ((min (100, 60/4 * 5)) / 2) / 5
   = 17.5 – 8.75 + 7.5)

Observed functional behavior:

1. the competence itself and the overlying competence are updated:
   a. delegeren, score = 60
   b. verantwoordelijkheid_nemen, score = 12.5
   c. sociale_vaardigheden, score = 19.0625
2. the following nodes and leafs of the competence tree in the belief base are updated:
   a. communiceren, score = 60
   b. sociale_vaardigheden, score = 16.25

Set 3: the performance score is good

Input: the performance score is 90, and

1. the *leaf* competence 'delegeren' was trained
2. the *node* competence 'communiceren' was trained

Expected functional behavior:

1. the competence itself and the overlying competence are updated:
   a. delegeren, score = 90
   b. verantwoordelijkheid_nemen, score = 16.6667
   (= 0 – ((min(100, 0/4 * 5)) / 2) / 3 + ((min (100, 90/4 * 5)) / 2) / 3
   = 0 – 0 + 16.6667)
   c. sociale_vaardigheden, score = 19.5833
   (= 17.5–((min(100, 0/4 * 5)) / 2)/5 + ((min (100, 16.6667/4 * 5)) / 2) / 5
   = 17.5–0 + 2.0833)
2. the general tree score increases, the overlying competences are updated
   a. communiceren, score = 90
   b. sociale_vaardigheden, score = 18.75
   (= 17.5 - ((min(100, 70/4 * 5)) / 2) / 5 + ((min (100, 90/4 * 5)) / 2) / 5
   = 17.5 – 8.75 + 10)

Observed functional behavior:

1. the competence itself and the overlying competence are updated:
   a. delegeren, score = 90
   b. verantwoordelijkheid_nemen, score = 16.6668
   c. sociale_vaardigheden, score = 19.5832
2. the following nodes and leafs of the competence tree in the belief base are updated:
   a. communiceren, score = 90
   b. sociale_vaardigheden, score = 18.75

Again, there only seems to be a slight rounding difference between the manually calculated score and the scores given by the user model.

D.5 Select an appropriate learning focus

A learning focus is selected to initiate the next scenario. This learning focus can be the same as the current focus or it can be a new learning focus. To determine if the same or a new learning focus should be selected, we take into account the following parameters: 1) the number of times the current learning focus has been consecutively trained, 2) the updated level of motivation, and 3) the average of the performance scores regarding the current learning focus.

General input

The number of times the current learning focus has been consecutively trained ranges between 1 and 5. The competence 'communiceren' is used as current learning focus. To test the different combinations of parameters, for each case the belief base of the user model agent is adjusted by adding the following predicates:

- learningFocus(communiceren, *Times_trained*)
- learningFocusAverage(communiceren, *Score*)
- motivationLevel(*Level*)

Additionally, the method initiateNewScenario is added to the initial plan base of the user model. This means that when the MADS is started, the next learning focus is determined at once. The same competence tree as in Section D.4 will be used:

sociale_vaardigheden, score = 17.5
verantwoordelijkheid_nemen, score = 0
    gerust_stellen, score = 0
    betrokkenheid, score = 0
    delegeren, score = 0
communiceren, score = 70
    uitvragen, score = 90
    rapporteren, score = 80
        abc_rapportage, score = 80
plannen_en_organiseren, score = 70
samenwerken, score = 0
assertiviteit, score = 0

Observation of the result

The next learning focus and the times the learning focus was trained is defined in the belief base of the trainee agent by the predicate learningFocus(*Competence_name*, *Times_trained*). After determining the next learning focus, either the value of *Competence_name* or *Times_trained* is changed. When a *new* learning focus from the competence tree is chosen, it should either be a LEAF with a score lower than 80 or a NODE with a score between 50 and 80.

Set 1: learning focus has been trained for less than 3 times

Input: the learning focus 'communiceren' has been trained consecutively 2 times

Expected functional behavior: the learning focus stays the same

Observed functional behavior: the belief base is not changed

Set 2: learning focus has been trained between 3 and 5 times

Input: the learning focus 'communiceren' has been trained consecutively 4 times, and

1. The current motivation is high

1.1. The average score is 70 (below 80)
1.2. The average score is 90 (equal to or higher than 80)
2. The current motivation is low
2.1. The average score is 70 (below 80)
2.2. The average score is 90 (equal to or higher than 80)

Expected functional behavior:

1. The current motivation is high
1.1. the learning focus remains the same
1.2. a new learning focus is selected, conforming with the additional requirements
2. The current motivation is low
2.1. a new learning focus is selected, conforming with the additional requirements
2.2. a new learning focus is selected, conforming with the additional requirements

Observed functional behavior:

1. The current motivation is high
1.1. the belief base is not changed
1.2. the previous predicate learningFocus(communiceren, 4) is deleted and the belief learningFocus(betrokkenheid, 0) is added to the belief base. Since the score of betrokkenheid is 0, it conforms with the additional requirements.
2. The current motivation is low
2.1. the previous predicate learningFocus(communiceren, 4) is deleted and the belief learningFocus(samenwerken, 0) is added to the belief base. Since the score of samenwerken is 0, it conforms with the additional requirements.
2.2. the previous predicate learningFocus(communiceren, 4) is deleted and the belief learningFocus(plannen_en_organiseren, 0) is added to the belief base. Since the score of plannen_en_organiseren is 0, it conforms with the additional requirements.

Set 3: The learning focus has been trained 5 or more times

Input: the learning focus 'communiceren' has been trained consecutively 5 times

Expected functional behavior: a new learning focus is selected, conforming with the additional requirements

Observed functional behavior: the previous predicate learningFocus(communiceren, 5) is deleted and the belief learningFocus(samenwerken, 0) is added to the belief base. Since the score of samenwerken is 0, it conforms with the additional requirements.

D.6 Select an appropriate initial level of scaffolding

An initial level of scaffolding (LoS) is selected to initiate the next scenario. To determine this initial LoS, the associated score of the learning focus as selected in Section D.5 is used. The overall behavior that needs to be observed, is that if the score is low, the LoS should be high and when the score is high, the LoS should be low.

General input

The method call determineLoS(competence(communiceren, *Score)*) is added to the initial plan base of the user model. This means that upon start, the MAD calculates the LoS.

Observation of the result

The current LoS is registered in the belief base through the predicate currentLoS(*Level*). This belief should change after a new initial LoS is determined

Set 1: select the initial level of scaffolding

Input: the next learning focus is 'communiceren', with

1. the score is 20 (between 0 and 24)
2. the score is 49 (between 25 and 49)
3. the score is 50 (between 50 and 74)
4. the score is 80 (between 75 and 100)

Expected functional behavior:

1. LoS is set to 3
2. LoS is set to 2
3. LoS is set to 1
4. LoS is set to 0

Observed functional behavior:

1. the beliefbase is changed by adding the belief currentLoS(3)
2. the beliefbase is changed by adding the belief currentLoS(2)
3. the beliefbase is changed by adding the belief currentLoS(1)
4. the beliefbase is changed by adding the belief currentLoS(0)

Acknowledgements

I would like to thank several people who helped me during the process of this project. First, I would like to thank my supervisors, John-Jules Meyer and Karel van den Bosch for their feedback and refreshing points of view. Very special thanks goes to Marieke Peeters, for her patience, trust and never-fading enthusiasm. Without her support, input, and knowledge, the thesis would not have become what it is.

I would also like to thank my roommates at TNO, Ruben, Pauline and Sylvia, for their helpful input and for the many hours we spent together working and procrastinating. Without them, the internship at TNO would not have been as interesting as it was. A rematch ping pong is still obligatory though.

Further thanks goes to my parents and other family, for their support and interest during the past year. An additional special thanks goes to Hazal Elif Yalvaç, for digitally supporting and encouraging me during the many hours of work. To illustrate this support and to conclude this thesis, I would like to quote one of our conversations that took place near the end of the project:

Christian: *No, Elif. I can't recall the taste of food ,nor the sound of water, nor the touch of grass. I'm naked in the dark, with nothing. No veil between me and the master thesis of fire. I can see it with my waking eyes!*

Elif: *Come on Christian. I can't carry the thesis for you. But I can carry you!*

Bibliography

[Bakkes et al., 2012] S. Bakkes, C. T. Tan and Y. Pisan, "Personalised Gaming," *Creative Technologies,* vol. 3, 2012.

[Bratman, 1987] M. Bratman, *Intention, Plans, and Practical Reason*, Harvard University Press, 1987.

[Brisson, 2012] A. Brisson, "Artificial Intelligence and Personalization Opportunities for Serious Games," *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference,* 2012.

[Brun, et al., 2010] A. Brun, A. Boyer and L. Razmerita, "Compass to Locate the User Model I Need: Building the Bridge between Researchers and Practitioners in User Modeling," in *User Modeling, Adaptation and personalization (UMAP)*, Big Island, United States, 2010.

[Chirico, 2013] U. Chirico, "JIProlog - Jave Internet Prolog," Cryptware, 2013. [Online]. Available: http://www.jiprolog.com/.

[Ci, etal., 2009] M. Ci, S. Marsella and D. Pynadath, "Directorial Control in a Decision-Theoretic Framework for Interactive Narrative," *Lecture Notes in Computer Science,* vol. 5915, pp. 221-233, 2009.

[Dastani, 2008] M. Dastani, "2APL: a practical agent programming language," *Autonomous agents and multi-agent systems,* vol. 16, no. 3, pp. 214-248, 2008.

[Dastani, 2009] M. Dastani, "Modular Rule-Based Programming in 2APL," in *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*, Information Science Reference, 2009.

[Dastani, 2013] M. Dastani, "2APL: A Practical Agent Programming Language," University of Utrecht, 2013. [Online]. Available: http://apapl.sourceforge.net/.

[Deepa, et al., 2012] N. Deepa, S. Priyadarsini, V. Mahadevan and J. Wilson, "Adaptive hypermedia using link annotation technology and recommender model (AHLARM)," *Journal of Theoretical and Applied Information Technology,* vol. 46, no. 2, pp. 1022-1028, 2012.

[Dignum, et al., 2011] V. Dignum and H. Aldewereld, "Opera," University Utrecht, 2011. [Online]. Available: http://www.cs.uu.nl/research/projects/opera/.

[Durrani, 1997] Q. S. Durrani, "Cognitive Modeling: A domain independent user modeling," in *IEEE International Conference on Computational Cybernetics and Simulation*, 1997.

[Elsom-Cook, 1993] M. Elsom-Cook, "Student modelling in intelligent tutoring systems," *Artificial Intelligence Review,* vol. 7, no. 3-4, pp. 227-240, 1993.

[Feight, et al., 2012] K. M. Feigh, M. C. Dorneich and C. C. Hayes, "Toward a Characterization of Adaptive Systems: A Framework for Researchers and System Designers," *Human Factors: The Journal of the Human Factors and Ergonomics Society,* vol. 54, no. 6, pp. 1008-1024, 2012.

[Ferdinandus, et al., 2013] G. Ferdinandus, M. Peeters, K. van den Bosch and J.-J. Meyer, "Automated Scenario Generation - Coupling Planning Techniques with Smart Objects.," in *Proceedings of the 5th International Conference on Computer Supported Education (CSEDU 2013)*, 2013.

[Fischer, 2001] G. Fischer, "User Modeling in Human-Computer Interaction," *User Modeling and User-Adapted Interaction,* vol. 11, pp. 65-86, 2001.

[Gazzaniga, et al., 2006] M. Gazzaniga, T. Heatherton, D. Halpern and S. Heine, *Psychological science*, 3rd ed., New York: WW Norton, 2006.

[Georgeff, et al., 1999] M. Georgeff, B. Pell, M. Pollack, M. Tambe and M. Wooldridge, "The belief-desire-intention model of agency," *Intelligent Agents V: Agents Theories, Architectures, and Languages,* pp. 1-10, 1999.

[Grootjen, et al., 2006] M. Grootjen, M. A. Neerincx and J. C. M. Van Weert, "Task-based interpretation of operator state information for adaptive support," in *Foudations of Augmented Cognition and 2nd Edition*, 2006.

[Harbers, 2011] M. Harbers, *Explaining agent behavior in virtual training*, PhD thesis, vol. 35, SIKS Dissertation Series, Utrecht University, 2011.

[Hargreaves, et al., 2010] T. Hargreaves, M. Nye and J. Burgress, "Making energy visible: A qualitative field study of how householders interact with feedback from smart energy monitors," *Energy Policy,* vol. 38, no. 10, pp. 6111-6119, 2010.

[Heller, et al., 2006] J. Heller, C. Steiner, C. Hockemeyer and D. Albert, "Competence-based knowledge structures for personalised learning," *International Journal on E-Learning,* vol. 5, no. 1, pp. 75-88, 2006.

[Heuvelink, 2009] A. Heuvelink, *Cognitive models for training simulations*, PhD thesis, Vols. 2009-24, SIKS Dissertation Series, Vrije Universiteit Amsterdam, 2009.

[Heuvelink, et al., 2008] A. Heuvelink and T. Mioch, "FeGA: a Feedback-Generating Agent," in *International Conference on Web Intelligence and Intelligent Agent Technology*, 2008.

[Jaques, et al., 2007] P. Jaques and R. Vicari, "A BDI approach to infer student's emotions in an intelligent learning environment," *Computers & Education,* vol. 49, no. 2, pp. 360-384, 2007.

[Jo, 2011a] C.-H. Jo, "CPSC 589, Seminar in Computer Science, Agent-based Software Engineering," Department of Computer Science, California State University Fullerton, 2011. [Online]. Available: http://jo.ecs.fullerton.edu/cpsc589/notes/cpsc589-06-AgentSE.pdf.

[Jo, 2011b] C.-H. Jo, "CPSC 589, Seminar in Computer Science, Agent-based Computing," Department of Computer Science, California State University Fullerton, 2011. [Online]. Available: http://jo.ecs.fullerton.edu/cpsc589/notes/cpsc589-05-AgentComputing.pdf.

[Jo, 2011c] C.-H. Jo, "CPSC 589, Seminar in Computer Science, Agent-based Applications," Department of Computer Science, California State University Fullerton, 2011. [Online]. Available: http://jo.ecs.fullerton.edu/cpsc589/notes/cpsc589-07-AgentApp.pdf.

[Johnson, et al., 2005] W. L. Johnson, H. H. Vilhjálmsson and S. Marsella, "Serious games for language learning: How much game, how much AI?," in *International Conference on Artificial Intelligence in Education*, 2005.

[Karam, et al., 2012] R. Karam, P. Fraternali, A. Bozzon and L. Galli, "Modeling end-users as contributors in human computation applications," *Model and Data Engineering,* pp. 3-15, 2012.

[Kay, 2000] J. Kay, "Stereotypes, student models and scrutability.," *Intelligent Tutoring Systems,* vol. 1839, pp. 19-30, 2000.

[Kay, et al., 2012] J. Kay and G. McGalla, "Coming of age: celebrating a quarter century of user modelling and personalization," *User Modeling and User-Adapted Interaction,* vol. 22, no. 1-2, pp. 1-7, 2012.

[Kobsa, 2001] A. Kobsa, "Generic User Modeling Systems," *User Modeling and User-Adapted Interaction,* vol. 11, pp. 49-63, 2001.

[Kobsa, 2007] A. Kobsa, "Generic User Modeling Systems," in *The Adaptive Web*, Springer, 2007, pp. 136-154.

[Korteling, et al., 2011] J. Korteling, A. Helsdingen, R. Sluimer, M. Emmerik and B. Kappe, "Transfer of Gaming: transfer of training in serious gaming," TNO innovation for life, 2011.

[Lorenz, et al., 2005] A. Lorenz, P. Dolog and J. Vassileva, "A specification for agent-based distributed user modelling in ubiquitous computing," in *Workshop on Decentralized, Agent Based and Social Approaches to User Modelling (DASUM) 9th International Conference on User Modeling*, 2005.

[McTear, 1993] M. F. McTear, "User modeling for adaptive computer systems: a survey of recent developments.," *Artificial intelligence review,* vol. 7, no. 3-4, pp. 157-184, 1993.

[Mioch, et al., 2013] T. Mioch, M. Peeters, J. Siljee, B. van Schoonhoven and F. Goethals, "*ePartners High-Level Functionality Architecture*," TNO Behavioural and Societal Sciences, 2013.

[Mohamad, et al., 2013] Y. Mohamad and C. Kouroupetroglou, "W3C: User Modeling," Research and Development Working Group, 2013. [Online]. Available: http://www.w3.org/WAI/RD/wiki/User_modeling.

[Morley, et al., 2004] D. Morley and K. Myers, "Balancing Formal and Practical Concerns in Agent Design," in *AAAI Workshop*, 2004.

[Norling, 2003] E. Norling, "Capturing the Quake Player: Using a BDI Agent to Model Human Behaviour," in *AAMAS '03 Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 2003.

[Nurmi, et al., 2007] P. Nurmi and T. Laine, "Introduction to user modeling," University of Helsinki, 2007. [Online]. Available: http://www.cs.helsinki.fi/u/ptnurmi/teaching/UM/Seminar_040907%20-%20Introduction.pdf.

[Nwana, 1990] H. Nwana, "Intelligent tutoring systems: an overview," *Artificial Intelligence Review,* vol. 4, pp. 251-277, 1990.

[Peeters, et al., 2011] M. Peeters, K. van den Bosch, J.-J. Meijer and M. Neerincx, "Scenario-based Training: Director's Cut," in *Proceedings of the 15th International Conference on Artificial Intelligence in Education*, 2011.

[Peeters, et al., 2012a] M. Peeters, K. van den Bosch, J.-J. Meijer and M. Neerincx, "Situated cognitive engineering: the requirements and design of automatically directed scenario-based training.," in *Proceedings of the conference on Advances in Computer Human Interaction (ACHI 2012)*, 2012.

[Peeters, et al., 2012b] M. Peeters, K. van den Bosch, J.-J. Meijer and M. Neerincx, "An Ontology for Integrating Didactics into a Serious Training Game.," in *Proceedings of the 1st International Workshop on Pedagogically-driven Serious Games (PDSG 2012)*, 2012.

[Peeters, 2014a] M. Peeters, "Personalized educational games: developing agent-supported scenario-based training", PhD thesis, University Utrecht, currently under review, 2014.

[Peeters, et al., 2014b] M. Peeters, K. van den Bosch, J.-J. Meyer and N. Mark, "The design and effect of automated directions during scenario-based training," *Computers & Education,* vol. 70, pp. 173-183, 2014.

[Peeters, CA] M. Peeters, K. van den Bosch, J.-J. Meijer and M. Neerincx, "Situated Ontology Engineering for Adaptive Educational Games," in *International Journal of Technology Enhanced Learning (IJTEL), special edition on: pedogogically-driven serious games*, CA (conditionally accepted).

[Razmerita, et al., 2003] L. Razmerita, A. Angehrn and A. Maedche, "Ontology-based user modeling for knowledge management systems," *User Modeling,* pp. 213-217, 2003.

[Razmerita, 2009] L. Razmerita, "User Modeling and Personalization of Advanced Information Systems," in *Encyclopedia of Information Science and Technology, Second Edition*, University of Galati, Romania, Information Resources Management Association, 2009.

[Razmerita, 2011] L. Razmerita, "An ontology-based framework for modeling user behavior—A case study in knowledge management," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans,* vol. 41, no. 4, pp. 772-783, 2011.

[Razmerita, et al., 2012] L. Razmerita, N. Thierry and K. Kirchner, "User Modeling and Attention Support: Towards a Framework of Personalization Techniques," in *CENTRIC 2012, The Fifth International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services*, Lisbon, Portugal, 2012.

[Russel, et al., 2003] S. Russell and P. Norvig, *Artifial Intelligence, A Modern Approach*, 2 ed., Pearson Education International, 2003.

[Sannes, 2011] P. Sannes, " *A Generic Framework For User Modeling*," MSc thesis, Vrije Universiteit Amsterdam, 2011.

[Schiaffino, et al., 2008] S. Schiaffino, P. Garcia and A. Amandi, "eTeacher: Providing personalized assistance to e-learning students," *Computers & Education,* vol. 51, pp. 1744-1754, 2008.

[Solinger, et al., 2005] D. Solinger, P. Ehlert and L. Rothkrantz, "*Creating a Dogfight Agent*," technical report, TUDelft, 2005, .

[Specht, et al., 2006] M. Specht, A. Lorenz and A. Zimmermann, "Towards a Framework for Distributed User Modelling for Ubiquitous Computing," in *1st Workshop on Decentralized, Agent Based and Social Approaches to User Modelling (DASUM2005)*, 2006.

[Toch, et al., 2012] E. Toch, Y. Wand and L. F. Cranor, "Personalization and privacy: a survey of privacy risks and remedies in personalization-based systems," *User Modeling and User-Adapted Interaction,* vol. 22, no. 1-2, pp. 203-220, 2012.

[van den Bosch, et al. 2009] K. van den Bosch, M. Harbers, A. Heuvelink and W. van Doesburg, "Intelligent Agents for Training On-board Fire Fighting," *Lecture Notes in Computer Science,* vol. 5620, pp. 463-472, 2009.

[van Merriënboer, et al., 2002] J. van Merriënboer, C. Richard and M. De Croock, "Blueprints for complex learning: The 4C/ID-model," *Educational Technology Research and Development,* vol. 50, no. 2, pp. 39-61, 2002.

[Vygotsky, 1978] L. S. Vygotsky, *Mind in Society: the Development of Higher Psychological Processes*, Cambridge: Harvard University Press, 1978.

[Webb, et al., 2001] G. Webb, M. Pazzani and D. Billsus, "Machine Learning for User Modeling," *Modeling and User-Adaptive Interaction,* vol. 11, pp. 19-29, 2001.

[Wooldridge, 2009] M. Wooldridge, *An introduction to multiagent systems*, second ed., Wiley, 2009.

[Yang, 2010] F.-J. Yang, "The ideology of intelligent tutoring systems," *Association for Computing Machinery Inroads,* vol. 1, no. 4, pp. 63-65, 2010.