An ITS View on AOSE

Rosa Maria Vicari*
Instituto de Informática, UFRGS, Porto Alegre, RS, Brazil
E-mail: rosa@inf.ufrgs.br
*Corresponding author

João Carlos Gluz
Programa de Pós-Graduação em Computação Aplicada (PIPCA),
UNISINOS, São Leopoldo, RS, Brazil
E-mail: jcgluz@unisinos.br

Abstract: The creation of ITS applications involves tasks that require the modelling of complex agents (i.e., human students); therefore, a cognitive agency approach is more appropriate than a weak agency approach. ITS applications need to know, plan, and understand several aspects of the other intelligent entities that inhabit the same environment. This is a viewpoint not considered in AOSE methodologies that this work intends to cover. Such aspect is the main innovation of this work, and also presents a critical requirement that should be fulfilled by applications before someone tries to use the AOSE methods proposed here. In this context, the paper presents a set of AOSE methods derived from ITS research, which define applicability criteria, design principles, and implementation guidelines to be applied in the software analysis, design and development process. The application of such methods is exemplified by several ITS developed in our research group.

Keywords: agent-oriented software engineering, BDI models, multiagent systems, intelligent tutoring systems.

Biographical notes: R. M. Vicari received her PhD in Computer and Engineering at University of Coimbra, Portugal and is Professor at Universidade Federal of Rio Grande do Sul (UFRGS), Brazil.

J. C. Gluz received his PhD in Computer Science at Universidade Federal of Rio Grande do Sul (UFRGS), Brazil and is Associate Professor at Universidade do Vale do Rio dos Sinos (UNISINOS), Brazil.

1 INTRODUCTION

Artificial Intelligence (AI) is probably the most anthropomorphic of all Computer Science research areas and this, in our opinion, should not be regarded as a weakness, instead, it should be considered an advantage, especially when we handle certain complex tasks or domains, such as Intelligent Tutoring Systems (ITS). AI should pursue its goals keeping the scientific parameters of accuracy, precision and reproducibility at the highest possible level, however, it is not possible to forget that this task requires the analysis of the basic phenomena related to human cognition and to human subjective experience or emotions. The several fields of AI should be able to answer on how they will try to emulate this kind of behaviour. A common approach is to employ an indirect method, trying to achieve the intended behaviour based on emergent properties of systems. The problem with such approach is that achieving the proper intended high-level abstract property or behaviour of the system is a complex and difficult goal. This problem has been identified in the research area of Agent Oriented Software Engineering (AOSE). Nowadays AOSE is a very active research line, presenting several distinct methodologies for software engineering (for a general presentation see Zambonelli & Omicini (2004)). The issue cited before is stated in the work by DeWolf and Holvoet (2005): “Agent-oriented methodologies today are mainly focused on engineering the microscopic issues, i.e. the agents, their rules, how they interact, etc., without explicitly engineering the required macroscopic behavior [...]. Engineering the macroscopic behavior is currently done in an ad-hoc manner because there is no well-defined process that guides engineers to address this issue.”. To us this problem appears to be related to the abstractions used to analyse, design and implement required emergent properties in multiagent systems. The “macroscopic behaviour” and the “microscopic issues” cited by these authors are not in the same abstraction level. It seems that for some complex applications, there is no method able to correlate microscopic properties with desired macroscopic behaviour.

To try a new top-down approach to this question, starting from higher abstraction levels, is interesting to see how Cognitive AI works. The goal of Cognitive AI is to analyse and propose cognitive models that present: (a) viable computational interpretations, (b) clear epistemological and psychological foundations and (c) precise formal specifications.
Each condition had its justification. The cognitive models should be computational from the theoretical point of view; otherwise, Cognitive AI cannot be considered part of Computer Science. Concepts used in these models should not be based only on naive intuition or common-sense psychology, but should be firmly rooted on epistemological and psychological foundations. The formal specification is the answer to avoid excessive anthropomorphism: the formal definition of any concept is independent of subjective belief, perception or emotion about this concept, even when the concept that is being formalised is the concept of “subjective belief” or “perception” or “emotion”.

Cognitive AI is far from achieving its objectives; however some results have been achieved in the last years, specially related to the Belief, Desire and Intentions (BDI) model (Cohen & Levesque, 1990; Rao & Georgeff, 1991), which probably is the most used Mental State approach for modeling agents and Multiagent Systems (MAS). One active research line of Cognitive AI is centred on the creation of ITS (Self, 1998) based on MAS ideas (Giraffa & Vicari, 1998). The ITS research field contributed with a very important design paradigm, the Student Model paradigm (Self, 1994; Dillenbourg & Self, 1992).

The main claim of the current work is that the Student Model paradigm, and general ITS MAS concepts, in conjunction with BDI models for agents could have an important application in the AOSE research area. From the experience accumulated from the design and development of several ITS, the authors will propose a set of applicability criteria, design principles, and development guidelines that can help to solve the matter on how to built systems that are able to achieve the desired properties such as autonomy, flexibility, and adaptability for this class of applications.

The paper is organised as follows. Section 2 summarises background and related work. Section 3 presents ITS models and architectures, and show standard results from ITS research. Section 4 provides background on Cognitive AI and BDI Cognitive Model for agents. Section 5 shows several ITS applications developed by our research group. Section 6 presents the set of proposed AOSE methods, defining applicability criteria, design principles, and implementation guidelines derived from ITS research. Section 7 brings final considerations and conclusions.

## 2 DISCUSSIONS AND RELATED WORK

The main claim of this work is that cognitive (BDI) agent models extended with ITS models and architectures are ready to be used as software engineering methods for the design and development of ITS applications. A necessary assumption of this claim is that the adoption of something similar to the Student Model paradigm should be an integral part of these methods.

The idea behind this paradigm is that ITS applications must create internal models of other subjects with whom they interact. These subjects are purposeful and intelligent entities, and may be artificial agents or human beings. This is the main difference between ITS and other kinds of applications. Agents that work in ITS applications not only are continuously trying to understand the environment where they "live", but also need to know, theorise, plan and understand several aspects of the other intelligent entities that inhabit the same environment. All pedagogical relationships between the ITS and their students are based on these models. Indeed, it is a premise of our work on ITS that, without some psychological subjective knowledge about these other entities, ITS are unable to efficiently execute their teaching tasks. This is not a requirement for all sorts of MAS or agents, but it is unique to ITS applications.

This need to support internal models of students, implies that model related criteria analysed in Tran et al. (2004) work are the determinant criteria to verify if some AOSE methodology is useful for ITS. Considering the psychological aspects of ITS, it is necessary that personality and adaptability criteria be supported, because these criteria verify, respectively, if the models support and represent attributes of a “believable” human character, or have the ability to learn and improve with experience. In the case of ITS is also necessary to preserve the continuity of teaching-learning process during long periods. This requires the support of the temporal continuity criterion, which ensures that models maintain the persistence of identity and state over long periods. Unfortunately, all methodologies surveyed in Tran et al. (2004) work (MaSE, GAIA, BDIM, MAS-CommonKADS and Prometheus) do not support the personality, adaptability, and temporal continuity criteria.

However, this is not the only problem with these and with other AOSE methodologies. Considering the psychological and cognitive aspects of student models, we need a methodology where cognitive high-level agent abstractions are considered since the beginning of the software engineering process, including the requirements engineering phase. The modelling of the application domain with these abstractions must be an integral part of the methodology.

Several AOSE methodologies proposed to date were influenced by Object-Oriented (OO) methodologies (Giorgini & Henderson-Sellers, 2005). They adopt a weaker non-mentalistic approach to the notion of agency. This is the case, for instance, of GAIA (Zambonelli et al., 2003), MaSE (Deloach, 2005), MESSAGE/UML (Caire et al., 2001) and INGENIAS (Gomez-Sanz & Pavon, 2003) methodologies. These methodologies will be analysed more carefully because they represent important branches of the “genealogy tree” presented by Giorgini & Henderson-Sellers (2005) work, which shows the influences of OO methodologies on AOSE methodologies.

In all these methodologies, the mental and social concepts related to BDI models are almost ignored in behalf of concepts related to organisations, groups, roles for agents, and similar notions. GAIA, for instance, was proposed as a generic MAS design methodology that focuses on abstraction concepts related to organisations and roles. Organisations are important entities to be modelled, and roles are the functional elements of these organisations. Agent abstractions
are considered only under the perspective of roles in these organisations, so concepts related to roles are more significant than agent concepts. Surely, GAIA is a good option to model structured (organised) societies with clear roles established for all members. However, it is not clear how GAIA can be generalised to model other societies not so well organised, neither how it can be used to model small teams, where the particular behaviour of each agent must be fully considered in the model. Examples are how to model cooperative classroom work based on small and dynamic groups, or model the individualized social interactions with students based on their student model. In our opinion, GAIA was simply not designed to handle such type of societies.

MaSE is a generic AOSE methodology, which considers the use of BDI models as a class of individual agent architectures. The problem with MaSE is that BDI concepts are only taken into account in the design phase, when agent classes must be selected. Our opinion is that this is too late when it comes to ITS applications. If abstractions and concepts derived from a cognitive agent’s model, as the BDI model, are useful for some application, then they should be considered from the beginning of the software engineering process, including the analysis and requirements elicitation phase.

MESSAGE/UML methodology and its correlated framework INGENIAS, provide good design tools to specify the software architecture of individual agents, but they cannot be applied to the analysis phase. MESSAGE/UML is not completely developed to handle MAS concepts. There is an organisation level, but this level does not provide good abstractions as the one used to model individual agents.

The case of Tropos (Giorgini et al, 2004) also deserves a careful analysis because, different from previous cases, this methodology takes into account mentalistic concepts from the start. Tropos adopts the i* mentalistic agent modelling framework, which provides a graphical notation for BDI models (Dasgupta, et al., 2005). Tropos uses mentalistic concepts in all phases of software development, from analysis down to the implementation. This puts Tropos as the best candidate for an ITS development methodology. In general, this is true, and Tropos can be seen as a complementary methodology for the set of AOSE methods to be proposed in this work. However, from our viewpoint, Tropos fall short to be a complete methodology for design and development of ITS, because of its approach to model social relationships. Tropos adopts the same idea of GAIA to model social relationships. It regards the concept of human organisations, in which agents must play appropriate roles, as the main metaphor to model social relationships. In our opinion, this is not enough for ITS applications, because pedagogical relationships, which are the main social relationships of these applications, should be established during runtime based on the student model created by the ITS through the interaction with their students. Modelling these relationships through organisation’s roles does not capture the inherent complexity of social relationships based on cognitive models.

Summing up, to our knowledge AOSE methodologies proposed to this date are insufficient for the development of applications we are aiming at. To try to solve the issue we consider a top-down model-driven approach to analyse, design and develop these applications. The idea is that, when cognitive models for agents are considered from the beginning, then the problem on how to achieve desired high-level properties would become an integral part of the analysis and design process.

This approach resulted in the set of AOSE methods proposed in Section 6. These methods are not intended to constitute a complete methodology for software engineering based on ITS practices. They should be considered as a set of complementary techniques to be used together with current software engineering methodologies (including AOSE methodologies), always when necessary. Indeed, when we develop the applications described in Section 5, we took this combination with other software engineering tools for granted. Always that possible, we used standard tools as FIPA-ACL (FIPA, 2002) or AUML (Odell et al, 2001), to model aspects of these applications.

3 ITS MODELS AND ARCHITECTURES

We intend to provide an overview of Intelligent Tutoring Systems focusing on their conception using multiagent approach and affective and cognitive student modelling.

ITS applications are educational softwares containing AI components. The software tracks students’ work, tailoring feedback and hints along the way. By collecting information on a particular student’s performance, the software can make inferences about strengths, weaknesses, and can suggest additional work.

The traditional architecture of an ITS is composed of a domain base (knowledge about a subject), student model (a representation of the student cognitive state about the subject), and teaching strategies to guide the student.

3.1 Pedagogical Agents and Multiagent ITS

In recent years many educational systems, especially ITS, are being implemented according to the agent paradigm. The intelligent agents that have an educational or pedagogical role to facilitate or improve learning are called Pedagogical Agents (Gürer, 1998). They can be modelled as cooperative agents that work in the background as part of the architecture of the educational system, or as personal and animated agents that interact with the user.

In the first case, the educational system is modelled and implemented through a multiagent approach, were each agent has a specific function in the system. These agents act in the background, i.e., users are not able to see them, and change information among themselves in order to carry out actions that are appropriate for better learning. In this scope, we highlight the works of Frasson, et al. (2005), and Bica et al. (2006). According to Giraffa and Vicari (1998), the architectures based on this approach are variations of the tra-
ditional and functional architecture of an ITS (domain base, student model, teaching strategies), where one or more agents implement each function of the tutor. The control is distributed among agents. However, the user sees the single system, while internally it is composed of a society of agents.

In the second case, agents are personalised and animated agents represented by a lifelike character that interacts with the student through voice, emotional attitudes, and gestures. Some examples of animated pedagogical agents are: Vincet (Paiva & Machado, 1999), Steve (Rickel & Johnson, 1998), and Cosmos (Lester et al., 1999).

The two major advantages of the use of agents in the conception of educational software are: modularity and openness. As the agents are independent, they are powerful tools to make the ITS a modular system. Some efforts have been carried out towards the construction of components of tutors as agents who can be joined to form an ITS. Moreover, if each agent is an exclusive module independent from the others, it is easier to add other agents to these systems, which will carry out new functionalities.

As the agents are autonomous, they need only to know information on how to interact with other agents to be integrated to the system (what type of new information the system expects the agent to send, for example).

The MAS modularity also enables the management of bigger and complex problems: each agent can be specialised in its own tasks in the problem solving process (can have specialised knowledge and abilities for these tasks). This modularity simplifies the design and development of the educational system. The developer can concentrate on knowledge representation, granularity analysis, and ways of reasoning that are different for each agent. Such modularity also allows components to be reused in different systems.

Besides, the distributed nature of multiagent architectures lets the functionality of an educational system be distributed through a computer network into different platforms. This distribution permits that ITS be built from several components that are in different platforms, allowing the use of appropriate tools without worrying about the details of these platforms. The distributed nature of these architectures also allows partial parallel processing. Early in 1998, Johnson (Johnson et al., 1998) said that the use of a multiagent paradigm permits relatively easier migration from a single to a multi-user system.

3.2 ITS Architecture

To adapt the actions of a tutoring system to the student’s necessities is a complex process that requires a variety of knowledge and expertise, as well as problem resolution capacities, strategies of man-computer interaction, evaluation of pedagogical input, and presentation of multimedia information. Splitting this process into appropriate components, which are autonomous, pro-active, and flexible, can reduce the complexity of building a tutor.

Dillenbourg and Self (1992) describe a formal abstract framework that shows how basic entities (modules or agents) of an ITS can be structurally organised in several abstraction layers. This framework also shows the relationships among the entities of each layer and what kind of knowledge is related to these layers. The abstract layers of the framework form the 'vertical' dimension (see Figure 1 upper portion). They are based upon the computational distinction among concrete behaviour, behavioural knowledge, and conceptual knowledge. The basic relationship in the vertical dimension is about consistency among abstraction layers. This includes the consistency between the learner’s concrete behaviour, and the knowledge about possible behaviours, and the consistency between this knowledge and the conceptual knowledge about the learning domain. The sub-classification of entities forms the 'horizontal' dimension of the framework (see Figure 1 bottom portion). It assumes the existence of three entities: the “system”, “the learner”, and “the system’s representation of the learner” (the student model). The identification of discrepancies among these entities forms the basic relationship among them (see Figure 1). The interaction between the learner and the system is contextualised as a search space problem. Methods for setting up the search space in learner models and for carrying out the search process were also reviewed by Dillenbourg and Self (1992). .

Such interactions of pedagogical nature are the most important units in teaching and learning process analysis. The challenge is in the search of symmetry between man and machine. Such symmetry should allow the same possibilities to the user and the system as symmetric actions, and symmetric rights for decision taking. In an asymmetric mode, one agent has always the decisive point, as there is no space for a real negotiation. In the symmetric mode, there is no predefined winner, as conflicts need to be solved through negotiation. The cognitive process that triggers an explicit declaration to justify an argument or to refuse the partner’s point of view is a likely reason to explain why collaborative learning is more efficient than lonely learning.

![Figure 1](http://example.com/diagram.png)

*Figure 1 Vertical and Horizontal Dimensions of Dillenbourg’s Abstract Framework*
The main functions of ITS systems (explanation, education, and evaluation) are traditionally implemented as one-way mechanisms, which means that the system has control over interactions. Recent work with pedagogical negotiation processes (Gluz et al. 2006a), tries to treat these interactions as bilateral processes. The model is built collaboratively and there are some moments of negotiation. It is clear that for a negotiation to take place there must be a level of latitude available to agents, otherwise anything can be negotiated. Discussions about the use of negotiation mechanisms in learning environments are not recent. According to Self (1992), there are two major motivations for the use of negotiation in ITS: i) make it possible to foster discussions about how to proceed, which strategy to follow and which example to look for in an attempt to decrease the control that is typical of ITS, and ii) give room for discussions that yield different viewpoints (different beliefs), providing that the agent (tutor) is not infallible. Dillenbourg and Self (1992) say that human partners do not negotiate a single shared representation, but they develop several shared representations. They move in a mosaic of different negotiation spaces.

As a contrasting view about ITS evolution and properties, consult the work of Sklar and Richards (2006), which presents an overall view of the state of the art in ITS research. However, there are several important differences between the approach followed by Sklar and Richards from the approach used in this work. Sklar and Richards’s reference architecture for ITS systems supposes a central control component to integrate the system. We, otherwise, are modelling our ITS by multiagent systems where each agent is composed by a knowledge base, reasoning abilities, and, more important, is able to make autonomous decisions, without depending on a central component. Each agent can takes its decision based on Rationality, Logic, Decision Theory and Utility Theory, according to its belief and affective state. The work of Sklar and Richards also relies on a viewpoint about ITS research problems that is a little different from the approach adopted in the present work. In the final discussion they put clear that the difficulties to build effective tutoring systems rely on the modelling of learning domains, that is, in “deep understanding of the knowledge domain to be taught” (Sklar and Richards, 2005, p. 772). Although building adequate models of complex learning domains can be a very arduous task, we consider that creating useful cognitive models for students is also a very challenging task and an important goal for ITS research.

We also can point out the work of Mark and Greer (1993), which offer an overview of evaluation methodologies for ITS architectures. This work discusses several summative and formative evaluation techniques, that can be used to evaluate the architectural, behavioural and educational aspects of ITS. However, the focus of this evaluation analysis is a little bit limited for the present work, because it does not take into consideration the application of concepts of agents for ITS research, neither considers architectures and methodologies for design and development of ITS based on agents and multiagent systems.

4 COGNITIVE AI AND STUDENT MODELS

There is still a huge gap in our knowledge on how the natural mind works. However, by looking at the structure of mental states we may build replicas of these systems that can help us understand how these biological systems work. This can be the breakthrough on how to design cognitive agents turned to particular problem domains.

The Mental States approach allows us to trace more precisely the intrinsic dynamics of interactions between tutor and students. These results can be used to improve modelling and help us build better student models and tutor models.

Trying to go beyond the notion of behaviour, where the mental states approach already allows us to build appropriate models, we have been evolving our ITS architectures and models to represent some affective aspects of learners, like effort, confidence, and independence. The affective information improves the system, allowing it to be more helpful to the student. We believe that the use of mental states and affective aspects allows for an accurate selection of pedagogical strategies. In fact, researchers in education believe that the educational computing environments would be more pedagogically effective if they had mechanisms to show and recognise the student’s emotions. As sympathetic teachers usually do in their classes, these pedagogical environments should observe students, try to recognise their emotions and respond in a kind manner, giving them emotional support while motivating and encouraging them. In order to transmit emotions, some educational systems are implemented as animated pedagogical agents. When designed to interact affectively with the student, these agents show emotive behaviour through the animation of a lifelike character (see PAT agent in Section 5, for an example of this application).

4.1 Mental States for Student Models

To represent student models using the Mental State approach, first is necessary to choose an appropriate cognitive model of these mental states. The BDI model is one of the possible cognition models of the Mental State approach. Because BDI models allow a more granulated and proactive representation of the student’s mental states during problem solving activities, we considered it adequate for student model representation. In addition to beliefs, the BDI model is composed of proactive mental states (desires and intentions). Therefore, the student model can better represent the student’s actions in the ITS environment, and the desires and intentions behind these actions. Because this representation guides the tutor’s pedagogic tactics, it is possible to be more effective in teaching.

This also allows the student model to be modelled and developed as an agent that has knowledge about the student model, and that communicates this knowledge to others agents in the environment.

We have been exploring the concepts of agents and BDI models as abstractions to describe and design ITS, for a
while (Giraffa and Viccari, 1998). For instance, in MCOE and AMPLIA ITS (see Section 5), the student model can also make decisions about the student actions. It is an aspect that differs our work from others. Another important characteristic is that the desires may have contradictions with relation to other desires or beliefs, while the beliefs allow that agents constantly update their views of the world. Such characteristics of BDI are important to better represent the ‘chorography’ of teaching and learning interaction.

Agent and multiagent architectures are also key concepts to build ITS. However, because these architectures are quite implementation-oriented, providing schemas to develop agents and systems, they are inadequate as an analysis tool for the application domain of these agents. In the special case of pedagogical agents, it is not easy to respect the pedagogical theoretical foundations of such agents while someone is building them. These architectures are much oriented to implementation aspects, to allow the proper distancing (abstraction level) in the analysis process. To this end, formal models come into place as we are interested in both describing and analysing the autonomous behaviour of (artificial and natural) agents in ITS. These methods provide an adequate abstraction level to analyse and describe the phenomena that occur in these systems, and, consequently, the domains where pedagogical agents must work.

In effect, the application of formal approaches to understand or conceptualise aspects of educational processes brought ITS research close to cognitive agent models. The formal analysis presented by Self (1994) shows that there is a relationship among several areas of AI, like machine learning, cognitive agent modelling, and ITS research. The formal model defined by Self is derived from various areas of theoretical AI including BDI modal logics appropriate for cognitive and communication modelling of agents.

Another purpose of having formal models of pedagogical agents is to reduce the gap between specification and implementation. To achieve part of this goal, we have been using BDI architectures with event calculus as a logical support for time and actions. Examples of this are the MCOE and PAT BDI student models (see Section 5), which are based on ELP Logics (Mora et al., 1998). Recently, we started to work with probabilistic belief models, based on Bayesian Networks (BN) that are integrated to the BDI model through a Probabilistic Modal Logic (Gluz et al., 2006b). Our current research includes looking for a formal model that can be used as a framework to specify and test (simulate, build, and execute) pedagogical agents.

Following this analysis the student model can be defined as the representation of a set of learners’ characteristics and attitudes used to achieve individualised and appropriate interaction between computational environments and student. Its objective is to understand the exploratory behaviour of the learner in order to offer the necessary support whilst also maintaining a sense of control and freedom.

The student model has a description of the student knowledge, student skills, strengths, and weaknesses. The model can also take into account the domain of the problem that was taught, and the student’s learning process (which is the case of our applications). It should be updated when new information, such as affective aspects, is obtained.

4.2 An Extended BDI Model

In our systems, the student model is built over the cognitive structure of pedagogical agents. The general cognitive structure of these agents is formed by <B,D,I>T tuples where B is the set of agent’s beliefs, D is the set of agent’s desires, I is the set of agent’s intentions and T is the set of time axioms. As can be seen in modelling examples of MCOE and PAT (see Section 5), such structure is used by tutoring agents to create the student’s cognitive model. The tutor model is composed of beliefs, desires and intentions of the tutor agent about the students, at a particular instant of time.

The set of desires of some agent is a set of sentences DES(Ag,P,At) if Beliefs, where Ag is an agent identification, P is a property, At is a list of attributes of the desire, and Beliefs a condition in which the agent believes. Desires are related to the state of affairs the agent eventually wants to bring about. In our kind of applications, desires represent particular situations of pedagogical interest to be achieved by the agent. Desires do not necessarily drive the agent to act. The fact of an agent having a desire does not mean it will act to satisfy it. It means that before such an agent decides what to do, it will be engaged in a reasoning process, confronting its desires (the state of affairs it wants to bring about) with its beliefs (the current circumstances and constraints the world imposes). The agent will choose those desires that are possible according to some criteria.

Beliefs are the agent’s information attitude. They represent the information agents have about the environment, other agents, and about themselves. Set B contains sentences describing the problem domain using ELP logics (Mora et al., 1998). If an agent Ag believes that a property P holds at a time T if, from B and T, then the agent can deduce the belief BEL(Ag,P) for the time T. We assume that the agent continuously updates its beliefs to reflect changes it detects in the environment. We also assume that, whenever a new belief is added to the beliefs set, consistency is maintained.

The explicit inclusion of time was an extension that we made over BDI classical models. It was necessary because, in a pedagogical context, students can repetitively change their beliefs according to interactions with the teacher, during the teaching-learning process. This situation must be appropriately modelled by ITS applications; therefore, the student models must somehow represent time.

By definition, there are no constraints on the agent’s desires. However, the learning experience promoted by the ITS application must be adapted to the learner’s individual needs at each time. Our systems incorporate this idea and present an alternative way to investigate the personal teaching strategy connected with each student style based on her mental states set. For that end, the tutor agent must select a particular intention in accordance with its desires and with the current state of the teaching/learning process.
Intentions are characterised by a choice of a state of affairs to achieve, and a commitment to this choice. Thus, intentions are seen as a compromise the agent assumes with a specific possible future. This means that, different from desires, an intention may not be contradictory with other intentions, as it would not be rational for an agent to act in order to achieve incompatible states. Intentions should also be supported by the agent's beliefs. That is, it would not be rational for an agent to intend something it does not believe possible. Once an intention is adopted, the agent will pursue that intention, planning actions to accomplish it, re-planning when a failure occurs, and so forth. Agents must adopt these actions, as means used to achieve intentions.

Rationality constraints should be satisfied by intentions: an agent should not intend something at a time that has already passed; an agent should not intend something it believes is already fulfilled or that will be fulfilled with no efforts by the agent; an agent only intends something it believes possible to be achieved. When designing an agent, we specify only the agent's beliefs and desires. It is up to the agent to choose its intentions appropriately from its desires. Such rationality constraints must also be guaranteed during this selection process.

Our agents choose their intentions from two different sources: from its desires, and as refinements from other intentions. An agent may have conflicting desires, i.e., desires that are not jointly achievable. Intentions, on the other hand, are restricted by rationality constraints (in the case of applications like PAT agent we are also exploring the use of emotional constraints). Thus, agents must select only the desires that are related to those constraints. It is necessary to determine the subsets of desires that are relevant according to the agent's current beliefs. Afterwards, it is necessary to determine jointly achievable desires. There may be more than one subset of jointly achievable relevant desires. Therefore, we should indicate which of these subsets should be adopted as intentions. This is done through a preference relation defined on the attributes of desires, which specifies the importance of desires.

Agents in our applications should prefer first to satisfy most important desires. Additionally, in preferring the most important ones, the agent should adopt as much desires as possible. The selection is made by combining the different forms of non-monotonic reasoning provided by the logical formalism. In our applications, desires will be adopted as intentions if the tutoring agent believes that there is a sequence of actions, which the student can execute to satisfy these intentions.

Once the tutoring agent adopts its intentions, it will start planning how to achieve those intentions (for instance, on how to help student in a particular study topic). At this moment, the tutoring agent depends on knowledge about the students' mental states in order to decide what to do. Since it does not have (yet) this knowledge, it waits.

The next step is to define when this should perform all this reasoning about intentions. We argue that it is not enough to state that an agent should revise its intentions when it believes a certain condition holds, just like to believe that an intention has been satisfied or that it is no longer possible to satisfy. Hence, the agent needs to verify its beliefs constantly. Instead, we believe it is necessary to define, along with those conditions, a mechanism that triggers the reasoning process without imposing a significant additional burden on the agent. As a result, the tutoring agent inserts triggers in its belief that will make itself reconsider its options when new interaction occurs.

The tutoring agent should start the belief reviewing process about the student, when some particular condition, such as not having received any response from the student for an excessive period, is satisfied. This results in several new actions taken by the tutor agent, as sending messages to the student, starting interaction with sensors, for example.

Our approach is to define those conditions that make the agent start reasoning about intentions as constraints over its beliefs. Note that we assume that an agent constantly has to maintain its set of beliefs consistent whenever new facts are incorporated. When the agent reviews its beliefs and one of the conditions for revising intentions hold, a contradiction is raised. The intention revision process is triggered when one of these constraints is violated.

We are using several techniques to implement this decision process. In MCOE system and PAT agent we add production rules to the BDI model, to represent these constraints, identify, and execute appropriate pedagogical strategies and didactic tactics. Pedagogical agents in AMPLIA system use Influence Diagrams to select appropriate teaching strategies and tactics.

Another contribution of our research to BDI student models, which is related to the research goal of how to represent affective aspects of learners in ITS, is to regard affective aspects as additional constraints, besides rational constraints, when deciding what particular desire will become a new intention. Currently we are exploring the use of student confidence, effort and persistence as affective information associated to desires and used in the decision process. PAT agent is an example of this research, being able to use affective states to select desires.

5 ITS APPLICATIONS

As a research group, we developed several ITS applications to explore the use of agent technology and BDI models. Using agent technology to design, represent, and execute cognitive student models, we developed pro-active models similar to real students. This is one important contribution to ITS area. We will present three ITS applications that were the base for the elaboration of the AOSE methods proposed in Section 6 and provide examples on how such methods were employed.

The first one, MCOE MAS system, is a game where two students play together. MCOE has two different student model agents and a tutor agent that are cognitive BDI agents. The tutor agent has 24 different action plans that can act at the environment. In one of our experiments (Mora et al., 1998), we used the X-BDI interpreter of the ELP logical
language, which make possible to represent time information associated with each belief, through event calculus. Beyond tutor and student cognitive agents, the game environment is composed of several reactive agents (see subsection 5.1).

In our second experience with the use of BDI to design pedagogical agents, we have only one cognitive agent that works as an animated character: the PAT agent. PAT infers aspects of the affective state of the student in teaching/learning situations. Through PAT, the tutoring system obtains information about the student affective states and acts on the tutoring environment. In both cases, the BDI student model is generated from the student’s actions.

Our more recent application is AMPLIA MAS (Vicari et al., 2003; Flores et al., 2005). Its main function is to support the development of diagnostic reasoning and modelling of hypotheses in the medical area, by using BN for probabilistic knowledge representation. AMPLIA was a successful test bed for several experiences, which were aimed at verifying how our experience in ITS can be used in teaching in the medical area, and how to combine subjective (bayesian) probabilities with BDI models in an ITS.

5.1 MCOE System

MCOE system (Giraffa, Mora & Vicari, 1998) was one of our first works using BDI to build students and tutors models. The MCOE architecture (see Figure 2) is composed of a society of agents that work to achieve a common goal, to fight against the pollution resulting from foreign elements, and to maintain the environment balanced. MCOE has reactive agents such as the bottom of the lake, microorganisms - plankton, water, plants, and three types of fish. MCOE has also cognitive agents as the tutor agent, and student agents represented in the system by characters. These cognitive agents are modelled by using the BDI model.

The model of each BDI agent is composed of a set of beliefs, desires and expectations (beliefs in the future). The set of the students’ mental states is obtained through inference sensors that track students’ attitudes. The MCOE dialogues are composed of a sequence of speech acts.

![Figure 2 MCOE Cognitive Agents Architecture](image)

MCOE cognitive agents work with time-extended <B,D,I>T models presented in Section 4. In terms of interactions, the tutor receives information about each student model. Information is composed of the mental states of each student, the students’ action and the environment sensors. Sensors pick up information about energy levels checked by a control. The tutor uses the information to select a specific strategy with the associated pedagogic tactic. The system works as if the tutor could hear and see what happens in the environment. The main purpose of the system is to aid the student to maintain an adequate energy level in the environment (a lake). It believes that it may aid by sending messages to students. The contents of these messages will depend on the pedagogical strategy adopted by the tutor.

The following modelling example shows how the mental states of the tutor and student agent may be represented (at some moment) by desires and beliefs:

\[
\begin{align*}
\text{BEL}(\text{tutor, aid students}) & \text{ if } \text{BEL}(\text{tutor, send message}) \text{ if } \\
\text{DES}(\text{tutor, aid students})\text{ if } & \text{BEL}(\text{tutor, send message})
\end{align*}
\]

When there is only one desire, this desire is the only candidate to intention. It will be adopted as intentions if the tutor believes there is a sequence of actions the student can execute. At this moment, the tutor depends on knowledge on the student’s mental states to decide what to do. Since it does not have (yet) this knowledge, it then waits. The tutor includes its beliefs, which trigger and make it reconsider its options when interaction occurs. For instance, the tutor has the following belief:

\[
\begin{align*}
\text{BEL}(\text{tutor, next(BEL(\text{Student, receive aid}))}) & \text{ if } \\
\text{BEL}(\text{energy level, ecometer} \geq 70) & \text{ if } \\
\text{message(“Continue paying attention to energy level”)}
\end{align*}
\]

This means that if it believes that the student expects some help, and it believes the energy level is above 70, than the only help it can offer is to advise the student to pay attention to the energy level. This belief could be placed as a trigger, assuring that when the pre-conditions for sending a message are satisfied, it is able to aid the student. The agent models then transform the desire into an intention to satisfy the tutor, which sends the message:

\[
\text{BEL}(\text{tutor, next(BEL(\text{Student, receive aid}))}) & \text{ if } \\
\text{BEL}(\text{energy level, ecometer} (EE), EE \geq 70)
\]

Once the tutor has that intention, it uses its beliefs about the environment (tool, energy level, foreign elements, and scenery elements), on how to fight against pollution in the environment (different strategies on the use of the tools to eliminate pollution), on how to advise the students in order to help them to control the environment, and so on.

Contributions of the MCOE system should be compared with the state of the art in ITS research around 1999, when it was developed. Since then we have been running some experimental evaluations, using traditional ITS as tools for explanatory class, and multiagent ITS to cooperative work using local network and Web. MCOE was one of the first ITS modelled using the BDI paradigm. These experiments were made in order to find the advantages and restrictions of using new approaches to build better ITS. For more details about these results see (Giraffa, Mora & Vicari, 1998).

5.2 PAT Agent
PAT agent (Jaques & Vicari, 2007) was developed with the purpose of inferring student’s emotions during teaching and learning processes. This agent was developed as an individual agent with more restricted domain of application than MCOE. The main idea of PAT is to proceed with affective recognition and diagnosis through Mental States. The cognitive model used by PAT follows the <B,D,I>T models. The PAT agent is divided in two parts: the Mind component that does infer student’s emotions and the Body component, which performs the animated interaction with the student (see Figure 3). The Mind component was modelled in X-BDI as an autonomous individual agent that infers emotions.

**Figure 3  PAT Agent Architecture**

Main pedagogical relationships to be identified and pursued by PAT are driven by student’s emotions, which can be inferred by the student’s observable behaviour (the student’s actions), in the interface of the learning environment. Examples of observable behaviour are: time to accomplish an exercise, success or failure in tasks, request or refuse help, etc. In such cases, the system predicts the user’s emotions based on a cognitive psychological model of emotions. We used the OCC model (Ortony, Clore & Collins, 1988) to predict emotions. The idea is to use the information provided by the psychological model in order to build an interpretation of a situation from the user’s viewpoint and to reason about which emotion this interpretation leads to. The OCC model is based on the cognitive approach to understand emotions that explains the origins of emotions by describing the cognitive processes that elicit them. It is called appraisal by the cognitive theoreticians of emotions. The main idea of the appraisal theory is that “emotions are elicited and differentiated on the basis of a person’s subjective evaluation of the personal significance of a situation, event or object on a number of dimensions or criteria” (Scherer, 2000). The appraisal consists in this evaluation of value of personal meaning for a situation. In our work, we acknowledg joy and distress, satisfaction and disappointment, gratitude and anger, as well as pride and shame emotions.

Let us observe how the Mind component of PAT selects the affective tactics from the following scenario: the student has a performance goal and feels both distress and disappointment because he provided an incorrect answer to an exercise. The Mind module receives the following information from the agent's sensors: 

\[
\begin{align*}
&\text{current_time}(2), \text{sense}(\text{student\_goal}(\text{performance}), 1), \\
&\text{current_time}(3), \text{sense}(\text{event}(\not\text{correct\_answer}), 2), \\
&\text{sense}(\text{effort}(\text{high}), 2). \\
\end{align*}
\]

The sensor notifies the module that the student has performance goals, that the effort is considered high, and that an event happened - the student provided an incorrect answer to the exercise. Therefore, the PAT agent activates the desires "apply_tactics" and "emotion_sent" as intentions. The desire “emotion_sent” aims at sending to the diagnostic agent the student’s emotions. It uses this information to help the teaching agent choose the pedagogical tactics that are adequate in the cognitive and affective viewpoint. The desire "apply_tactics" is responsible for choosing the affective tactics that will be applied, according to the affective state. The teaching agent was identified as "ag".

/* The agent’s desires to apply an affective tactic */
\[ \text{DES}(\text{ag, apply_tactics(Tactic)}, \text{Tf}, [0.6]) \]

\[ \text{BEL}(\text{ag, apply_tactics(Tactic)}) \]

/* The agent’s desires to send the student’s emotions to the diagnostic agent */
\[ \text{DES}(\text{ag, emotion_sent(Emotion,Intensity)}, \text{Tf}, [0.8]) \]

\[ \text{BEL}(\text{ag, student_emotion(Emotion)}) \]

In that case, in order of the PAT agent to satisfy its intention of applying an affective tactic, it must accomplish the action of sending this tactic to the agent’s actuator (“send_tactic” predicate). To satisfy the intention “emotion_sent”, it needs to send the emotion to the diagnostic agent (“send_emotion” predicate).

In order to send the emotions to the diagnostic agent, the tutor agent must know the student’s emotions. It infers the student’s emotions from the following beliefs:

/* The student is displeased with the event */
\[ \text{BEL}(\text{ag, event_pleasantness(\not\text{correct\_answer}, displeased)}) \]

/* It is a prospect of an event */
\[ \text{BEL}(\text{ag, is_prospect_event(\not\text{correct\_answer})}) \]

The student is displeased with the event because the event is undesirable, or it is desirable but it did not happen. When the student is displeased, it experiences distress and disappointment if it is the prospect of an event that was confirmed (“is_prospect_event” predicate). It is the case of the event “\not\text{correct\_task\_answer}” when the student accomplishes a task and has an expectation that this event would
happen. It is indicated by the predicate “BEL (ag-is_mediator_action)”. It is also important to verify the value of the variables that affect how intense emotions are:

\[
\text{BEL (ag, emotion_intensity(disappointment, high)) if }
\text{BEL (ag, effort(high)).}
\]

The variables that affect the emotion’s intensity are effort, realization, unexpectedness and undesirability for disappointment, and undesirability for distress. If one of these variables has a higher value (marked with high), the student experiences the specific emotion with high intensity, and otherwise he experiences emotions with medium intensity. The sensor of the body module sends the values of the variables that affect the emotion’s intensity and these values can be medium or high. The sensor is responsible for identifying the value of these variables with questionnaires and student’s observable behaviours.

Finally, the agent chooses the tactics through beliefs. The affective tactics are: (1) to increase the student’s self-ability; (2) to increase the student’s effort; and (3) to offer help to the student. The contribution of PAT agent student models was the inference of affective states using the cognitive approach and the use of this information inside the BDI mechanism in order to select a particular desire to become an intention of the tutor agent (see (Jaques&Vicca, 2007)).

### 5.3 AMPLIA System

AMPLIA MAS architecture works with three types of agents: Learner, Mediator, and Domain agents (Figure 4). The Learner agent corresponds to the student model. The Domain agent stores knowledge about the domain and evaluates knowledge modelled by the student. The Mediator agent chooses pedagogical strategies to help the student.

![Figure 4 AMPLIA System Architecture](image)

Figure 4 AMPLIA System Architecture

Logical knowledge shared by AMPLIA agents is represented by logical and probabilistic beliefs. \(\text{Sol(CoS,L,S)}\), and \(\text{Class(CoS,L,S,C)}\) are logical beliefs that represent, respectively, the solution \(S\) that student \(L\) believes will solve case of study \(CoS\), and the classification \(C\), for pedagogical purposes, of the student solution \(S\). \(\text{Conf(CoS,L,S)}\) and \(\text{Cred(CoS,L,S)}\) are probabilistic beliefs that express, respectively, the declared degree of self-confidence that the student had about his own solution (BN), and the degree of credibility that Learner agent infers about the solution \(S\) that was made by the student \(L\) (a related work can be find in (Conati et al 1997)).

All interactions that occur between AMPLIA and their students are modelled as Pedagogical Negotiation (PN) processes (Gluz et al. 2006a). The main goal of a PN process is to set and reinforce a high level of confidence among its participants, related to skills the student has showed about the learning domain.

The PN process is successful only when there is a student’s solution \(\text{Sol(CoS,L,S)}\), which is acceptable for the system \(\text{Class(CoS,L,S,Complete)}\) and that self-confidence \(\text{Conf(CoS,L,S)}\) and credibility \(\text{Cred(CoS,L,S)}\) reach (or surpass) a proper pre-defined threshold level. The Domain agent evaluates \(\text{Class(CoS,L,S,C)}\) and pass this information to Mediator agent. The Learner agent uses the student’s log to infer the values of \(\text{Cred(CoS,L,S)}\), and passes this information to the Mediator agent, which decides what tactics to apply for the continuity of PN process through the influence diagram shown in Figure 5. The Mediator agent use the pedagogical strategies presented in Seixas et al. (2006) work to achieve successful conclusions of PN processes.

So far, we have conducted two sets of experiments with AMPLIA. The first experimental phase comprised seven medical students, eleven resident physicians, and twelve medical professionals from Hospital de Clínicas de Porto Alegre (HCPA), who attended study sessions from May to June 2005. The second set of experiments was conducted in a Workshop at the 44th COBEM (Brazilian Congress in Medical Education), where AMPLIA was used in a workshop for development of diagnostic reasoning abilities. During the experiment 62 workshop participants answered questions evaluating different aspects of potential applications and pedagogical use of AMPLIA.

![Figure 5 Mediator Agent Influence Diagram](image)

Figure 5 Mediator Agent Influence Diagram

Results obtained in these tests have shown that there is a convergence with the observations carried out by teachers who followed students during the process of diagnostic model construction. This means that teachers would use tactics and strategies similar to those selected by AMPLIA, to mediate this process. In short, the student model elaborated by teachers is similar to the model constructed by AMPLIA, and decisions taken by the environment are in accordance with teachers’ pedagogical positions.
6 AOSE METHODS DERIVED FROM ITS PRACTICES

AOSE methods proposed in this Section rely on a top-down model-driven approach for the analysis, design, and development of multiagent applications. In these methods we basically assume that cognitive high-level agent abstractions should be considered since the beginning of the software engineering process, including the requirements engineering phase. Cognitive abstractions such as beliefs, goals, intentions, and social abstractions based on cognitive models, provide the ground where high-level properties of the domain can be intuitively understood and enunciated as application requirements. Formalisms as modal logics provide the way to state these requirements in a precise and non-ambiguous form. High-level agent architectures and models derived from ITS research provide design frameworks for prototype systems that implement these applications.

Methods proposed in this Section will be organized in three sets, each one related to important activities of the Software Engineering (SE) process. The set of Applicability Criteria (AC) will be presented in Subsection 6.1 and should be verified since the beginning of SE process, during the analysis of software requirements. The set of Design Principles (DP) will be presented in Subsection 6.2 and should be considered during the software design. The set of Development Guidelines (DG) will be presented in Subsection 6.3 and should be applied during software construction and testing.

6.1 Applicability Criteria

The main purpose of the applicability criteria is to check if the design principles and development guidelines to be proposed further ahead, really are useful for some application. These criteria should be considered during the process of analysis and elicitation of application requirements. Consequently, they will force the requirements engineer to take into account agent abstractions when analysing the application domain and eliciting application requirements. As a result, if the application really satisfies the criteria being proposed here, then the requirements specification will naturally incorporate agent concepts and abstractions.

The particular methods used for requirement analysis and elicitation is not in concern here. The criteria stated in this section should be regarded as extra conditions to be added to contemporary analysis and elicitation methods of Requirements Engineering.

The first applicability criterion is a fundamental presupposition about the application domain: [AC.1] The application domain contains entities that are better understood as agents and the application should be conceptually understood as a system composed of agents working together.

This criterion states an intuitive and abstract principle about reality, and the way entities are mapped from reality to computer systems. It is not an operational criterion, but a conceptual presupposition that the requirements engineer should assume about the properties of application domains. In essence, it is equivalent to the presupposition behind object-oriented analysis that there are objects in the reality and that such real objects can be mapped to computer objects. In practice, the greatest challenge found to apply this requirement is to understand what an agent is, and what its abstract properties are.

In this work, we will settle this issue defining that an agent is a computational process situated in an environment, which is designed to achieve its purposes in this environment through an autonomous, deliberate and flexible behaviour.

The second criterion enforces the separation of application domain knowledge in knowledge concerning agents and in knowledge about other non-agent entities. This criterion was satisfied by all the ITS multiagent applications described in Section 5. It can be stated as:

(AC.2) Agents beliefs about the domain must be divided in beliefs about non-agent entities and beliefs about agents.

This criterion becomes an operational criterion, once agents are identified according (AC.1). It is also important for the design process because, when combined with (AC.4), it helps the partitioning the knowledge employed by agents, dividing knowledge that is critical do the modelling of agents from other kind of knowledge.

In MCOE system the compliance to this criterion was implicit in the separation of knowledge about bottom of the lake, micro-organisms, plankton, water and similar entities, which was incorporated in reactive agents, from knowledge about students and tutoring activities, incorporated in cognitive agents. In PAT this criterion separated the knowledge related to the model of emotions (the OCC model), incorporated in the Mind component of PAT, from the knowledge related on how to realize affective states, incorporated in PAT’s sensors. In AMPLIA, this was the separation criterion between the knowledge about student’s credibility and confidence, from the knowledge about the effectiveness of diagnostic hypothesis.

In the third criterion we adopt the idea of Newel’s knowledge level, and, following the classic theory behind agent communication research (Cohen and Levesque, 1995; Chaib-Draa and Dignum, 2002), we assume that agents share beliefs with each other, through the use of communicative acts. It requires agents to communicate knowledge (beliefs) with other agents: [AC.3] The communication among agents is symbolic and occurs at the knowledge level.

---

1 Recent development tools for logical programming languages, like JASON (Bordini et al., 2005), or X-BDI (Mora et al, 1998), make possible to transform system specifications into prototypes, at least for certain kinds of applications.

2 For an introductory text that presents the concepts and abstractions related to agent and multiagent systems considered in this work, see the book of Wooldridge (2002).

3 See the works of Nuseibeh & Easterbrook (2000) and Sommerville & Sawyer (1997) for an introduction to Requirements Engineering concepts.

4 This is an adaptation of the definition of agent by Jennings (2001), which explicitly incorporates the notion of design purpose and deliberation.
This criterion is an operational criterion assumed in the analysis and design of all agents of our ITS applications. The communication with non-agent entities usually will take the form of symbolic exchanges too. Only the requirement that they occur at the knowledge level, needing to be belief exchanges, is not mandatory in this case.

Concerning the cognitive approach taken by this work, the fourth criterion is the main criterion to be satisfied by the application agents. This criterion is directly derived from the cognitive student models presented in Section 4. It was satisfied by all ITS applications described in Section 5.

(AC.4) Agents need to explicitly model other agents and these models should be cognitive models.

We regard the BDI cognitive model for agents presented in Section 4 as the basic normative cognitive model for agents identified by (AC.1). We also assume that deliberative aspects of agents are a consequence of BDI cognitive modelling.

MCOE’s cognitive agents, the entire PAT agent, and AMPLIA’s Learner and Mediator agents, offer examples of agents that need to model cognitive properties of other agents to accomplish their goals.

The premise of (AC.4) is that the application will only be successful if agents that implement it can model the behaviour of other agents (artificial or real) in the environment and understand how to interact with these agents. The modelling must be explicit: application agents must have explicit models for the properties of other agents. The form of knowledge about non-agent entities is not of concern here. Due to the agent model regarded in this work, the knowledge that agents will form about non-agent entities will be composed by beliefs. However, it is not necessary to ascribe intentions or desires to this kind of entities. Otherwise, it is better to reconsider these entities as agents.

It is necessary, but not enough to presuppose that agents need to model cognitive properties of other agents. From our perspective, ITS applications should be considered as social systems. Agents will form social relationships with other agents, based on the cognitive properties they know about these other agents:

(AC.5) Social relationships and interactions are based on models of agents. Application’s agents are sincere and co-operative between themselves.

For the fifth criterion we adopted, and generalised the perspective of Castelfranchi & Falconi (1998, 1999) about trustfulness relationships to other social relationships. The idea is that, from some agent’s viewpoint, all interactions with another agent will be based on its internal model about the other agent. In this stage of analysis process, it is not important how this model is built. The agent can construct the model according to previous interactions, or the model can have been incorporated by design, but always there is an internal model that guides the social interaction processes.

The creation process of internal models for other agents could offer a complex design situation. However, there are social principles that simplify this process. Based on the experience with ITS MAS, we will assume that agents that compose the application are by definition, cooperative agents among themselves. The purpose of these agents does not conflict by design. The communication among these agents is always sincere and negotiation is possible as a form of coordination of work, but not of competition.

To evaluate the impact of this criterion on the systems described in Section 5, is necessary to understand that social relationships established in ITS have a pedagogical purpose. For instance, main social relationship between MCOE tutor agent and the student is to aid the student to maintain an adequate level of energy in the environment. The establishment of a successful relationship depends on the cognitive model that this agent has built about the student. Main social relationships to be established by PAT agent with some student are driven by the model about the student’s emotions, which can be inferred by the student’s observable behaviour in the interface of the learning environment. Social relationships of AMPLIA system are based on Pedagogical Negotiation (Gluz et al. 2006a) interaction processes. The main purpose of the Learner and Mediator agents is to model confidence and credibility aspects of the student in order to achieve successful conclusions of these negotiations.

Criteria defined until now provide conceptual and operational tools to analyse the application domain in search of appropriate requirements for agents. These requirements need to be documented:

(AC.6) The requirements of application that will be assigned to agents should be clearly stated in the requirements specification. The specification needs to define what knowledge (belief bases) is required to fulfil these requirements.

An application requirements specification is a set of statements declaring what are the purposes of the application about their users, stakeholders, designers, and programmers. We will add the extra condition that purposes will also need to be specified for the agents that will compose the application. Note that it is not necessary to define, in this phase, exactly which agents will compose the application, only what purposes will be assigned to the agents. The specification also needs to define which knowledge agents will need to achieve these purposes. Optionally it should also show how and where it can be obtained.

6.2 Design Principles

The design process of pedagogical agents or ITS MAS applications should provide satisfactory answers to three basic questions: (a) How will be represented the learning domain of the system? (b) How students will be modelled? (c) How will be evaluated and represented the pedagogical relationships between the system and the students? The design of each ITS MAS presented in Section 5 was aimed to answer these questions.

These questions can be generalised to other application domains. The insight we had was the correspondence between learning domains, which constitute the focus of interest of students in the case of ITS, with application domains in the case of other systems. Students will correspond to
users in other applications. The study topics of some learning domain will correspond to the problems in the application domain, which are of interest to users\(^5\). Following this idea, student models used in ITS will correspond to the knowledge that the system has about external entities that interact with it (including users). Finally, pedagogical relationships in ITS will correspond to social relationships that should be established between the system and its users.

This insight has led us to propose four principles for the design of MAS. These principles combine the Cognitive AI approach to model application domains discussed in Section 1, with ITS practices and models described in Sections 2 and 3, providing a multiagent architecture for applications that satisfy the applicability criteria defined in Subsection 6.1. These principles are the core of our AOSE methods. They are our top-down design-oriented answer to the macroscopic behaviour problem cited in Section 1, about how to engineer the emergent properties of some MAS.

The first design principle will use the proposed correspondence between ITS learning domains with other application domains, showing how these domains should be partitioned:

**(DP.1) Partition of application domain:** the knowledge about the application domain should be divided, at least, in three distinct subdomains:

- The *Problem Solving* (PS) subdomain, which contains knowledge about problems that the application is intended to solve.
- The *Users and Agents Modelling* (UAM) subdomain, which contains knowledge about users and external agents.
- The *Social Mediated Interactions* (SMI) subdomain, which contains knowledge about social interaction mechanisms.

The PS subdomain is the generalization of the knowledge about the learning domain, incorporating knowledge about other kinds of domains, and being focussed on how to represent and solve the user’s problems in these domains. The UAM subdomain generalizes the concept of student model to incorporate knowledge about other entities that interact with the application, with focus in its users and in other intelligent agents that can use the application. Finally, the SMI subdomain generalizes the knowledge about how to mediate pedagogical relationships. This subdomain incorporates knowledge on how to mediate social relationships with application’s users or external agents.

From the experience with ITS applications, it will be assumed that each one of these subdomains should be under the responsibility of a different type of agent. Thus, a specific type of agent will incorporate the knowledge of each one of these subdomains:

**(DP.2) Definition of types of agents:** a specific type of agent should incorporate the knowledge of each subdomain defined in (DP.1):

- PS agents should be created to solve application problems.
- UAM agents should be created to understand, and make cognitive models of users and external agents.
- SMI agents should be created to mediate social interactions between the user, or external agents and the application.

Figure 6 presents a graphical schema, which combines (DP.1) and (DP.2). Note that (DP.2) requires the creation of types of agents, thus each subdomain can have more than one agent of the same type, as can be seen in Figure 6. It is also important to mention that the subdomains do not need to be completely separated. It is possible (and even necessary) that there exist non-null intersection areas between subdomains, composed by knowledge that interrelates concepts in both subdomains.

\[\text{Figure 6 Application Subdomains and Agent Types}\]

Principle (DP.2) was partially supported by the design of our first multiagent ITS, the MCOE system, where the student agent is responsible for the student model (UAM knowledge), but tutor agent is responsible for the learning domain (PS knowledge) and pedagogical relationships (SMI knowledge). On the other hand, PAT as an individual agent, focused only on the aspects of application sub-domains defined in design principle (DP.1), i.e. on aspects of UAM and SMI sub-domains related to identify and promote emotional responses. This fact reflected in the software modelling of the agent, which adopted a centralized architecture. AMPLIA system was our first system to fully adopt both principles, as the Learner agent models the student and is the equivalent of the UAM agent. Domain agent stores domain knowledge and is the equivalent of the PS agent, and the Mediator agent is the SMI agent.

The requirements specification has a list of requirements to be assigned to agents (AC.6). These requirements need to be distributed among agents, considering the skills of each one of the agents defined in (DP.2):

**(DP.3) Assignment of purposes to agents:** the set of applications requirements to be assigned to agents need to be divided into three subsets of purposes in order to be distributed to each type of agent earlier defined in (DP.2).

This principle defines the mapping between application requirements and agent purposes, which are the high-level desires of some agent. It also implies that this mapping should be classified accordingly the subdomains defined in (DP.1).

\(^5\) From this point of view the problems that an ITS should solve is: “how to aid students to learn topics from this domain?".
The application of (DP.3) can be seen in MCOE tutor and student agents, whose stated purposes are, respectively, to aid the student to maintain an adequate energy level in the environment and to represent the student in the environment. Because PAT is modelled as an individual agent, it has only one set of purposes, which are related to UAM and SMI subdomains. The main purpose of PAT is to infer aspects of the student’s affective state in a teaching/learning situation. AMPLE system works with three types of agents: Learner (UAM), Mediator (SMI) and Domain (PS) agents. Following (DP.3), the goal of Learner agent is to represent the student virtually. Domain agent’s goal is to store domain knowledge and to evaluate the knowledge modelled by the student. The Mediator agent’s purpose is to choose appropriate pedagogical strategies for the interaction with the student.

After the domain structuring (AC.1), and the classification (AC.2), and assignment of purposes for agents (AC.3) it is necessary to organize the knowledge of how agents will achieve their purposes. In this work we assume that, for agents to achieve their purposes in a particular domain (environment), they need at least to: (a) be able to judge and decide what are the significant entities of the domain and what are the main properties of these entities, (b) be able to perceive events, execute actions over these entities, and be able to identify main properties of these entities through these interactions, and (c) dominate planning and problem solving skills in respect to these entities. This tripartite division implies in the following principle:

(DP.4) Organization of agent’s belief bases: specific bases of beliefs must be assigned for PS, UAM and SMI subdomains. Each one of these bases should include knowledge about the:

- Significant entities of the subdomain and their main properties.
- Basic identification abilities, and possible actions and perceptions of the agent, in respect of these entities.
- Planning and problem solving skills necessary to achieve desires (goals) related to these entities.

This principle is relatively complex because it implies that all types of agents proposed by (DP.2) will need to have belief bases customised in order to handle the different kind of issues manipulated by each subdomain. The following set of rules show what aspects should be customized (Figure 7 summarizes these rules).

The first rule is a direct adaptation of (DP.4) for PS agents. In this case, the significant entities of the subdomain correspond to the elements of the domain that users are interested in:

(DP.4a) Structure of belief base of PS agents. This belief base should include:

- Main properties of the elements of the domain that are important to users.
- Abilities, possible actions, and perceptions the agent has in relation to these elements.
- Specifications about problems that occur with these elements, how they are detected, and what strategies, tactics, and heuristics should be used to solve them.

The PS agent represents the “classical” part of the application, which is intended to solve problems of the user. It is only a part of the entire MAS. To be a complete application from our viewpoint, it is necessary to add knowledge to understand how users (or other external agents) are proceeding in their tasks. This is the function of UAM agents. The customisation of (DP.4) for UAM subdomain results in the following rule:

(DP.4b) Structure of belief base of UAM agents. This belief base should include:

- Main properties, including beliefs, desires, and intentions that will be used to model users and other agents.
- Perceptions used to detect these properties and actions related to them. Reasoning strategies, tactics, and heuristics necessary to make accurate diagnostics about the current state of the model.
- Definitions on what decisions of the agent will be affected by the models of other agents and what planning and problem solving mechanisms will be affected by these decisions.

The SMI subdomain defines what social relationships should be established between the application and their users (and how this is accomplished). Principle (DP.4) should be adapted to this subdomain as follows:

(DP.4c) Structure of belief base of SMI agents. This belief base should include:

- Main properties of social relationships that should be pursued by the application.
- Knowledge necessary to diagnose the state of social relationships. This assessment should be done regarding the state of the environment, as perceived and understood by PS agents, and regarding the user model or external agent model, as perceived and understood by UAM agents.
- Strategies, tactics, and heuristics to maximise the chance to set intended social relationships. Knowledge of PS and UAM agents can be used if necessary.

![Figure 7 Divisions of Knowledge for Agents](image-url)
Combined with subdomains defined in (DP.1) the principle (DP.4) creates a 3x3 structuring “grid” for agent’s belief bases presented in Figure 7. It is important to note that this grid was not superimposed on ITS applications described in Section 5, but gradually emerged from the design and development of these applications.

The use of (DP.4) in MCOE system can be seen in the modelling example presented in Subsection 5.1, which shows how tutor’s and student agent’s mental states views may be represented by desires and beliefs. PAT agent has a centralized architecture, not divided in PS, UAM and SMI agents. It assumes a limited combined role from UAM and SMI agents. Internal aspects of the organization of knowledge used to infer the affective state (DP.4) can be seen in Subsection 5.2. Each type of agent used in AMPLIA incorporates a complex knowledge base to achieve their goals, which is organized as specified by principle (DP.4). For instance, the knowledge necessary for Mediator agent to evaluate the current state of the PN process is modelled by the Influence Diagram presented in Figure 5. Main properties of this state are represented by \(\text{Class}^{\text{CoS}}(\text{CoS}, \text{S}), \text{Conf}^{\text{CoS}}(\text{CoS}, \text{S}), \text{Cred}^{\text{CoS}}(\text{CoS}, \text{S})\) shared beliefs. Strategies and tactics used to achieve successful conclusions of PN processes are presented in Seixas et al. (2006) work.

Design principles (DP.1) to (DP.4) are our proposed core principles for a multiagent architecture conforming to ITS design practices and architectures. However, to be a fully MAS these agents will need to communicate with each other in order to achieve their purposes. The following three design principles will guide the modelling of this communication.

It is important to note that the next three principles are different from specialized principles (DP.1) to (DP.4). These principles were not abstracted from ITS research. In the present form, they were derived from FIPA agent communication standards (FIPA, 2002), due to our experience with AMPLIA. We consider them (with possible minor extensions) as established principles of good practice in the design of MAS communication that should be supported by any AOSE methodology.

The first of these three principles defines the basic media of communication among agents:

\(\text{(DP.5) Definition of agent communication media: all communication among the system agents must be carried out through an Agent Communication Language (ACL). All knowledge should be encapsulated and transported by a content language.}\)

The main point of (DP.5) is that all the communication among agents be carried out through specific communicative acts, with contents with precise syntax and semantics (this is the reason to use the ACL and content languages).

Not all beliefs of each agent need to be communicated to other agents. In the same way, not all beliefs informed by other agents will necessarily be understood (or be meaningful) for a particular agent. For that end, it is necessary to define the subset of beliefs that will be shared and understood by each agent of the system:

\(\text{(DP.6) Specification of system ontology: the subsets of beliefs shared among agents, users, and external agents should be specified in the ontology to be used by the system.}\)

This ontology should also specify the subset of the knowledge bases specified in principle (DP.4), which will be exchanged between SMI, PS, and UAM agents.

Possible exchanges of communicative acts are managed through interaction protocols. Agents participate in some interaction protocol with the intention to achieve purposes or satisfy desires. In terms of the application architecture design, this implies the following principle:

\(\text{(DP.7) Definition of interaction protocols: all possible interactions that occur among system agents should be specified through an appropriate interaction protocol.}\)

From all ITS systems presented in Section 5, only AMPLIA’s architecture fully adopted principles (DP.5) to (DP.7). All communication among AMPLIA’s agents is implemented using FIPA-ACL communicative acts (DP.5). The knowledge to be shared between AMPLIA’s agents form an ontology (DP.6) composed of several distinct propositions that represent the logical or the probabilistic beliefs that these agents need to communicate one to another. According to (DP.7), the entire negotiation process is modelled by interaction cycles composed of inform and query interaction protocols defined by FIPA.

The support for an agent communication language (DP.5) or interaction protocols (DP.7) is incipient in MCOE system and PAT agent. These principles were implicitly used in the design of these applications, but not enforced by any standard communication, content or ontology language. Instead, the X-BDI language was used in the development of these applications, extended with communication primitives for belief exchange. This can be seen in the modelling examples and use cases presented in Section 5. A particular subset of beliefs is communicated between MCOE’s tutor and student agents, and between PAT’s Mind component and the Body component of the agent.

Design principles (DP.1) to (DP.7) specify the multiagent architecture envisioned to capture ITS design practices. These principles do not detail how each individual agent will be designed. They specify some requirements to be supported by each type of agent, as the overall organization of the knowledge base of agents prescribed by (DP.4), or the communication tasks that must be implemented in each agent to support principles (DP.5) to (DP.7). However, they do not prescribe any individual software architecture for these agents.

The software architecture of many individual agents should comprise the definition on how this agent can understand the knowledge it receives, or on how it can process this knowledge to perform actions to achieve its purposes. Agent architectures tell how to build individual agents and how they will work.

Agent architectures are usually divided into several distinct abstraction layers. InteRRap (Müller et al. 1995) and the architecture of Glaser and Morignot (1997) are representative examples of layered agent architectures.
We also regard useful to assume that the architecture of agents specified by (DP.2) should be divided in abstraction layers. The layering principle used here is based on the vertical layers of Dillenbourg and Self (1992) abstract framework (see Figure 1 upper portion). It assumes that each agent has layers to: (a) register perceptions and execute actions (the concrete layer), (b) recognize the current state of the world and plan course of actions for a desired state of affairs (behaviour layer), and (c) assume intentions and take appropriate decisions to achieve a desire (conceptual layer).

There is no need to specify a social abstraction layer for agents defined by (DP.2), like the CPL layer of InteRRap or the social layer of Glaser and Morignot architecture. In our proposed multiagent architecture, the social issues are distributed on the purposes of each one of these types of agents. The states of these relationships depend on knowledge and information obtained by UAM and PS agents. The establishment, reinforcement and termination of these relationships are responsibilities of SMI agents.

The following principles define the layered agent architecture considered in our applications:

**Definition of the architecture of individual agents:**
the internal architecture of each type of agent defined in (DP.2) should be divided in the following three layers:

- **Judgment and Decision Layer (JDL):** the higher layer, which incorporates the purposes of the agent. Main decision processes happen here, including judging and diagnostic activities related to the agent’s own cognitive status and to what happens in the external world. Decisions taken in this layer become new intentions.

- **Assessment and Planning Layer (APL):** the intermediate layer responsible for planning and performing decisions taken in JDL, and for planning and running information gathering actions needed for the higher layer.

- **Perception and Action Layer (PAL):** the PAL is the low-level layer responsible for all direct interaction acts of the agent with the external world.

There is a correlation between these layers and the organization of belief bases specified by principle (DP.4). The knowledge about basic perceptions and actions will reside in PAL layer. The identification skills of the agent and mostly of its planning and problem solving abilities will reside in APL layer. JDL layer will combine the purposes of the agent with the knowledge about the significant entities on the domain (and with some planning and problem solving knowledge) to make appropriate judgements, and decisions about its intentions in respect to these entities.

The architectural design of each individual agent of AMPLIA system follows (DP.8). The project of each agent is divided into three distinct layers: a) the decision (JDL) layer models the high-level decision processes of the agent; b) the operational (APL) layer plans how decisions taken in decision layer are mapped into real actions and operations in the environment; c) the interaction (PAL) layer is responsible for sending/receiving messages among agents and also for the interaction with users.

The Mind component of PAT agent can be considered the top layer of the agent, combining functions of JDL and APL layers defined in (DP.8). Body component, on its turn, is the agent’s PAL layer.

The design of MCOE agents was not explicitly organized in layers. However, the division of the architecture in the Inference and Select stages (see Figure 2) provided one of the first insights on how to use layering for this kind of agent.

### 6.3 Development Guidelines

The following set of developed guidelines were derived from the development process of the ITS applications described in Section 5 (and also from other applications like the Eletrotutor multiagent system (Silveira & Viccari, 1999)). These guidelines provided useful ideas about how to transform application architectures designed according to the principles seen in Subsection 6.2 in operational systems.

The first guideline was very valuable for the development of all our ITS applications. It can be stated as:

**Program with agent abstractions.**

One of the most difficult tasks in the translation process from agent based software architecture specification to concrete implementations is to find out programming languages with appropriate programming abstractions to support agent-derived concepts.

In the case of MCOE system and PAT agent, it was possible to develop the reasoning modules of their cognitive agents directly in X-BDI interpreter for BDI logic. Indeed, one of main reasons for the creation of X-BDI interpreter was to aid in the development of these agents. This procedure allowed a fast and flexible development process, with easy testing and adjustment of their knowledge base.

The main problem of applying this guideline is that the definition on what are the appropriate agent-derived programming abstractions is still an open research issue. In this work we assume that this problem is partially circumvented assuming that a good agent programming language should support the set of agent abstractions already used in the analysis and design phases: (a) the use of BDI mental models for agents; (b) the support of agent communication at the knowledge level through the use of communicative acts; and, (c) the use of ontologies and interaction protocols to provide meaning and guide interactions among agents.

These abstractions were successfully included in logical programming languages like AgentSpeak(L) and X-BDI. Consequently, there is a good chance of the specification to be correctly represented in the implementation. This effectively happened in the development process of MCOE system and PAT agent.

However, it is not always possible to find an appropriate agent-programming environment that supports the right abstractions. For instance, the development process of AMPLIA’s agents did not follow this guideline because there is no integrated programming language that supports BDI and bayesian network concepts yet. Not having an appropriate language with the abstractions used in the analysis and design of the system was the main reason why the development took three years to be completed.
The next guideline was fundamental for the development process of AMPLIA system:

(DG.2) Develop the support for agent communication first.

The first tasks that should be implemented for each agent are the operations related to agent communication support. This includes the operations used to send, receive and handle communicative acts, manipulate ontologies, and follow interaction protocols. This implementation should be as flexible and extensible as possible because new communicative acts, ontology elements, and interaction protocols will be necessary with the development of the system. The use of an agent communication platform is critical to reduce the complexity and cost of the development.

The experience with the development of MCOE, PAT, and AMPLIA systems has shown us that the development process of agents normally depends on the incremental creation of several prototypes, which are continually rewritten, until the final prototype achieves the required competence. This is stated in the following guideline:

(DG.3) Use extensive prototyping and refactoring; develop agents incrementally.

To implement agent prototypes with a minimum know-how is necessary to test communication interfaces defined in (DP.5), (DP.6), and (DP.7). The implementation of these prototypes should be the most flexible and documented as possible. If possible, they should be programmed directly with agent abstractions (DG.1). The agent prototypes will be used as the basis for new prototyping cycles.

These prototypes should always be developed incrementally. There is always the possibility to add a new purpose to the agent or to include a new problem solving or planning ability to it. Refactoring and reengineering processes should be constantly applied to the prototype’s code, allowing these extensions to be made in an organized way.

When there are sufficient resources, it is possible to accelerate the overall development cycle of MAS, doing some tasks in parallel:

(DG.4) Develop different types of agents in parallel.

Each type of agent defined in (DP.2) works with distinct parts of the environment. These agents can be developed in parallel if there are resources to do so. Usually these different types of agents must interact in order to achieve results, but if the interface of communication of each agent is clearly defined as required by design principles (DP.6) and (DP.7), then it is possible to do a great part of the implementation job of each agent in parallel. Then the prototypes of individual agents can be easily tested through simulated communication calls.

The effect of this guideline appeared mainly on AMPLIA system. The three kinds of agents of AMPLIA were partially developed in parallel, and this measure was determinant to the successful conclusion of this system, reducing the problems caused by lack of compliance to (DG.1).

In all the ITS applications described in Section 5 the most time consuming development task was to incorporate the problem solving knowledge to the agents (the know-how of these agents). This is expressed in the next guideline:

(DG.5) Take time to add know-how to agents.

Design principle (DP.4) identifies knowledge the agents must have to achieve their purposes. The implementation of individual agents starts from this specification, but requires the incorporation of the specific knowledge on how to achieve particular purposes (the know-how of the agent). This process corresponds to the programming tasks in other kinds of systems. However, due to the complexity commonly associated to diagnostic, decision and planning problems implied by (DP.4), usually it is not possible to use traditional programming techniques to solve these problems.

The solution we use to solve this problem was to apply AI-derived problem solving and planning techniques over the domains of knowledge defined in (DP.4). The success of these techniques depends strongly on the quality of the heuristic knowledge used to prune the huge search spaces associated to these domains. The difficulty in the development of agents is the determination and implementation of successful problem solving tactics and strategies based on these heuristics. Great part of development efforts and resources should be allocated to these tasks.

The following guideline derived from the evolutionary approach used to develop the AMPLIA system.

(DG.6) Use iterative and incremental cycles of development (evolutionary development).

This guideline is the complement of (DG.3) for the case of the entire MAS. The proposed cycle of development for MAS is iterative and incremental, based on the spiral model for software development defined by Boehm (1988). Following (DG.2) and (DG.3), agent prototypes should be developed first to test communication interfaces of the system, then to build proof-of-concept systems to test versions of individual agents’ know-how in simulated situations. The process continues with the incremental evolution of the individual agent’s knowledge and with the refinement of system ontology, and interaction protocols. The evolutionary development process is guided by simulated or real tests, until the system achieves a satisfactory performance level. Problems detected in these tests can imply the rewriting of software architecture specs or system’s requirements. This is not necessarily wrong, but a matter-of-fact situation derived from the complex environments these systems will work with.

In our approach of ITS applications development, tests should be considered under the perspective of the scientific method. The ITS application represents a concrete model about some aspects of cognition and reality, and experiments should test the predictions of this model:

(DG.7) Plan tests as scientific experiments.

Once there are no formal analytical models of cognition to be used as comparative models in simulated experiments, simulated experiments have a limited value. They can be useful to check agent’s communication interfaces or particular problem solving abilities, but they cannot verify if the model is correct. Final tests should be made through concrete experiments with real users (subjects), preferentially with control groups that try to solve the same problem without the system’s help. These experiments should be con-
ducted in order to check if critical situations previewed in the model are handled by the system. Only when the results of these experiments are positive, and the control group cannot achieve the same results, it is possible to assume that the model and the system that implements it are valuable approximations of cognition phenomena.

7 CONCLUSIONS

This paper presents discussions about software engineering methods for agents and multiagent systems that have anthropomorphic characteristics. The basis of these discussions is our experience in the design and development of ITS using agents as a software engineering technology. This ITS applications were tested several times with students, and some of them are currently in use. Another reason to present this proposal is that there are a large community that is using agents to specify, design, implement, and test ITS, providing an extra motivation to share our experience.

Based on this experience in the analysis, design, specification, implementation, and test of these systems, we propose a new set of AOSE methods. We are convinced that these methods are useful for ITS systems, although we expect that they can be generalised to other kinds of systems and applications. Indeed, a careful requirement analysis should be conducted first in order to decide whether the proposed methods are adequate to the application. On the other hand, if the application can be specified according to these requirements, then there are design principles and development guidelines that can be successfully applied to model the software architecture of the application. In particular, the choice of logical formalisms for these specifications has been deeply influenced by the restriction that these formalisms should be computationally effective. This allowed the minimisation of the gap between specification, programming, and agent execution.

The set of AOSE methods proposed here were applied in the development of several ITS systems modelled through agents. These methods worked well with a heterogeneous reactive and cognitive multiagent system; the MCOE system, and were suitable for the design and development of the single agent PAT, which work with emotive and affective domains. They also were successfully applied in the AMPLIA multiagent system, which operates in a distributed way over probabilistic and logic domains.

However, we understand that these methods need to be submitted to further testing. We agree that their application to other domains unrelated to ITS, is only on the initial stages. Indeed, an important ongoing goal of our work is to verify the applicability of these methods in new application domains. For instance, presently we are applying these methods to the analysis and design of a project management application, intended to aid the human resources allocation process in software development projects, based on the competences and skills of programmers.

During the development of our systems, in particular during the AMPLIA development, we missed a tool that helped in the specification, programming and testing process. A good programming language or tool that provides integrated abstractions mechanisms to represent BDI models, probabilistic knowledge (BN and dynamic BN) is still not available, but would have been very useful in AMPLIA development.

We continue to work in this research line of integration of probabilistic models with BDI models, studying Probabilistic Modal Logics that can handle subjective (Bayesian) probabilities and BDI models (Gluz et al., 2006b), and investigating an ontology-based approach to promote the interoperability among agents that represent their knowledge through Bayesian networks (Santos et al., 2007).

Another problem involving agent development is that today there is no mainstream programming language that supports a common set of agent programming abstractions, acknowledged by all research community. This is due to several reasons, but the result is that the definition of what the appropriate agent-derived programming abstractions are is still a complex and open research issue. In our opinion, programming languages derived from modal logics such as AgentSpeak(L) and X-BDI, or tools like JASON, provide appropriate agent programming abstractions and are viable implementation tools. These languages and tools can lack some important abstractions, but they are in the right direction. In some aspects, it is a situation that is approaching the object-oriented programming panorama in the beginning of the 1980’s, when SmallTalk was a language very difficult to program, but that incorporated the basic concepts of object orientation.

Summing up, a few mainstream topics in multiagent research are: methodologies for robust software development; standards and communication infrastructure for open multiagent systems; confidence and reputation assessment; reasoning technologies for open environments; mental states architectures; Game Theory and partially observable Markov decision processes (POMDP). The focus of this paper clearly fits the first item: methodologies for robust software development, but it also influences the other topics.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the Brazilian agencies CAPES, CNPq and FAPERGS for the partial support to the several research projects involved in this work.

REFERENCES

An ITS view on AOSE


