5

The Agent Oriented Multi Flow Graphs Specification Model

I. D. Zaharakis[1,2]
[1]*Department of Informatics Applications in Management and Economy, Technological Educational Institute of Patras,*
[2]*Research Academic Computer Technology Institute, Patras, Greece*

1. Introduction

The term agent is widely used in many areas with diverse meaning. In this work, agents have been used as they have been defined in the context of AI. In this context, many researchers use notions to describe an agent that are normally applied to humans, such as knowledge, beliefs, desires, intentions, capabilities, choices, commitments or obligation, (Shoham, 1993; Cohen & Levesque, 1990; Rao & Georgeff, 1991a) or even emotions (Bates, 1994). Wooldridge in (Wooldridge, 1999) uses a broader definition and argues that an agent denotes a hardware or software-based system that exhibits properties such as autonomy, social ability, reactivity, and pro-activeness. Several other agent properties that have been suggested include mobility, veracity, benevolence and rationality.

This work shares the need of the above properties and in order to express them an agent is considered as an *intentional system* based on the description of the philosopher Daniel Dennett. He proposed the *intentional stance* from which systems are ascribed mental qualities. Using this term, he describes entities "whose behaviour can be predicted by the method of attributing belief, desires and rational acumen" (Dennett, 1987). The intentional stance deals mainly with the behaviour of the described entities and helps to avoid paradoxes and clashes of intuition (Russell & Norvig, 2003). Its main disadvantage is that it cannot distinguish among implementations since any given behaviour can be implemented in many different ways and does not imply anything about the system's internal workings or representation. However, from an engineering point of view, it allows the better understanding and consequently the description of a complex system and for this reason is adopted by this work. Concluding that an agent is a system that is most conveniently described by the intentional stance, Wooldridge and Jennings (Wooldridge & Jennings, 1995) consider that the appropriate attitudes for representing agents are classified in *information attitudes* (belief and knowledge) which are related to the information that an agent has about the world it occupies, and *pro-attitudes* (desire, intention, obligation, commitment, choice etc.), which in some way guide the agents' actions.

More than one agent, who cooperate or deliberately act together in order to accomplish a task (usually common) constitute a multi agent system. Multi agent systems are inherently complex since "concurrency, problem domain uncertainty and non-determinism in

execution together conspire to make it difficult to understand the activity within a distributed intelligent system" (Gasser et al., 1987). In order to solve the problem of complexity, there is a need for practical and helpful tools, which benefit a better understanding and managing of a multi agent system. Furthermore, there is a need of suitable and well-defined formalisms, which use appropriate attitudes for representing agents and can express the specific features of a multi agent system such as autonomy, concurrency, social ability and cooperation. Despite the development of several formalisms for this purpose, there are not many tools to embody a formalism and to turn it in a real system. The main reason for the gap between theories and real agent systems is due the abstract notions used and to the many unrealistic assumptions made about the agents' capabilities (Wooldridge, 1996). This work focuses on this issue and proposes the AOMFG model (*Agent Oriented Multi Flow Graphs*), which embodies and combines characteristics of agents' theories, Petri Nets and Cognitive Engineering.

Although formal specifications and methods have been the object of study since the late 1960s, they have not been widely used in software development nor have they been accepted as both worthwhile and practicable, even in mainstream computer science (Wooldridge, 1996). Formal specifications are expressed in a language whose vocabulary, syntax and semantics are formally defined. There are a number of reasons why one should use formal methods for system specification. The most significant among them are that they provide insights into and an understanding of the software requirements and the software design, they can be analyzed by mathematical methods and their consistency and their completeness can be proved, and finally they may be automatically processed.

In this direction and focusing on multi agent systems, some related attempts are reviewed. In addition, some methods describing human action and human computer interaction are examined. This is because several characteristics of human behaviour are ascribed to agents in order to simplify the reasoning about them. According to the agent definition that has been given, it seems that there is a need for a combination of the above methods in order to capture such mental notions, extend without losing specification expressiveness and to maintain the model simplicity to the greatest possible.

2.1 Structure of the paper

In the following sections, AOMFG a graphical formal model for agent specification is introduced. In the next section, several formalisms that have been used for the reasoning about multi agent systems as well as some formalism that are relevant with the scope of this work, are reviewed. Then, a description and a formal definition of the model are given in detail. Furthermore, the model execution is described as well as the model decomposition techniques. Finally, it is shown how an AOMFG can be transformed to a Petri Net. In the last section, the validation of the model is presented with the use of four examples: Shoham's algorithm, FIPA communication protocols, BDI related properties, and a multi agent tutoring system specification. The work concludes with a summary of the work and a description of model application.

3. Background

This section presents several attempts for reasoning about multi agent systems and other attempts in related subjects, which influenced the development of AOMFG model.

3.1 Models for multi agent systems

The most widely used formalism among those that have been developed for reasoning about multi agent systems is intentional logics, which are formalisms developed by researchers in AI and philosophy to describe systems with beliefs, goals, intentions and so on (Wooldridge, 1996). According to formalisms based on intentional logics, mental notions characterize an agent and reasoning about the system is carried out by attributing these notions (Cohen & Levesque, 1990; Hintikka, 1962; Konolige, 1986; Shoham, 1993). The mental notions correspond to a set of possible worlds with an accessibility relation between those worlds. However, because of the possible worlds interpretation, the well-known "logical omniscience" and "side effects" problems have emerged. These mean respectively, that an agent is a perfect reasoner and that it desires the logical consequences of its actions. Many attempts tried to settle these matters by grounding, weaken or even completely reject the possible worlds semantics. Although these efforts show some encouraging results, they are not clear on how attitudes such as desires or intentions can be expressed.

One of the most influential works in agent theory is due to Cohen and Levesque (Cohen & Levesque, 1990), who consider intention as a central notion and as a necessary characteristic for agent's reasoning and actions. Following this point of view, Rao and Georgeff developed a number of formalisms and decision procedures for multi agent systems based on the notion of beliefs, desires and intentions, which are known as the BDI formalism (Rao & Georgeff, 1991a; Rao & Georgeff, 1995). With respect of the intentional stance and in the spirit of speech act theory (Searle, 1969), Shoham proposed the agent oriented programming, as "… a new programming paradigm … [which] promotes a societal view of computation…" (Shoham, 1993). Using choice as a primitive notion and an explicit reference to time in his formalism, he defines a simple programming language and its interpreter, showing in this way, how the theory comes to practice.

A more radical approach that rejects the possible worlds semantics altogether, is Konolige's deduction model of beliefs (Konolige, 1986), which models resource-bounded reasoners (agents as any real system have no unlimited resources) and allows logically incomplete reasoning via logically incomplete deduction rules. Later Wooldridge (Wooldridge, 1992), based on Konolige's model, developed a general model for the specification of realistic (resource-bounded) multi agent systems by using temporal logics. Temporal logics are used for reasoning about multi agent systems since the latter are reactive, concurrent and typically non-terminating. Although temporal logics seem to be powerful for describing past actions and for predicting the future actions of a reactive system according to changes in its environment, it seems to be a need of a combination with other notions such as those of intentional logics in order to become practical. The same holds for the process algebras, too. Process algebras are typically used for reasoning about concurrency in systems. In the case of multi agent systems, mental notions and conceptual properties must be described which are difficult to express using only methods of process algebras.

3.2 Models for human behaviour

Since many human characteristics are ascribed on agents, this section reviews the most important attempts in formal user representation (for details see (Kameas, 1995)) and, determines which of these methods and how may be applied in reasoning about multi agent systems. All these models approach users through Cognitive Engineering (Norman, 1987). In particular, users are considered to interact with computers in order to accomplish tasks that

may contain subtasks. From this point of view, two model categories have been proposed: task analysis, which represents the task decomposition and, cognitive task modelling, which represents the tasks through the computer functionality and the interrelation between the physical and mental actions that are required for the task accomplishment.
According to task analysis (Johnson & Nicolosi, 1990), the user's actions are hierarchically decomposed into subtasks. There are abstract relations between the specifications of the accomplished tasks and their corresponding structures and a good understanding of these relations is required. The analysis techniques use natural language, are ambiguous and there are no specifications tools.
Cognitive task modelling regards human-computer interaction as an iterative procedure for the approximation of a goal through a strategy (Miller et al., 1960; Newel & Simon, 1972), and embodies two architectures. The Model Human Information Processor (Card et al., 1983) includes sensible, cognitive and applied processors together with a representation of the long and short-term memories. The Problem Space Model (Newel & Simon, 1972) treats the logical behaviour in a problem space, a set of situations representing the parts of problem solution and a set of actions provided to the user. According to this approach, a user has a goal, which represents the final state of the system. This goal is iteratively decomposed in subgoals that constitute action plans. The decomposition stops when actions take place for the achievement of the goals. Depending on the actions or the action plans, the cognitive models are divided into models of hierarchical representation of user tasks and goals, linguistic and grammar models and, models of device level. The latter two categories have only marginal interest for this work, so only the first one will be studied further.
A widely known representative of this category is GOMS model (Card et al., 1983), where the users have goals representing their intention for the achievement of a task, operators denoting the primitive actions, methods that are a sequence of actions and, selection rules that determine the choice of a method. Since a goal is decomposed into subgoals, a goal stack containing the intended goals is maintained and represents the users' long-term memory. Another approach is the model of Goal Structure Graphs (Kieras & Polson, 1985) as an application of Cognitive Complexity Theory (CCT). The specification with Goal Structure Graphs is a graphical representation of the system functionality as well as the reasoning of a user. In such a system the user's knowledge is represented as a set of production rules. The rules have a structure "if <condition> then <action>". The model provides mechanisms for the elaboration of the rules that produce a sequence of cognitive actions of the user.

3.3 Petri Nets based models

All the above models require a good understanding of their methods as well as an adequately deep knowledge of their theory. In many cases, it is difficult and infeasible to force software engineers to study specific mathematical models (such as temporal logic or process algebra) or even cognitive models (such as human psychology or user behaviour). On the contrary, it would be desirable to hide the complexity and the strict formalism of the model and to offer a friendlier tool that reflects the expressiveness of the model. In this way "… practitioners can learn from theoreticians how to make their models methodical, and theoreticians can learn from practitioners how to make their models more realistic" (Murata, 1989). Graphical tools with theoretical background satisfy some of these requirements and the best delegates are Petri Nets, which can specify systems that are concurrent/parallel, asynchronous, distributed, non-deterministic and/or stochastic. They have been used in a

wide area of applications that include communication protocols, distributed systems, concurrent and parallel systems, formal languages, logic programs, human factors, and decision systems.

A Petri Net is a directed, bipartite graph and it consists of two kinds of nodes, called places and transitions. Directed arcs reside between the places and the transitions, which are labelled with their weights. The places maintain tokens, which are produced or consumed by transitions. The tokens are moved and distributed through the arcs. Every transition has a set of input and output places. Typically, a transition represents an event, input places represent preconditions and output places represent post-conditions. However, the interpretation of transitions and places is depending on the modelling concept used. For example, a transition may mean a clause in logic and input and output places may mean conditions and conclusions respectively, or a transition may mean a task and input and output places may mean resources needed and released respectively. A marking is an assignment of tokens to each place. A Petri Net has an initial marking (initial state) and this marking is changed according to a simple firing rule:

- A transition t is said to be enabled if each input place p of t is marked with at least $w(p, t)$ tokens, where $w(p, t)$ is the weight of the arc from p to t
- An enabled transition may or may not fire
- A firing of an enabled transition t removes $w(p, t)$ tokens from each input place p of t, and adds $w(p, t)$ tokens to each output place p of t.

By modelling a system with a Petri Net, one can study behavioural properties, which include reachability, boundedness, liveness, reversibility and home state, coverability, persistence, synchronic distance, and fairness, or structural properties, which include structural liveness, controllability, structural boundedness, conservativeness, repetitiveness, and consistency (Murata, 1989). Powerful analysis methods help the study of those properties and may be classified into the coverability tree method, the incidence matrix and state equation approach, and reduction or decomposition techniques. However, the classical Petri Nets have a series of restrictions, the major among them being the need of large nets in order to describe systems of medium complexity (Murata, 1989). In addition, the lack of formal treatment or of clear identification of individuals and the implicit structuring mechanisms used, made inevitable the introduction of new approaches, which are known as High-Level Petri Nets (HPN) (for an extensive review refer to (Gerogiannis et al., 1998)), and extend the classical Petri Nets from several perspectives.

Among all the HPN proposed, those of interest to this work are Predicate/Transition nets (PrTN) and Coloured Petri Nets (CPN). PrTN (Genrich & Lautenbach, 1981) are suitable for modelling logic programs since places play the role of predicates, transitions may associate logical formulas and tokens are predicate extensions. In addition, a PrTN may contain free variables, which are substituted according to the firing sequence. A variation of PrTN is the Predicate/Event Nets (PrEN) (Schmidt, 1991), which adopt methods of process algebra and are used to represent parallelism and concurrency. CPN (Jensen, 1981) associate colours with the tokens, the places, or the transitions. Typically, a colour is a data type assigned to the elements, which are handled by the model. In order to overcome some problems concerning the size of the CPN, an extension of the structuring mechanism has been proposed. This extension is the Hierarchical CPN (HCPN) (Jensen, 1997) consisting of interrelated subnets, called pages, and represent a substitution of transitions. PrTN and CPN are equivalent on computational power although their underlying formalism is

different since the PrTN are based on an algebraic notation while the CPN resemble the procedural programming languages.
Based on a hierarchically high level Petri net, Interactive Multi Flow Graphs (IMFG) (Kameas, 1995) embody characteristics of GOMS and CCT and in such a way combine into a Petri net-based formalism the more important features of state transition models with elements of cognitive models of user behaviour. IMFG is a process-based state transition model specially designed for the specification of interactive dialogue and interactive applications. It provides powerful analysis techniques, task decomposition formalism and event handling mechanisms. Along these lines, another modelling technique concerning the plan decomposition has been proposed (Kinny et al., 1996), however it uses different perspectives as it extends existing object oriented models and the decomposition is represented by state charts rather than Petri Nets. This technique models agents, which are based on BDI architecture, and employs (in some augmented way) object-oriented dynamic models as agents' plans, which are directly executable descriptions of agents' behaviour.

4. AOMFG model

Agent Oriented Multi Flow Graphs (AOMFG) model is a graphical model for the specification and design of multi agent systems. Its main purpose is to describe how the agents reason and act together in a dynamic environment for the accomplishment of common tasks. In essence, it is a high level Petri Net, which encapsulates notions of multi modal logic and reflects the modelling technique of agent systems based upon the BDI architecture. It constitutes a modification and an extension of IMFG model (Kameas, 1995) both in its philosophy and syntax. Although IMFG is a formal model that has been originally proposed for interactive dialogue description or for the specification of interactive applications (Kameas & Pintelas, 1997; Zaharakis et al., 1995), it is modified and extended for reasoning about multi agent systems. Its well-defined formalism for task decomposition and its event handling mechanisms are adjusted for the description of an agent's goal decomposition and for plan representation mechanisms as well as for the communication of messages between agents. The basic components of AOMFG reflect notions generally used by humans such as believes, desires and intentions that help in analysis and better understanding of complex computational systems. Indeed, the specification and design of those systems would be very difficult or even impossible without the use of such notions (Shoham, 1993). However, AOMFG is not intended to describe the human reasoning and behaviour or human social systems, so phenomena of human belief, communication and action are not the objects of this study.
The BDI architecture constitutes the theoretical background of the model and according to this, an agent in a multi agent system is viewed as having the three mental attitudes of belief, desire and intention (BDI). Beliefs are the information the agent has about the world it acts upon and the information it sends to the world. Desires are the tasks or the objectives that the agent is allocated or has to accomplish. Desires are usually referred as goals in the literature, although sometimes there is a distinction between these two notions, and goals have the meaning of the desires that are being pursued. However, this work does not distinguish them and they have the same meaning except when otherwise declared. Intentions are the agent's commitments to a desire. In addition, every agent contains a plan library. Plans are the possible ways that an agent can bring about an intention. In general, a plan is a partial commitment on how to achieve a desire. AOMFG explicitly represents the above notions in its structure.

4.1 General model description

The actions that an agent can execute, the alternative ways an action can be achieved, the perception of the environment and the communication with the environment, are all considered as a set of AOMFG. Every graph consists of two basic active entities, the *actors* (Fig. 1) and *links* (Fig. 2), corresponding to the transitions and the places of the Petri Nets respectively. These entities produce and consume *tokens*, which are the passive entities of the model and move along the connecting *edges* between actors and links.

Tokens

Tokens are transferred on the graph edges and represent data types, which are handled by the links and actors. Every token has an attached data type called *token colour*. The token colour may be a simple Boolean variable or an integer or even a complex structure such as a record. The token colour resembles the variables of programming languages and the value of token colour resembles the value assignment on the variables. In AOMFG, particularly, which is designed to express elements of multi modal logic, the tokens correspond to predicates or to the modal operators used. In graphical representation tokens are represented as black dots.

Actors

Actors represent the system processes and produce, consume or otherwise modify the tokens. Fig. 3 shows the internal structure of an AOMFG actor. Every actor has a *name*, which describes the actor. The interface part of an actor consists of a set of *input links* that provide the actor with tokens, and a set of *output links* that the actor supplies with tokens. In addition, every actor contains a set of firing *rules*, which determine the situations for the execution of an actor as well as the results of such an execution. Every rule has a left part that describes the pre-conditions for the rule firing and a right part that describes the post-conditions of the rule firing. The rules constitute the actor's behavioural part. The operations, which are carried out by the actor, are described with an *operation function* (it makes up the actor's functional part) and the binding of variables is done by a *binding function*. The *type* defines the meaning of the actor and consequently its allowed decomposition schemes. An actor may be:

- **action actor:** it represents a primitive action that an agent can execute and cannot be further decomposed. A primitive action may be **private** or **foreign**. A private action is executed by the agent himself. A foreign action is caused by the agent but its content is requested to be executed by another agent
- **context actor:** it represents a set of actions that lead to the achievement of a higher goal. A context actor is decomposed in sub-goals iteratively until it is refined into a set of action actors
- **library actor:** it represents the way a higher goal is decomposed and is executed in sub-goals. The goal decomposition is achieved with the directed distribution of the tokens. The library actors have inherent in their behaviour the laws of this distribution. The set of available library actors is predefined and the model determines their operation. Library actors play the role and correspond to the language structures of sequence, condition and iteration of high-level programming languages. Furthermore, they can be used to express and support non-determinism and concurrency issues. Depending on the way they manage and control the tokens, they are divided in:
  - **sequence library actor**, where the actors participating in the decomposition must be executed sequentially

  - **non deterministic choice library actor,** where (depending on the firing rule, which is described in the sequel) some actors are chosen and executed
  - **concurrency library actor**, where the actors participating in the decomposition are executed concurrently
  - **condition library actor,** where some actors are executed according to a contained firing condition
  - **iteration library actor,** where the actors participating in the decomposition are executed repeatedly until a termination condition is satisfied
- **group actor:** it has no computational meaning but is used for model clarity. Essentially, group actors are substitution transitions and work in a similar way as the *pages* in Coloured Petri Nets (Jensen, 1997).

As AOMFG is designed to specify concurrent tasks, it is usually necessary to reference time. This reference concerns how concurrent actions occur, when an action takes place or how long an action lasts. In addition, being a specification model, it has to be independent from the target platform where the specified application will be executed. Considering the above, AOMFG presumes time as a regular interval, which is not constant, but it is determined by the target platform (this approach has been originally proposed by (Shoham, 1993)). An executed action by an agent may need more than one regular interval. In addition, the amount of those regular intervals either is not known initially or it cannot be foreseen. However, in the current model version there is no explicit reference to time in actors; instead this can be described as a fact in actor's input links.

Fig. 1. The different actors provided by AOMFG

Links

Links represent the state that precedes or follows a process execution and they store, check and handle the tokens. Fig. 3 shows the internal structure of an AOMFG link. A link has a *name*, which describes the link, a set of *producing actors* that provide the link with tokens, and a set of *consuming actors* that the link supplies with tokens. The way a link is used is significant as it determines the role of the link inside the model and constitutes its behavioural part. In addition, a link can do several *operations* on a range of accepted coloured tokens. This establishes the link's computational part. Links correspond to the places of the Petri Nets and in AOMFG model represent notions of the BDI logic as well as some kinds of events. A link may be of one of the following types:

- **event link:** it reflects the caused events during the execution of the agent. Since, multi agent environments are populated by several agents, perhaps together with human participants, three kinds of events should be distinguished: those caused internally by an agent which concern exclusively himself, those concerning interaction between agents and those caused by the user of the system which are directed to the agents. Consequently, the event links are divided into
  - **system event links,** which are caused by the agent itself and represent the changes in the state of the system,

  - **system communication event links,** which are caused by the agent and are directed to other agent(s) in the environment in favour of a request for an action or a notification of an information (e.g. inform message),
  - **user event links,** which are caused by the user of the application. The model considers the user as an agent that generates events; thus its behaviour need not be further elaborated. The user event link is used for distinguishing between these events and the events of the system. This grading differentiates the agent's response towards the user to the interaction and communication between the agents (communication protocols etc.)
- **desire link:** it reflects the broader context where the several processes and the events of the system or the user take place. In essence, it represents the decomposition hierarchy where the corresponding actor belongs. The desire link when it is an input link of an actor, represents the starting point of an agent's goal, and when it is an output link of an actor, represents the ending point of an agent's goal. If an input desire link contains tokens, then it indicates the adopted intention of an agent to bring about the desire, and if an output desire link contains tokens then it indicates the drop of intention
- **belief link:** it is related to the information that an agent has about the environment it occupies and represents the information, which the agent handles or sends to the environment. In particular, the information, which must be sent to another agent, constitutes a subset of belief links and is represented by the **communication belief link**. The belief links can also be used as conditions that precede or follow an action of the system.

Fig. 2. The links provided by AOMFG

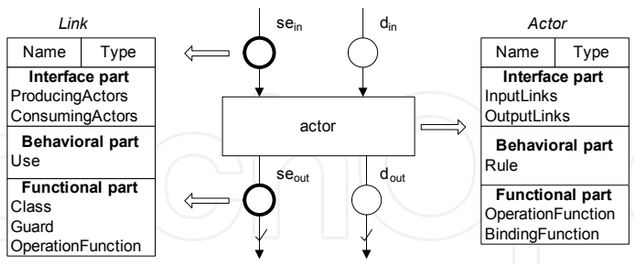


Fig. 3. The internal structure of actors and links

4.2 Formal definition of AOMFG

An AOMFG is a directed, bipartite graph combining text and graphics. It is described by the tuple AOMFG=(A, L, E, T), satisfying the following requirements:

- A is a non-empty set of actors
- L is a non-empty set of links
- E is a non-empty set of edges connecting the links and actors, where $E \subseteq (A \times L) \cup (L \times A)$
- T is a non-empty set of coloured tokens

- $A \cap L = \varnothing$ and $A \cup L \neq \varnothing$

The set of actors A is a non empty set such that

$A = ActionActor \cup ContextActor \cup LibraryActor \cup GroupActor$, where

$ActionActor = PrivateAction \cup ForeignAction$ and

$LibraryActor = SequenceLibraryActor \cup ConcurrencyLibraryActor \cup ConditionLibraryActor \cup NonDeterministicChoiceLibraryActor \cup IterationLibraryActor$.

In addition, the intersection between the members of A is the empty set, e.g.

$ActionActor \cap ContextActor = \varnothing$ etc.

Every actor $a \in A$ is a tuple

$a = (Name, InputLinks, OutputLinks, Rule, OperationFunction, BindingFunction, Type)$, where

Name is the name of the actor,

InputLinks is a function, which returns the set of input links and is defined as

$InputLinks: A \rightarrow \wp(L)$,

$InputLinks(a) = \{l \mid l$ is a link that is connected with an edge directed to the actor $a\}$ where

$\wp(L)$ is the powerset of L,

$\forall a \in A$, $InputLinks(a) \subset L$ and,

$\forall a \in A$, $\forall l \in InputLinks(a)$, $<l, a> \in E$

OutputLinks is a function, which returns the set of output links and is defined as

$OutputLinks: A \rightarrow \wp(L)$,

$OutputLinks(a) = \{l \mid l$ is a link that is connected with an edge directed from the $a\}$ where

$\wp(L)$ is the powerset of L,

$\forall a \in A$, $OutputLinks(a) \subset L$ and,

$\forall a \in A$, $\forall l \in OutputLinks(a)$, $<a, l> \in E$

Rule is a function, which returns the set of rules that describe the actor's behaviour during firing. As it is mentioned above a rule has a pre-condition part and a post-condition part. The functions *PreCond* and *PostCond* return the links that take place in the firing and in the firing results of the actor respectively. In addition, they return the tokens, which are consumed and produced respectively during actor firing. The domain of those functions is the set of rules R; the union of every actor's set of rules constitutes the R. The functions *PreCond* and *PostCond* are defined as

$PreCond: R \rightarrow (L \times T)$ and $PostCond: R \rightarrow (L \times T)$

where

$\forall (l, t) \in PreCond(r)$, $l \in InputLinks(a)$, $a \in A$, $t \in T$ and

$\forall (l, t) \in PostCond(r)$, $l \in OutputLinks(a)$, $a \in A$, $t \in T$

As a result the *Rule* function is defined as

$Rule: A \rightarrow \wp(R)$,

$Rule(a) = \{r \mid r \equiv PreCond(r) \Rightarrow PostCond(r)\}$, where $\wp(R)$ is the powerset of R.

OperationFunction is the function that describes the operations an actor does when its firing is caused

BindingFunction is the function that determines the binding of variables; essentially, it determines the colour value of each token that the actor handles

Type is a function that returns the actor internal complexity and is used during the decomposition of the actor. For example the returned value may be *ContextActor*, *PrivateAction*, *SequenceLibraryActor*, etc.
The set of links L is a non empty set such that

$L=EventLink \cup DesireLink \cup BeliefLink$, where

$EventLink=SystemEventLink \cup SystemCommunicationEventLink \cup UserEventLink$ and

$CommunicationBeliefLink \subseteq BeliefLink$.
In addition, the intersection between the members of L is the empty set, e.g.

$EventLink \cap DesireLink = \varnothing$ etc.
Every link $l \in L$ is a tuple

$l=\{Name, ProducingActors, ConsumingActors, Class, Use, Guard, OperationFunction, Type\}$, where
Name is the name of the link,
ProducingActors is a function, which returns the set of producing actors, which supply the link with tokens and is defined as

$ProducingActors: L \rightarrow \wp(A)$,

$ProducingActors(l)=\{a \mid l \in PostCond(r), \forall r \in Rule(a), a \in A\}$, and

$ProducingActors(l) \subset A$,

where $\wp(A)$ is the powerset of A.
ConsumingActors is a function, which returns the set of the actors, which consume the tokens of the link and is defined as

$ConsumingActors: L \rightarrow \wp(A)$,

$ConsumingActors(l)=\{a \mid l \in PreCond(r), \forall r \in Rule(a), a \in A\}$ and

$ConsumingActors(l) \subset A$,

where $\wp(A)$ is the powerset of A.
Class is a function that returns a predicate, which describes the class of the link and determines the accepted colour of tokens
$Class(l) \in \{EventLink, DesireLink, BeliefLink\}$
Use is a function that returns a predicate, which describes the use that a link allows on its tokens. The returned value *Normal* refers to input links while the returned value *OK* refers to output links (graphically, the links of *Use*=*OK* are represented with a check mark on their edge)

$Use(l) \in \{Normal, OK\}$
Guard is a function that describes the accepted values of the coloured tokens
OperationFunction is the function that describes the operations a link does such as token integrity or error handling based on the value returned by the *Guard* function
Type is a function that returns the link's internal complexity and is used during the decomposition of the link. For example the returned value may be *SystemEventLink*, *DesireLink*, etc.

4.3 Model execution

The model execution is based on the notion of the state of a system, and of changes in state being caused by events. A *state* of an AOMFG represents the distribution of tokens over the

system links at some moment in time and is defined as a function S, such that S: $L \rightarrow \{0, 1, 2,\ldots\}$. AOMFG has an *initial state* S_0 (initial marking), which is the initial assignment of tokens to the system's links. An AOMFG initial state is defined as S_0: $L \rightarrow \{0, 1, 2,\ldots\}$ and consequently an AOMFG together with its initial state is defined as (AOMFG, S_0).

Essentially, the above include the beliefs, desires and intentions of every agent the current moment; furthermore, describe what events are happening in the environment and how the whole multi agent system may evolve the next moment. As the behaviour of agents can be described in terms of agent states and their changes, AOMFG simulates this dynamic behaviour by *state transition*, which is a change from a state S_i to a state S_{i+1}. A state transition causes the redistribution of the tokens to links according to the actors' *firing rules*. The history of the model execution can be considered to be a sequence of states and state transitions.

A state changes according to the state transition function (firing rule). In addition, consider a weight function W on an arc $<x, y>$, where x is a link or an actor and y is an actor or a link, such that W: $E \rightarrow \{1, 2,\ldots\}$. For a link $l \in L$ and an actor $a \in A$, l is an input link l_{in} of a iff $<l, a> \in E$ (there is an arc from l to a), and l is an output link l_{out} of a iff $<a, l> \in E$ (there is an arc from a to l). Every firing rule contains at least one desire link and one event link in its left and right part and optionally some belief links. The AOMFG firing rule is as follows:

1. an actor $a \in A$ is *enabled* if there is a rule $r \in Rule(a)$ with an input desire link d_{in} on its left part ($d_{in} \in PreCond(r)$), which contains token. In this case the agent intends to bring about the specific desire,
2. an enabled actor $a \in A$ *fires* only if the input event link e_{in} of the rule $r \in Rule(a)$ ($e_{in} \in PreCond(r)$), contains token and the rest input links of the rule contain tokens,
3. the actor firing causes the consumption of tokens from all the input links l_{in}, ($l_{in} \in InputLinks(a)$, and $l_{in} \in PreCond(r)$), of the firing rule and tokens' production to all the output links l_{out}, ($l_{out} \in OutputLinks(a)$, and $l_{out} \in PostCond(r)$), of the firing rule, causing a state transition. The new state S' is defined as $\forall l$: $l \in L$ $S'(l) = S(l) + W(a, l) - W(l, a)$.

The above imply that i) actors with the same desire link belong to a certain desire and they all lead to the accomplishment of the same higher goal, ii) all the state transitions are caused by events and this reinforces the model with the properties of responsiveness and pro-activeness, and consequently iii) the undertaken actions lead to a certain goal that reflects a goal-directed behaviour. Furthermore, the history determines the state as previous beliefs (preconditions) are taken into consideration for the decision of the execution of an action.

The above firing rule stands in the general case, whilst according to the actor's type there are some parameters that alter the rule:

- the colour of the tokens determines which rules of an actor are going to fire. The check of the tokens' colour value is realized in the left part of the actor's rule and a new value assignment or production of new tokens is realized in the right part of the actor's rule in association with the binding function
- the consumption of tokens is instantaneous if the actor terminates its execution in one time regular interval. In a different case the tokens of input desire and system event links (d_{in}, e_{in}) are not consumed but they are retained until the actor terminates its execution. After actor termination these tokens are consumed and new tokens are produced in the output links

- every actor has an output desire link d_{out} and an output system event link e_{out} with $Use(l)=OK$. The d_{out} link means the termination of the pursued goal, and the e_{out} link means the successful processing of the event that caused the pursuing of the desired goal. It is mentioned that d_{out} contains no information about the achievement of the goal; instead such information is contained in the actor's output belief links. If tokens exist in the d_{out} and the e_{out} links then the execution of the actor is considered as successful.

Every AOMFG graph describes the alternative ways that an agent can bring about a desire that is the plans the agent has in order to accomplish a goal. Furthermore, considering that the way an agent reasons, or deliberates with the others of the environment where it exists, can be described by the plans that an agent has, then a multi agent system can be considered as the union of all the AOMFG, that is

$$\mathrm{MAS} = \bigcup_i \left(\mathrm{AOMFG}_i, S_{0_i} \right)$$

Considering that $\boldsymbol{L}$ is the set of all MAS links, and *MASS* is the state transition function of a the multi agent system such that $MASS: \boldsymbol{L} \rightarrow \{0, 1, 2, 3, \ldots\}$, then a system state transition is described as

$$MASS'(\boldsymbol{L}) = \bigcup (S(l)+W(a, l)-W(l, a)).$$

According to the firing rules, an actor is enabled when the input desire link of the behavioural rule contains token. This is very important for the model because it means that the agent has a goal and in addition it is committed to that goal; in other words, the agent intends the goal. All the actors that the agent intends are gathered in a list, which the model maintains, called *intention list*. The intention list comprises the third fundamental element of the BDI theory, the intentions. Thus, it is strongly connected with the theoretical model that the AOMFG is based upon. Only one of the actors residing in the intention list can be executed when the next event appears. Every new intended actor is placed at the head of the list and because of the head-to-tail examination of the list the head represents the current context. From the actors residing in the intention list those that belong in the current context are first examined and if no one recognizes the event then the rest of the list are examined. The actors remain in the intention list as long as their execution lasts. When the execution of the actor terminates, the actor is removed from the intention list. The above characteristics deal with the priorities between intentions and especially when an actor is more imperative than another. In the opposite case, belief links can be used as pre-conditions of the actor firing.

The intention list represents the commitments of an agent to the others and to itself. These are parts of its mental state, which is reviewed and updated in every regular time interval. When an actor's rule fires the contents of the intention list are changed according to the actor's type. Thus, the firing of a context actor puts all the contained actors in the intention list, whilst the firing of an action actor usually removes a set of actors from the intention list. This is expected since a context actor contains sub-goals that must be accomplished while the firing of an action actor means the termination of a set of goals. Furthermore, a context actor can represent a persistent goal; the goal persists until the sub-goals are accomplished. For instance, the persistent goal "I persist to avoid crash when I want to go somewhere by car", can be represented by a context actor "*avoid crash*" that is decomposed in several other

actors such as "*driving a car*", "*driving a motorcycle*" and so on, and a belief link "*destination is reached*". The actor "*avoid crash*" is kept in the intention list until it is believed that destination is reached that is the execution of the actor "*driving a car*" terminates and the belief link "*destination is reached*" has a token with value true. The library actors administrate the intention list and depending on their type they add, remove or preserve goals and actions, which must be executed or have been already executed.

4.4 Decomposition

Since an AOMFG is an agent's plan that describes the necessary actions for the fulfilment of a desire, it is useful to support the composition and the decomposition of the plans in order to express higher goals and sub-goals respectively. This is realized with the use of special structures, in the style of dynamic logic (Harel, 1984), which determine the type of context actors and define the decomposition. The model supports goal decomposition and execution of the participating actions in the following ways:

- sequential (;), that is the actions are executed one after the other,
- non deterministic choice (|), that is one of the available actions may be executed,
- concurrency (||), that is the actions are executed in the same regular time interval,
- with test (?), that is proportionately on the conditions which may hold some actions are executed,
- iteration (*), that is the action is executed repeatedly until some termination condition is satisfied.

Actually, the above kinds of decomposition are operators that determine the execution of the plans (Table 1), where a, b are action or context actors and c is the value of a coloured token. The functionality of these operators is embodied in the AOMFG library actors.

a;b	a is executed and then b is executed
a | b	Either a or b is executed but not both
a||b	a and b are executed concurrently
c?;a | ¬c?;b	If c holds then a is executed else b is executed
a*c and ca*	a is executed repeatedly until c is satisfied (*repeat-until* loop) and a is executed repeatedly if c is satisfied beforehand (*while-do* loop)

Table 1. Plan execution operators

4.5 Library actors

The decomposition capabilities of the model are supported with the help of the library actors. Library actors represent structure; they are in fact operators on computations. A library actor is in essence a pair of actors, where one of them constitutes the start (<*library actor name*>-S) of the decomposition and the other constitutes the end (<*library actor name*>-E). Those actors perform the stated actions of decomposition and supervise the correct distribution of tokens to the participating actors, and the correct termination of the execution. As participating actors, it is meant any combination of action, context or group actors. It is mentioned that when a context actor is decomposed then library actors represent this decomposition. The library actors provided by AOMFG are described in the sequel.

Sequence-S, sequence-E

When a token is produced in the input desire link of the sequence-S actor, then it produces tokens in the input desire links of the participating actors. The participating actors are placed in the intention list and remain there until the termination of the execution of the context actor. When a token is produced in the input event link of the sequence-S actor, then it produces tokens progressively in the input event links of the participating actors. The sequence-E actor produces tokens in its output desire and event links when it will receive tokens from all the output links of the participating actors. In addition, it removes the participating actors from the intention list.

Non-deterministic-choice-S, non-deterministic-choice-E

When a token is produced in the input desire link of the non-deterministic-choice-S actor, then it produces tokens in the input desire links of the participating actors. These actors are placed in the intention list and remain there until the termination of the execution of the context actor. When a token is produced in the input event link of the non-deterministic-choice-S actor, then it produces tokens non-deterministically in the input event links of the participating actors. The non-deterministic-choice-E actor produces tokens in the output desire and event links when it receives tokens from the respective output links of the participating actors. In addition, it removes the participating actors from the intention list.

Concurrency-S, concurrency-E

When a token is produced in the input desire link of the concurrency-S actor, then it produces tokens in the input desire links of the participating actors. The participating actors are placed in the intention list, and remain there until the termination of the execution of the context actor. When a token is produced in the input event link of the concurrency-S actor, then it produces tokens in the input event links of all the participating actors. The participating actors are executed concurrently. The concurrency-E actor produces tokens in the output desire and event links when it receives tokens from all output links of the participating actors. In addition, it removes the participating actors from the intention list.

Condition-S, condition-E

The condition-S actor contains condition "if…then…else" where it examines the conditions that must hold and produces tokens in the proper event links of the participating actors. However, it produces tokens in all the input desire links of the participating actors and it places all the actors in the intention list. The condition-E actor checks the termination of the actors' execution, and produces tokens in the output desire and event links. In addition, it removes the participating actors from the intention list.

Iteration-S, iteration-E

The iteration library actor is a kind of execution rather than a decomposition mechanism. Both iteration-S and iteration-E actors may have condition, which determines the start and the end of the execution repetition. In this way can be described loops such as "while…do" and "repeat…until". The same holds for the termination of the context actor, too. The participating actors determine the decomposition and administrate the intention list. If the termination condition of the iteration library actor is not satisfied then the tokens are not consumed from the input desire and event links of the context actor.

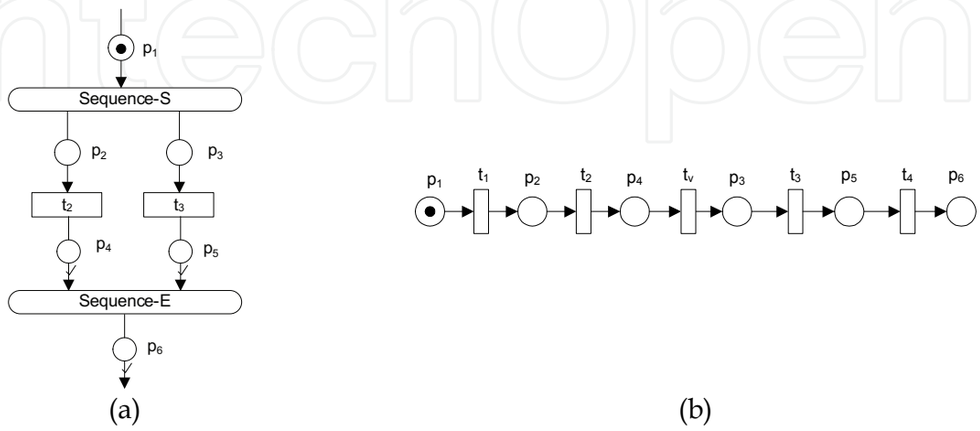
4.6 Transformation to Petri Nets

AOMFG is based on Petri Nets. Links resemble the places of Petri Nets. However, AOMFG provides a variety of link types and helps to the distinguishing of the different events that happen in a multi agent system, as well as to the reflection of the notions of an intentional system. Since Petri Nets are inherently event driven there is no representation of any kind of events. In order to transform an AOMFG to a Petri Net, the event links are eliminated without loss of expressiveness. Although action actors resemble the transitions of Petri Nets, context actors are more complex entities, as they are decomposition or composition structures. Since, the decomposition or composition of context actors is achieved via the library actors, their correspondence to the Petri Nets is presented in the sequel. The transformation of AOMFG model to equivalent Petri Nets makes it subject of analysis methods that imply on Petri Nets.

In Fig. 4, the sequence-S library actor sends tokens to the action actor t_2, and after the execution termination of the latter the action actor t_3 starts execution. In the correspondent Petri Net, the initial marking is at place p_1 and the transitions t_2 and t_3 are executed sequentially; the transitions t_1 and t_4 represent the sequence-S and sequence-E library actors respectively. The virtual transition t_v, simulates the operation of library actors that control the sequential execution.

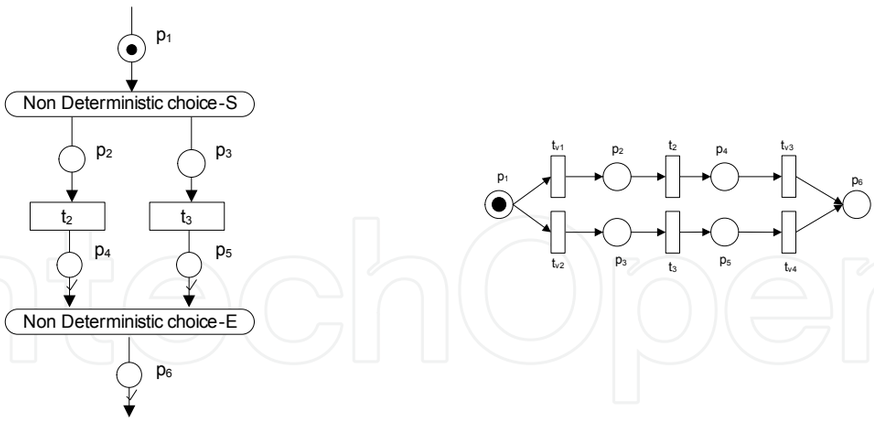
In Fig. 5(a) either action actor t_2 or action actor t_3 is executed. The operation of the non-deterministic-choice-S actor is represented in Fig. 5(b) where either transition t_{v1} or t_{v2} fires. In addition, only one of the transitions t_{v3} or t_{v4} is needed to fire for the process termination. In Fig. 6(a), the initial marking is in link p_1 and the concurrency-S library actor sends tokens to both action actors t_2 and t_3 so they can be executed concurrently. Similarly in Fig. 6(b) the transitions t_2 and t_3 fire simultaneously since transition t_1 provides with tokens both places p_2 and p_4.

In Fig. 7(a), the condition-S library actor sends a token to either action actor t_2 or action actor t_3 depending on the condition cond. Thus, only one of these action actors fires. In a similar way in Fig. 7(b) either transition t_2 or t_3 fires, and only one of the transitions t_{v1} or t_{v2} is needed to fire for the process termination. In Fig. 8(a) the action actor t_2 is executed repetitively until the satisfaction of the condition cond, and in a similar way in Fig. 8(b) transition t_2 does. In Fig. 9(a) if the condition cond=true then the action actor t_2 is executed repetitively, and in a similar way in Fig. 9(b) transitions t_2 does.



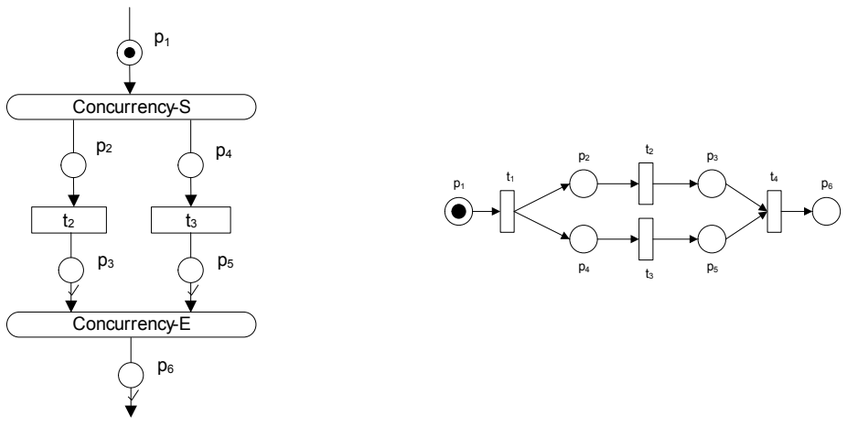
(a) (b)

Fig. 4. Sequential execution



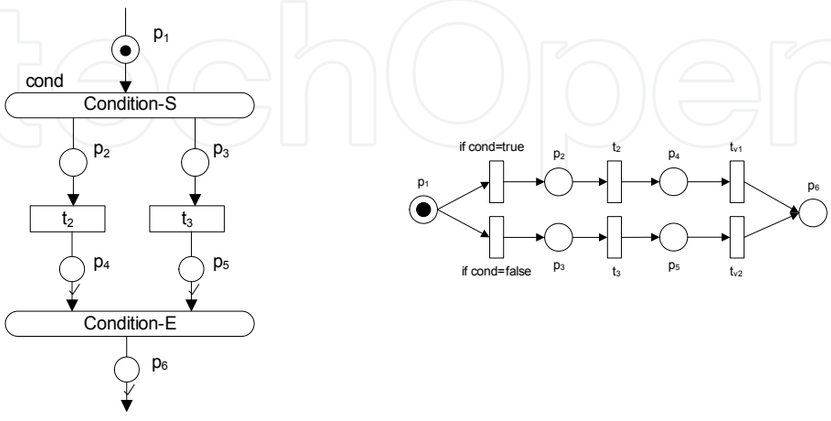
(a) (b)

Fig. 5. Non deterministic choice



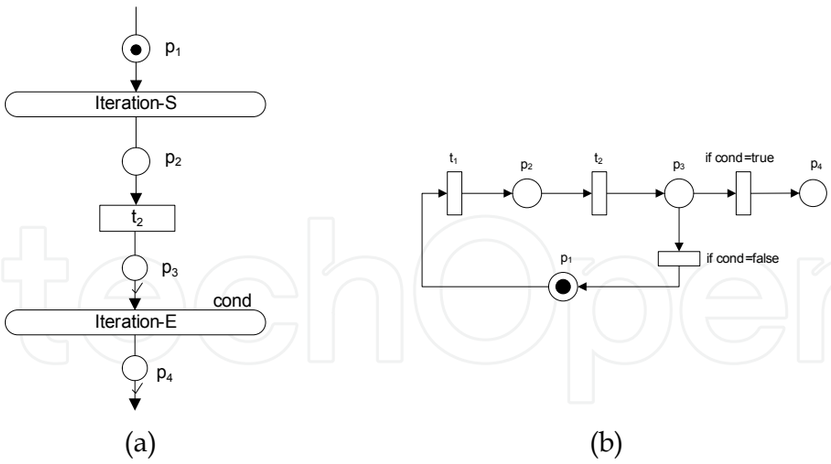
(a) (b)

Fig. 6. Concurrency



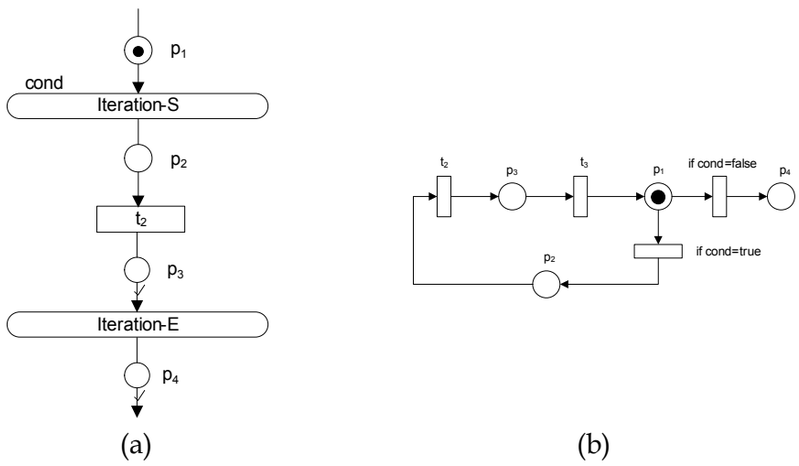
(a) (b)

Fig. 7. Condition



(a) (b)

Fig. 8. Iteration (repeat…until loop)



(a) (b)

Fig. 9. Iteration (while…do loop)

4.7 Discussion

The basic notions of BDI theory are embodied and represented by the structure of AOMFG. The beliefs are represented by the belief links that usually constitute the prerequisite or the results of an agent's actions. Thus, they are strongly connected to the planning of an agent's actions and decisions. The desires are described by the desire links and determine the context where the actions take place in order to achieve the agent's goals. The desires as input links determine the initiation of a higher goal and through the goal decomposition they determine the partially desires for the achievement of this goal. As output links, they examine and mark the successful termination of the executed actions that took place for the goals' achievement. The existence of the appropriate tokens in the desire links indicates the intentions to accomplish desires. All the enabled actors are placed in the intention list and point out the effort and the commitment of the agent to fulfil its desires as well as its obligations to the other agents. The communication between the agents is represented by the foreign action actors and as a part of the agent's actions is in accordance to the speech act theory. The communication belief links describe shared information, while the

communication event links represent the triggering of communication and interaction events. The existence of foreign action actors and communication belief or event links, indicates the awareness of an agent about the capabilities and the needs of the other agents and it is an important factor for the role of every agent as a member of a team. In addition, this reflects implicitly nested mental states as in the example of the section "Specifying a real life system".

The model focuses on the description of the agent's goals and on the different ways they can be achieved. Thus, it does not represent desires, which cannot ever be accomplished. Instead, are represented desires that either can or cannot be accomplished based on the current situation of the environment or on the context they are performed. In this way, the intentions eventually drop. This is in accordance to the models of the Cognitive Engineering and especially those of Cognitive Complexity Theory that regard the human-computer interaction as iterative goal decomposition that leads to a sequence of actions; in addition, they maintain the goals and consequently the subgoals in a short-term memory, which represents the adopted intention to accomplish the goal or even the persistence of the goals. The capabilities of an agent are not represented explicitly by an element of the model but they are specified by the actions that an agent is capable to perform. The participation and the role of an agent in teamwork depend on its capabilities and its desires.

The graphical nature of the model makes it a user-friendly tool for formal specifications of multi agent systems. As it is, in essence, a HLPN, it inherits from Petri Nets many characteristics. Petri Nets are graphical formal specification models with strong mathematical background, powerful analysis techniques, and broad use in a range of applications. AOMFG model based on Petri Nets, incorporates principles of cognitive models and mental notions for the description of the behaviour of the agents, and results a model with theoretical background and an attractive, comprehensible tool, which visualizes the modelling and it is easy to be learnt and used by software engineers and software system designers.

5. Model validation

During specification process a software engineer focuses on those characteristics that he considers critical or not clear. As a result, a system specification may be viewed by different perspectives. In order to demonstrate the generality of AOMFG model and its potential use in many different ways for reasoning about multi agent systems, four different perspectives are used. The case studies that are used for model validation include an algorithmic one and especially Shoham' s algorithm for the generic agent interpreter presented in (Shoham, 1993), a cooperating problem solving one and especially a communication protocol from the standards of FIPA (Foundation for Intelligent Physical Agents) (www.fipa.org), BDI related properties described in (Rao & Georgeff, 1995), and finally the specification of the Dermatology Tutor system (Zaharakis et al., 1998).

5.1 Generic agent interpreter

Shoham proposed the agent-oriented programming (AOP) paradigm as a specialization of object-oriented programming paradigm. Particularly, "AOP specializes the framework by fixing the state (now called mental state) of the modules (now called agents) to consist of components such as beliefs (including beliefs about the world, about themselves, and about one another), capabilities, and decisions, each of which enjoys a precisely defined syntax.

Various constraints are placed on the mental state of an agent, which roughly correspond to constraints on their common sense counterparts. A computation consists of these agents informing, requesting, offering, accepting, rejecting, competing, and assisting one another" (Shoham, 1993, p.56). In AOP a multi modal formal language is introduced which describes the agents' mental state and a generic agent interpreter that describes the agents' behaviour. The interpreter works by consistently iterating between the following processes:

- read the current messages, and update your mental state, including your beliefs and commitments (the agent program is crucial for this update)
- execute the commitments for the current time, possibly resulting in further belief change (this phase is independent of the agent's program) (Shoham, 1993, p.68).

Time is counted in regular intervals determined by the target execution platform. According to the above algorithm there are two independent processes, which run iterative at regular intervals.

A private AOMFG action actor, *the update-mental-state*, and a context AOMFG actor, the *execute-commitment* can represent these processes (Fig. 10). The *update-mental-state* action actor checks and updates the agent's mental state according to the model that specifies its properties. The check and update mechanisms are not examined from the presented perspective; instead they can be implemented with the use of a programming language. The system event input link s_{in}=*sel1* and the desire input link d_{in}=*ums* represent the system qualification for the fulfilment of the goal to check or update its mental state. The system communication input event link ce_{in}="*incoming messages*" represents the reading of incoming messages, which may be several requests or informs from other agents of the environment. The content of these messages is the subject that is examined by the mental state. The AOMFG model generates tokens in the s_{in}=*sel1* and the d_{in}=*ums*. When an incoming message exists a token is generated in the ce_{in}="*incoming messages*". When the action *update-mental-state* terminates then tokens are produced in the output belief link b_{out}="*update results*", which represents the conclusions of the check and update of the mental state, and in the system and desire output links s_{out}=*sel1.OK* and d_{out}=*ums.OK* respectively. The AOMFG model generates tokens in the s_{in}=*sel2* and the d_{in}=*ec* too, however the "*execute commitment*" context actor cannot start execution until b_{in}="*update results*" contains tokens. The execute commitment process is represented by a context actor since it is a process that corresponds to many actions the agent is able to perform and may be decomposed to several other actions. Because of its general nature, it is not further decomposed here. When the process terminates, tokens are produced in the system and desire output links s_{out}=*sel2.OK* and d_{out}=*ec.OK* respectively. Any resulting information is stored in the belief link b_{out}="*action results*", and if there are messages for other agents then the communication event ce_{out}="*outgoing messages*" is triggered.

AOMFG turned out to be a general enough model and capable of describing in a clear and straightforward way the two independent processes contained in the algorithm proposed by Shoham for his generic agent interpreter, which is the basis of the agents that can be built according to the AOP paradigm. The generality of the algorithm results in generality of the graphs, and lies mainly on the second process of the algorithm and consequently on the context actor "execute commitment". Nevertheless, in an implementation phase this context actor corresponds to a number of other actors that constitute instances of this actor. These instances can be executed concurrently or can be suspended as it happens in AGENT0 that implements Shoham' s algorithm. Furthermore, AOMFG can be used to describe the agent mental state and the alternative actions an agent may choose. However, the notion of time

that is used explicitly in AOP cannot be represented in AOMFG in a straightforward way. Instead, the use of belief links as action preconditions can substitute the explicit reference to time.

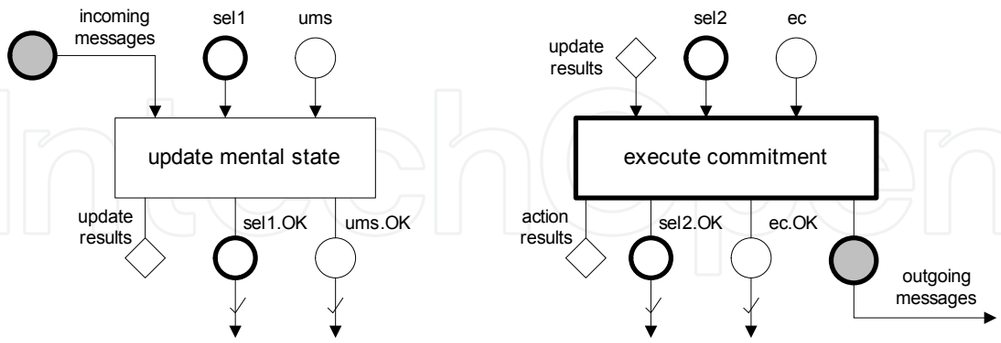


Fig. 10. Shoham' s generic agent interpreter

5.2 FIPA communication protocols

The Foundation for Intelligent Physical Agents (FIPA) has published a series of specification documents including the Communicative Act Library Specification (www.fipa.org). In this document, in addition to the formalism, a series of standard communication protocols between agents is presented. These protocols are based on *speech act theory* (Searle, 1969) and include communication actions such as request, request-whenever, request-when. Due to space limitations, the AOMFG is used to describe FIPA-request protocol only.

According to FIPA-request protocol, an agent requests an action from another agent and the latter either does not understand the message, or refuses the action and justifies the refusal, or agrees to perform the requested action and after the action termination it informs the requester agent about the results of the action.

For representing the FIPA-request protocol, two processes take place: the send process (Fig. 12), which is fired when a request is desired, and the receive process (Fig. 11), which is fired when a request has happened and there is a need for reaction. In essence, the AOMFG for the send process describes how an agent behaves after the request has been sent. The same holds for the receive process, i.e., after a request has been received. This concession has been done since it is trivial to describe these two processes e.g., by two single events.

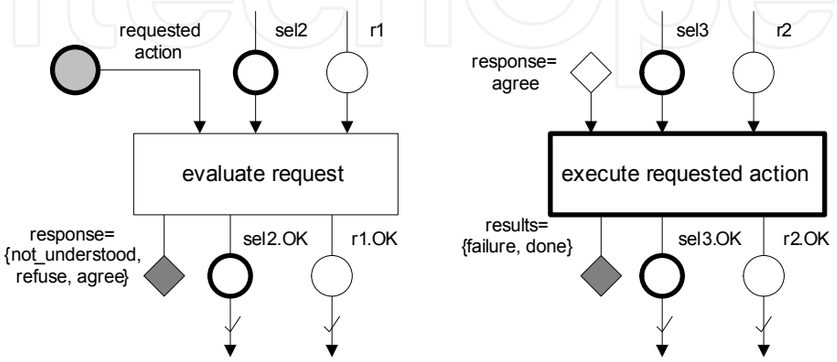


Fig. 11. FIPA-request protocol: receive process

Suppose that an agent (the receiver) received a request message from another agent (the sender). This is represented by the communication input event link ce_{in}="*requested action*". The receiver evaluates the request with action actor "*evaluate request*" and has to respond that it either did not understand the message, or that it refuses the action or that it agrees to perform the action. The response is represented as communication output belief link cb_{out}="*response*". It is mentioned that the values of cb_{out}="*response*" are one among not_understood, refuse or agree. If the receiver agrees to perform the requested action then the context actor "*execute requested action*" is fired since there exist tokens in its entire input links. After the execution of the action, the receiver has to inform the sender about the results of the performed actions. Those results are represented by the cb_{out}="*results*", which may have the values failure or done.

Suppose now, that an agent (the sender) sends a request message to another agent (the receiver). After the evaluation of the request by the receiver, a communication event link represents receiver's response as ce_{in}="*response*". The whole process is specified by a condition library actor, which checks the content of ce_{in}="*response*" and enables the appropriate action actor. In order to achieve this, the condition-S library actor contains the following rules:

i. if *response=not_understood* then (*sel1.1* and *ra1*)
ii. if *response=refuse* then (*sel1.2* and *ra2*)
iii. if *response=agree* then (*sel1.3* and *ra3*)

Since the successful termination of only one action actor is necessary, the condition-E library actor contains the rule:

if {(*sel1.1.OK* and *ra1.OK*) or (*sel1.2.OK* and *ra2.OK*) or (*sel1.3.OK* and *ra3.OK*)} then (*sel1.OK* and *ra.OK*)

The action actor "*wait until requested action terminates*" does not consume the tokens of its input links until it is believed that the action has been performed. The fact that something is believed can be modelled by another AOMFG that specifies the agent's mental state such as the one of Fig. 10.

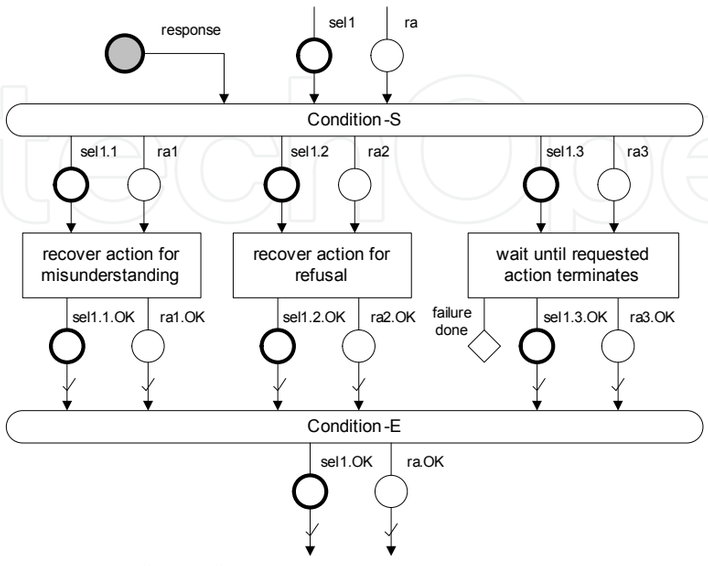


Fig. 12. FIPA-request protocol: send process

AOMFG was used to describe communication between agents in the sense of speech act theory. For this reason, the FIPA-request protocol was used as a test case. Although FIPA standards describe the communication process from an external observer's view, in this work the description is from an internal observer's view and focuses on the reactions of the agent in a communication action. This is due to the reasons of unity, uniformity, continuity and autonomy of the graphs and the specification method. Since AOMFG is used for the specification of the reasoning process and the actions of every agent separately, the communication should not be an exception. Besides, the model regards and manages the communication actions as physical actions. However, a different use of the model (external observer's perspective) is possible and this is due to the designer's choice.

5.3 BDI related properties

The AOMFG model reflects and explicitly represents the notions described by the BDI formalism, so this section attempts to represent some BDI related properties. The used BDI logic is a multi modal temporal logic based on the branching time logic CTL* (Emerson, 1991). The basic concept of the BDI logic is the transformation of a decision tree and the functions applied to it, to an equivalent model that represents beliefs, desires and intentions as accessibility relations over a set of possible worlds. Each world is a time tree with a single past and a branching future, with nodes corresponding to a possible system state and representing the options available to the system itself or the state of the environment which the system is embedded in, and with arcs (transitions) representing the different system actions for the achievement of an objective.

In this way, there are *belief-*, *desire-* and *intention-accessible worlds* corresponding to the states that the system believes to be possible, desires to bring about and intends to bring about, and *belief-*, *desire-* and *intention-accessibility relations* ($\mathcal{B}$, $\mathcal{D}$ and $\mathcal{I}$, respectively) which are the transitions between the worlds. The detailed syntax and semantics of the BDI logic are described in (Rao & Georgeff, 1995); in the following only its basic properties are briefly presented.

	Beliefs	**Desires**	**Intentions**
K-axiom	$\mathsf{BEL}(p) \wedge \mathsf{BEL}(p \supset q) \supset \mathsf{BEL}(q)$	$\mathsf{DES}(p) \wedge \mathsf{DES}(p \supset q) \supset \mathsf{DES}(q)$	$\mathsf{INTEND}(p) \wedge \mathsf{INTEND}(p \supset q) \supset \mathsf{INTEND}(q)$
D-axiom	$\mathsf{BEL}(p) \supset \neg\mathsf{BEL}(\neg p)$	$\mathsf{DES}(p) \supset \neg\mathsf{DES}(\neg p)$	$\mathsf{INTEND}(p) \supset \neg\mathsf{INTEND}(\neg p)$
4-axiom	$\mathsf{BEL}(p) \supset \mathsf{BEL}(\mathsf{BEL}(p))$		
5-axiom	$\neg\mathsf{BEL}(p) \supset \mathsf{BEL}(\neg\mathsf{BEL}(p))$		

Table 2. The axioms of the KD45 system

The normal modal system KD45 (weak-S5) is adopted for beliefs, while the K and D axioms are adopted for desires and intentions. K-axiom states that if an agent believes (desires or intends) p and believes (desires or intends) that $p \supset q$ then it will believe (desire or intend) q. D-axiom says that the agent' s beliefs (desires or intentions) are not contradictory. 4-axiom and 5-axiom are respectively the positive and negative introspection axioms: 4-axiom states that an agent is aware of what it knows and 5-axiom says that an agent is aware of what it does not know. The axioms of the KD45 system are summarized in Table 2, where p, q are

propositions of the classical propositional logic, "∧"and "⊃" are the propositional connectives for conjunction and implication, and BEL, DES and INTEND are modal operators representing the agent's beliefs, desires and intentions respectively.
The necessitation rule is applied in beliefs, desires and intentions (Table 3)[1]:

	Necessitation rule
Beliefs	If $\vdash p$ then $\vdash \mathsf{BEL}(p)$
Desires	If $\vdash p$ then $\vdash \mathsf{DES}(p)$
Intentions	If $\vdash p$ then $\vdash \mathsf{INTEND}(p)$

Table 3. The necessitation rule for beliefs, desires and intentions

As the belief-, desire- and intention-accessible worlds are sets and in the same time are time trees, there are set relationships ($\subseteq$, $\cap$ etc.) and structure relationships (sub-world, identical or incomparable worlds) between them. Depending on the combinations of the above relationships and the axiomatizations of them, three constraints are under consideration: *strong realism* (Rao & Georgeff, 1991a), *realism* (Cohen & Levesque, 1990) and *weak realism* (Rao & Georgeff, 1991b). Structurally, AOMFG reflects the properties of realism since according to realism constraint an agent intends the actions it desires to bring about and it desires the actions it believes that can be achieved. In AOMFG model, the desires are represented by desire links and the intentions by desire links containing tokens. In addition, it is known the existence of the desire links a priori. Formally, $\mathcal{I} \subseteq \mathcal{D} \subseteq \mathcal{B}$, that is if an agent intends an action then it desires it and it believes it too.
However, AOMFG does not restrict the designer of an application to use one or another constraint. It is the designer's responsibility to use AOMFG in such a way that the properties of a constraint can be reflected. So, it is possible to reflect the properties of the other constraints too as this is depending on the broader concept in which, designer uses the model. Some of the characteristics of the weak realism are referred in the following, as it is the constraint for which some properties will be described by AOMFG. According to weak realism, the intersection between the belief-accessible worlds and desire-accessible worlds is not null and an agent does not desire a proposition the negation of which is believed. The same holds between desire- and intention-accessible worlds and between belief- and intention-accessible worlds. An agent does not intend a proposition the negation of which is desired or is believed (Table 4).

Semantic Condition	**Distinguishing Axiom**
$\mathcal{B} \cap \mathcal{D} \neq \varnothing$	$\mathsf{BEL}(p) \supset \neg\mathsf{DES}(\neg p)$
$\mathcal{D} \cap \mathcal{I} \neq \varnothing$	$\mathsf{DES}(p) \supset \neg\mathsf{INTEND}(\neg p)$
$\mathcal{B} \cap \mathcal{I} \neq \varnothing$	$\mathsf{BEL}(p) \supset \neg\mathsf{INTEND}(\neg p)$

Table 4. The BDI modal system for weak realism

Some other properties that characterize the weak realism are the *asymmetry thesis* proposed by Bratman (Bratman, 1987) and extended by Rao and Georgeff (Rao & Georgeff, 1991b). Among them the intention-belief incompleteness ($\nvdash \mathsf{INTEND}(p) \supset \mathsf{BEL}(p)$)[2], which is

[1] '⊢' means that the formula which follows is valid

[2] '⊬' means that the formula which follows is not satisfiable

allowed by weak realism, is selected to depict how this property can be represented by using AOMFG. In essence, it means that an agent intends to do an action but does not believe that it will do it. As a validation scenario is used the example referred in (Rao & Georgeff, 1995, p.33), in which Robbie the robot has an intention of opening the door when the bell rings but it does not believe that it will open the door when the bell rings because it might be serving beer.

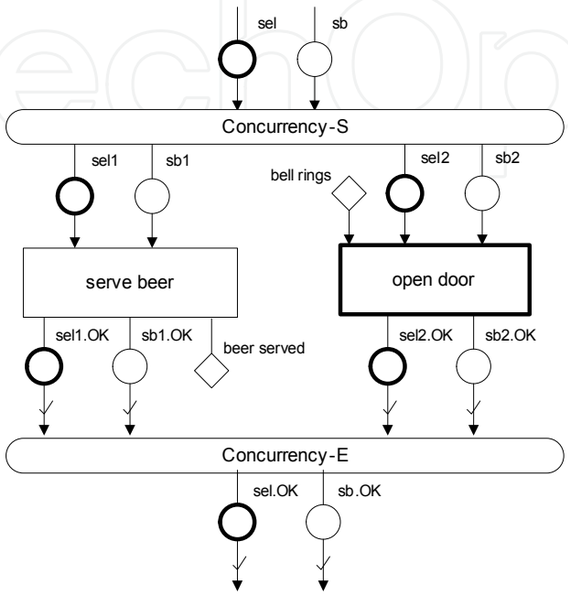


Fig. 13. Concurrent tasks for Robbie

Robbie serves beer while the bell rings (Fig. 13). The context actor *"open door"* is fired and is decomposed in the graph of Fig. 14. It is mentioned that both actors *"serve beer"* and *"open door"* may be in the intention list since Robbie might not have finished serving. In this case, the action actors *"move"* and *"open"* cannot fire since the condition link *"beer served"* contains no tokens and consequently no tokens can be produced in belief link *"door opened"*. So action actor *"do nothing"* fires and the context actor *"open door"* terminates its execution. It is mentioned that although *"do nothing"* is a void action it is considered as an action (in the flavour of speech act theory) that affects the environment as any other action does.

In the above, AOMFG was used to describe BDI related properties. Since it is based on BDI theory most of the properties are embodied in its semantics. Although the model structurally reflects the realism constraint, other BDI properties can be described too, but it is mentioned that is critical how the model will be used for such a purpose. This is not necessarily a drawback since even in theory there is a need of different logics in order to express the several constraints and their asymmetry thesis. It is mentioned that a primary aim of the AOMFG is to hide the complexity of strict formalisms from designers who are not experts in formal specification methods.

5.4 Specifying a real life system

General description of the Dermatology Tutor

The Dermatology Tutor (Zaharakis et al., 1998) is an intelligent tutoring system used for teaching Dermatology at the Department of Dermatology, School of Medicine, University of

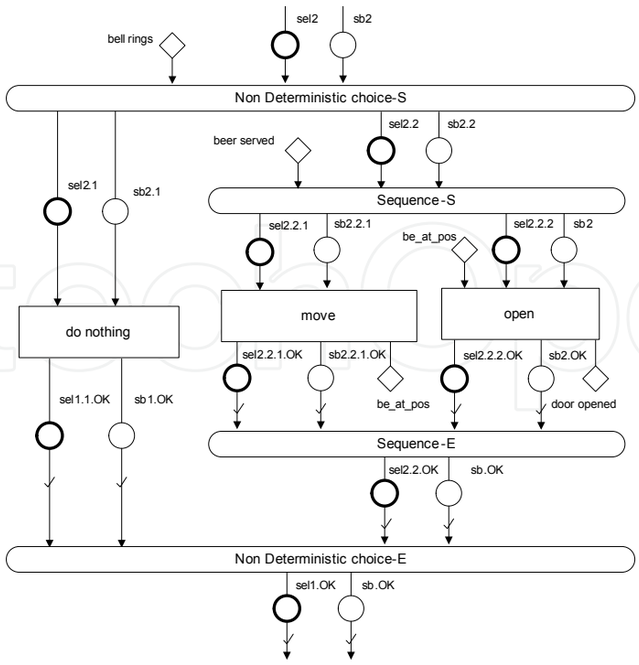


Fig. 14. Alternative actions according to Robbie's mental state

Patras. The Dermatology Tutor uses a self-organized society of autonomous software agents who have different capabilities or roles. In particular, the tutoring system for teaching dermatology consists of nine agents. A user can interact with the multi agent system through the user interface, which processes all the requests that come from the environment outside the system (i.e. the user). Depending on the event, this module (which has not been implemented as an agent) activates either the Dermatology Agent or the Help Agent. If they have a desire for the requested action and they believe that it is possible, then they adopt an intention about it. However, if there is a need to cooperate for task accomplishment then a structured society is constructed. As the architecture is composed for tutoring purposes, only the *Dermatology Agent* can act as an instructor; all the instructional strategies necessary for the tutoring process are contained in this agent's plans. The other agents populating the society called *medical agents*, can be members of a team formed by the Dermatology Agent and they can also communicate each other. Currently seven medical agents have been implemented: Histology, Physiology, Biology, Biochemistry, Microbiology, Radiology and Immunology agent. Each of them specializes on the corresponding medical domain and can provide knowledge and information about the domain matters. Furthermore, medical agents can request an action from third level agents (information agents) who may be local or remote and are responsible for information retrieval. The information agents are registered to the system and medical agents are aware of their existence. Although the information retrieval may be performed by a non-agent software system, their behaviour could be considered as one of an autonomous agent.

When the trainees use Dermatology Tutor, they have to select from a list of available actions, which includes "Teach", "Topic", "Search" and "Help". Each action causes the execution of one of the agent's high-level plans. For each trainee, a student environment is created, which records performance information. Currently, Dermatology Tutor creates a set of files for each trainee, where it records personal data, as name, year of study, specialty and performance

data, as the goals accomplished, the learning units seen, the time spent on each plan or learning unit, the answers given and the score taken in each test, and the level of competence. In this way, Dermatology Tutor records the last position of the trainee, while performance information is used to decide the values of the preconditions of the agents' plans.
As described above, the plans specify the ways that an agent brings about its intentions. They refer to an agent's desires and they consist of actions that must be executed. There may exist more than one plan for each desire. Typically, plans have a name (plan name), pre-conditions that describe the plan triggering, and a body part, which describes the actions that will take part in the plan execution.

Using AOMFG for the specification of Dermatology Tutor

Suppose that the issue of psoriasis has to be taught. The Dermatology agent has a desire named "Teach psoriasis" and a plan in its plan library in order to carry out such a desire (Table 5). Each step of this plan constitutes a lower-level plan that can be further analyzed. The Dermatology agent is aware of the medical agents and can form teams in order to go through with its goals. In the following, the etiology and pathogenesis of psoriasis plan is examined. Dermatology agent adopts an intention about this goal and executes the corresponding plan (Table 6). In order to do this it needs to form a team with Biology, Biochemistry, Immunology, and Histology medical agents.

Plan name:	*Teach psoriasis*
Pre-conditions:	
Plan body:	Definition and Epidemiology
	Clinical View
	Histological View
	Etiology and Pathogenesis
	Treatment

Table 5. Dermatology agent plan for teaching psoriasis

Plan name:	*Etiology and Pathogenesis*	
Pre-conditions:		**Associated Agents**
Plan body:	Disorder of protein expression in keratinocytes	Biology, Biochemistry
	Antigenic stimulation and immunology activation	Immunology
	Proteases and intense mitotic activity	Biology, Biochemistry
	Metabolic disorders	Biochemistry
	Histopathologic changes of psoriasis	Histology

Table 6. Etiology and Pathogenesis plan of the Dermatology agent

The representation of "Etiology and pathogenesis" plan of the Dermatology agent with an AOMFG is depicted in Fig. 15. The plan is decomposed in foreign actions, which are requested from other agents. The context actor "Proteases and intense mitotic activity" is further decomposed in Fig. 16. The input desire link *ep* corresponds to the desire to teach the topic of etiology and pathogenesis of psoriasis, while the ep_i, i=1,…,5, correspond to the sub-goals which must be accomplished. When the agent adopts an intention to bring about this desire, a token is produced in the *ep* link. In addition, a token is produced in the system event link *sep*.

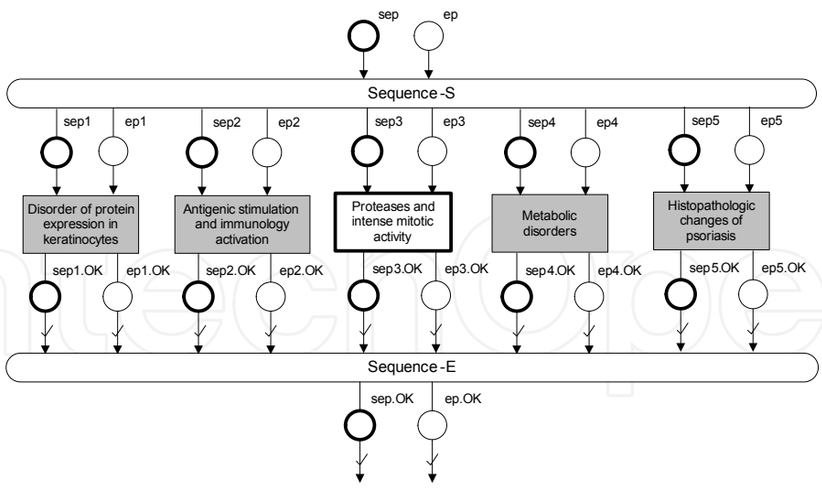


Fig. 15. Etiology and pathogenesis AOMFG

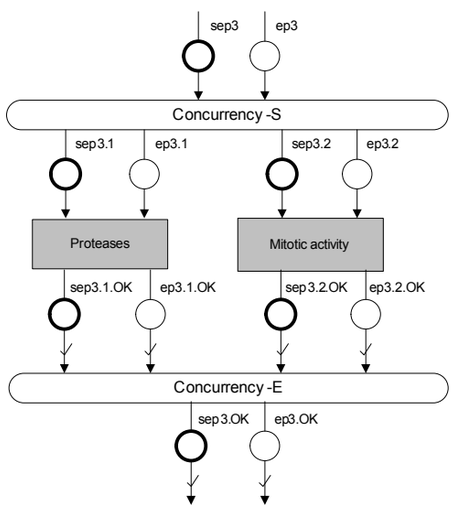


Fig. 16. Decomposition of "Proteases and intense mitotic activity" context actor

As the *sep* and *ep* input links contain tokens, the Sequence-S library actor produces tokens to every ep_i. All the actions are put in the intention list of the Dermatology agent as they are ready to fire (enabled). In addition, the Sequence-S library actor produces a token to the *sep1* and the action "Disorder of protein expression in keratinocytes" fires. When the execution of the action terminates, then the tokens of *sep1* and *ep1* are consumed and tokens are produced to *sep1.OK* and *ep1.OK*. Afterwards, the Sequence-S library produces a token to *sep2* and the same process follows. A concurrency library actor decomposes the context actor "Proteases and intense mitotic activity" and this means that when the actor fires, both actions "Proteases" and "Mitotic activity" are executed concurrently.

In order to fulfil its first partial goal ("Disorder of protein expression in keratinocytes") Dermatology agent requests the action from Biochemistry and Biology agents. As both Biochemistry and Biology agents have a desire about the requested action they adopt an intention about it (Table 7 and Table 8). AOMFG in Fig. 17 and Fig. 19 represent the plans of the Biochemistry and Biology agents respectively. The "Disorder of protein expression in

keratinocytes" plans are detailed while the rest can be similarly represented. When both Biology and Biochemistry agents request this action, a token is produced in each input communication event links of the corresponding AOMFG in Fig. 18 and Fig. 20. Both agents adopt an intention, which means they produce a token to *kp* and *pe* desire links respectively. Furthermore, they produce a token to the *skp* and *spe*, respectively.

Plan name:	*Disorder of protein expression in keratinocytes: Keratinocyte proteins*	
Pre-conditions:		**Associated Agents**
Plan body:	Types of keratinocytes	
	Protein conformation	Biology
	Abnormal conformation	Biology
Plan name:	*Proteases*	
Pre-conditions:		**Associated Agents**
Plan body:	Proteases definition	
	Abnormalities	
Plan name:	*Metabolic disorders*	
Pre-conditions:		**Associated Agents**
Plan body:	Disorder of metabolic fermentations	
	• Disorder of metabolism of the cyclic nucleotides	
	• Disorder of metabolism of the arachidonic acid	
	• Disorder of metabolism of the polyamide	
	• Disorder of metabolism of the phosphatides	
	Vascular dysfunction	

Table 7. Plans of the Biochemistry agent

Plan name:	*Disorder of protein expression in keratinocytes: Protein expression*	
Pre-conditions:	*Protein conformation, Abnormal conformation*	**Associated Agents**
Plan body:	Normal expression of keratinoproteins	Biochemistry
	Abnormal expression of keratinoproteins	Biochemistry
Plan name:	*Mitotic acivity*	
Pre-conditions:		**Associated Agents**
Plan body:	Phases of mitotic activity	Immunology
	• keratinocytes	
	• macrophages	
	• lymphocytes	
	• mast cells	
	• neutrophilic cytes	
	Abnormalities of mitotic activity	

Table 8. Plans of the Biology agent

Biology agent cannot execute the first step ("Normal expression of keratinoproteins") of its "Disorder of protein expression in keratinocytes: Protein expression" plan, because it has to wait for precondition "Protein conformation" to become true. Neither can it execute the second step, due to precondition "Abnormal Conformation" being false. These preconditions are represented by the communication beliefs *Pc* and *Ac.* Biochemistry agent handles both preconditions. When the latter can infer the truth or the falsity of those facts, it will inform Biology agent about the results. This happens only after the execution of the actions "Protein conformation" and "Abnormal conformation" respectively, which cause the production of a token to each *Pc* and *Ac* communication belief links. Note that each medical agent that participates in a team is responsible for informing all other medical agents of the same team about all changes in its environment.
The above process indicates implicitly how AOMFG deals with some kind of nested beliefs. More precisely, Biology Agent cannot execute the actions "Normal expression of keratinoproteins" and "Abnormal expression of keratinoproteins" until it believes that Biochemistry agent believes that Pc and Ac are true, i.e.,

BEL("Biology Agent" BEL("Biochemistry Agent", "Pc"))
BEL("Biology Agent" BEL("Biochemistry Agent", "Ac"))

Equivalently, some kind of other nested mental states can be represented, for example, as described above, Dermatology agent requests the action "Metabolic disorders" from Biochemistry Agent and implicitly desires what Biochemistry agent desires, i.e.,

DES("Dermatology Agent" DES("Biochemistry Agent", "Metabolic disorders"))

When precondition "Protein Conformation" becomes true, then Biology agent executes the step "Normal expression of keratinoproteins". At the same time, the second precondition ("Abnormal Conformation") may become true, which means that Biology agent may execute the step "Abnormal expression of keratinoproteins". When both agents bring about their intentions they inform Dermatology agent for the results of the requested actions. Afterwards it attempts to fulfil the rest of its partial goals. After the execution of the "Etiology and pathogenesis" plan, Dermatology agent drops its intention for this plan and it continues with the next plan.
Thus, in order to validate AOMFG with a real system such as The Dermatology Tutor, the following steps were taken. Initially, the plans of all agents involved were analyzed. In this way, the agents' capabilities, desires and alternative ways of their achievement were expressed. Next, the agents' awareness of their environment and their interaction was

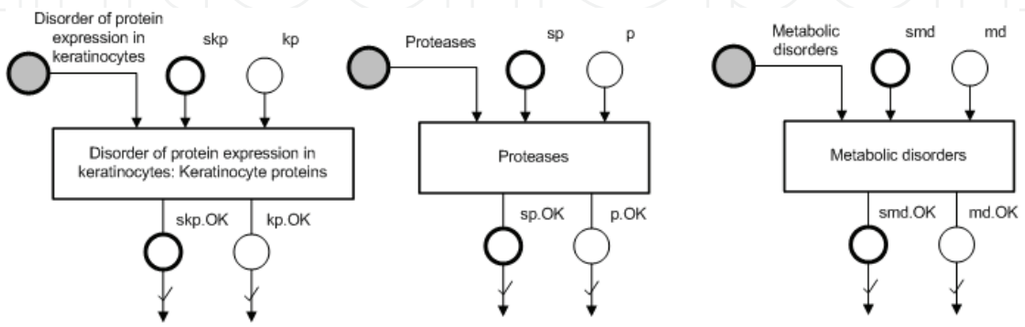


Fig. 17. AOMFG represents the plans of Biochemistry agent

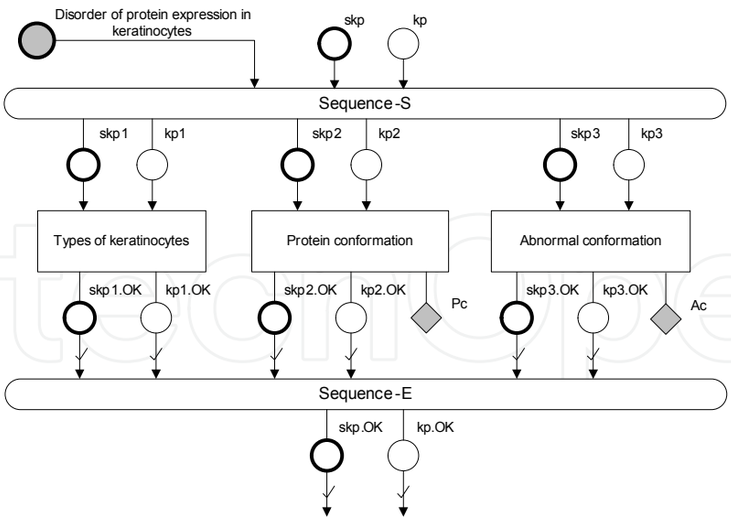


Fig. 18. Decomposition of "Disorder of protein expression in keratinocytes: Keratinicytes protein" plan

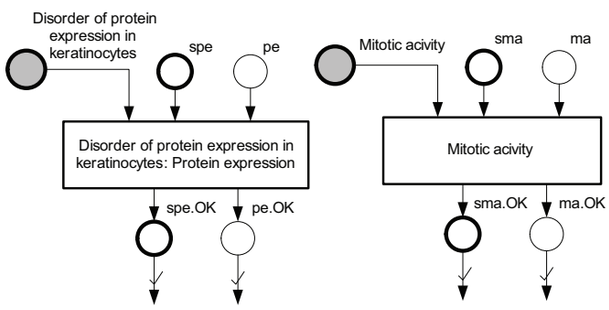


Fig. 19. AOMFG represents the plans of Biology agent

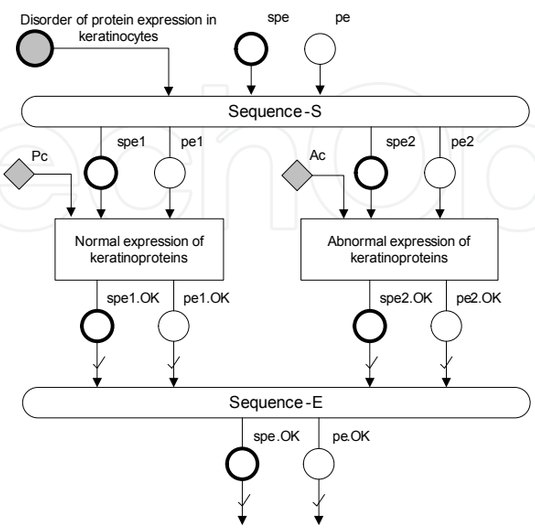


Fig. 20. Decomposition of "Disorder of protein expression in keratinocytes: Protein expression" plan

expressed with the use of the foreign action actors and communication event and belief links. Finally, their roles inside the formed teams according to their knowledge and capabilities were determined.
The design of the Dermatology Tutor was based on the description and the analysis of every agent separately and this gave a detailed view of the functionality and the participation in the team action. Thus, there are several autonomous design parts that relate with each other through communication actions, i.e. every agent is described by a set of graphs and the union of these graphs constitutes the whole system description. This approach does not allow a global and clear view of the composition and structure of the teams unless all the graphs are collected and studied. As a result, the team composition and the role of every agent are inducted from the foreign actors and the communication links. However, in the Dermatology Tutor the teams and the role of the agents are not determined at design time; instead the teams are formed dynamically depending on the tutoring process that justifies such a design approach.

6. Summary

Formal specifications are necessary for reasoning about multi agent systems. However, existing methods are not practical for wide use from designers and software engineers. In this work, AOMFG is introduced as a graphical model for formal specifications of multi agent systems. The model is based on well-defined formalism and possesses the mechanisms for expressing mental notions such as beliefs, desires and intention that are used for the description of the agents' behaviour. It provides ready to use mechanisms for goal decomposition and it also facilitates the expression of social activity of agents as it handles several kinds of events by providing reaction and communication mechanisms, and structures for the awareness of the environment.
In particular, AOMFG is based on the well-defined and broadly accepted BDI theory as well as on the principles of agent-oriented programming. The BDI theory has been chosen as the model underlying formalism because it reflects the intentional stance and employs a strong mathematical background. The principles of agent-oriented programming also reflect the intentional stance, in the same manner as the societal view of agents. In addition, AOMFG employs the most commonly used principles of state-based machines, programming languages, Cognitive Engineering and interaction specification models in a compact representation.
The presented model differs from the other related models in that it embodies in its building blocks the BDI and AOP concepts fused together with features taken from disciplines that relate to the design of applications based on multi agent systems. Furthermore, the graphical nature of AOMFG, which is based on High Level Petri Nets, provides a friendly way for multi agent systems specification by hiding from users the underlying use of mathematical formalism, which may be hard to use by non-specialists.
AOMFG is general enough to be used in a wide range of agent applications as well as to encapsulate other well-known agent theories. In the above sections, AOMFG model was used and validated under theoretical issues as well as in the description of a real system in order to provide evidence that it can cover the specification needs of both directions. In fact, as demonstrated, an AOMFG can be used for the specification of generic agent interpreters, FIPA communication protocols, and BDI properties as well as for the specification of real life systems. Furthermore, an AOMFG can be transformed to an equivalent Petri Net, which may then be subjected to formal analysis techniques. Other researchers and mainly practitioners in software specifications may use the model for easy and rapid specification and prototyping of multi agent systems. Programmers can also use the model as it embodies

programming structures familiar to them. The inclusion of an explicit representation of time in AOMFG is currently under investigation.

7. Acknowledgements

This work was initially carried out in Educational Software Development Laboratory, Department of Mathematics, University of Patras, Greece. I would like to thank Prof. A. D. Kameas and Prof. P. E. Pintelas for their support, their helpful comments and their valuable reviews.

8. References

Bates, J. (1994). The Role of Emotion in Believable Agents, *Communications of the ACM*, 37 (7) 122-125, ISSN: 0001-0782.

Bratman, M. E. (1987). *Intentions, Plans, and Practical Reason,* Harvard University Press: Cambridge, MA.

Card, S., Moran, T. P. & Newell, A. (1983). *The Psychology of Human-Computer Interaction,* Lawrence Erlbaum Associates Inc, ISBN: 0-898-59243-7, New Jersey.

Cohen, P. R. & Levesque, H. J. (1990). Intention is Choice with Commitment. *Artificial Intelligence*, 42 213-261, ISSN: 0004-3702.

Dennett, D. C. (1987). *The Intentional Stance*, The MIT Press, ISBN 0-262-54053-3, Cambridge, MA.

Emerson, E. A. (1991). Temporal and Modal Logic, In: *Handbook of Theoretical Computer Science*, J. van Leeuwen, (Ed.), 995-1072, The MIT Press, ISBN: 0-444-88074-7, Cambridge MA, USA.

Gasser, L. Braganza, C. & Hermann, N. (1987). MACE: A Flexible Testbed for Distributed AI Research, In: *Distributed Artificial Intelligence*, M. Huhns, (Ed.), 119-152, Morgan Kaufmann Publishers Inc., ISBN: 0-934-61338-9, San Francisco, CA.

Genrich H. J., & Lautenbach K. (1981). System Modeling with High-Level Petri Nets, *Theoretical Computer Science*, 13 109-136, ISSN: 0304-3975.

Gerogiannis, V. C., Kameas, A. D. & Pintelas, P. E. (1998). Comparative Study and Categorization of High-Level Petri Nets, *Journal on Systems and Software,* 43(2) 133-160, ISSN: 0164-1212.

Harel, D. (1984). Dynamic Logic, In: *Handbook of Philosophical Logic Volume II - Extensions of Classical Logic*, D. Gabbay and F. Guenther, (Eds.), 497-604, D. Reidel Publishing Company: Dordrecht, ISBN: 0-792-33097-8 The Netherlands.

Hintikka, J. (1962). *Knowledge and Belief,* Cornell University Press, NY.

Jensen, K. (1981). Coloured Petri Nets and the Invariant Method. *Theoretical Computer Science*, 14 317-336, ISSN: 0304-3975.

Jensen, K. (1997). *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use,* Vol. 2, Springer-Verlag, ISBN: 3-540-58276-2, Berlin Heidelberg New York.

Johnson, P. & Nicolosi, E. (1990). Task-based User Interface Development Tools, in: *Proceedings of the 3rd IFIP TC 13 International Conference on Human-Computer Interaction (INTERACT 90)*, pp. 383-387, August '90, Cambridge, UK.

Kameas, A. D. (1995). *A Formal Model for the Specification of Interaction and the Design of Interactive Applications*, Ph.D. Thesis, Department of Computer Engineering & Informatics, University of Patras, Hellas.

Kameas, A. D. & Pintelas, P. E. (1997). The Functional Architecture and Interaction Model of a GENerator of Intelligent TutORing Applications, *Journal on Systems and Software,* 36 (3) 233-245, ISSN: 0164-1212.

Kieras, D. E., & Polson, P. G. (1985). An Approach to the Formal Analysis of User Complexity, *International Journal of Man-Machine Studies*, 22 (4) 365-394, ISSN: 0020-7373.

Kinny, D., Georgeff, M. & Rao, A. (1996). A Methodology and Modelling Technique for Systems of BDI Agents, *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW 96)*, pp. 56-71, ISBN:3-540-60852-4, Eindhoven, The Netherlands, January 22 – 25, 1996, Springer-Verlag New York.

Konolige, K. (1986). *A Deduction Model of Belief*, Morgan Kaufmann Publishers, ISBN: 0-934-61308-7, San Francisco, CA.

Miller, G. A., Galanter, G. & Pribram, K. H. (1960). *Plans and the Structured Behaviour*, Holt, Rinehart and Winston, ISBN 0-937-43100-1, New York.

Murata, T. (1989). Petri Nets: Properties, Analysis and Applications, in: *Proceedings of the IEEE*, 77 (4) 541-580 ISSN: 0018-9219.

Newell, A. & Simon, H. A. (1972). *Human Problem Solving*, Prentice-Hall, ISBN 0-134-45403-0, Upper Saddle River, NJ.

Norman, D. A. (1987). Cognitive Engineering – Cognitive Science, in: J. M. Carroll, (Ed.), *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, 325-336, The MIT Press, ISBN 0-262-03125-6 Cambridge, CA.

Rao, A. S. & Georgeff, M. P. (1991a). Modeling Rational Agents within a BDI-Architecture, in: *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, R. Fikes and E. Sandewall, (Eds.), 473-484, Morgan Kaufmann Publishers: San Mateo, CA.

Rao, A. S. & Georgeff, M. P. (1991b). Asymmetry Thesis and Side-Effect Problems in Linear-Time and Branching-Time Intention Logics, In: *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 498-504, ISBN: 1-55860-160-0, Sydney, Australia, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Rao, A. S. & Georgeff, M. P. (1995). Formal Models and Decision Procedures for Multi-Agent Systems, *Technical Report 61*, Australian Artificial Intelligence Institute, Melbourne, Australia.

Russell, S. & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*, 2nd edition, Prentice Hall Inc, ISBN: 0-13-103805-2, Upper Saddle River, NJ.

Schmidt, H. W. (1991). Prototyping and Analysis of Non-Sequential Systems Using Predicate-Event Nets, *Journal on Systems and Software*, 15 (1) 43-62, ISSN: 0164-1212.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*, Cambridge University Press, ISBN: 0-521-09626-X, Cambridge, England.

Shoham, Y. (1993). Agent-Oriented Programming, *Artificial Intelligence*, 60 51-92, ISSN: 0004-3702.

Wooldridge, M. (1992). *The Logical Modelling of Computational Multi-Agent Systems*. Ph.D. Thesis, UMIST, Manchester, UK.

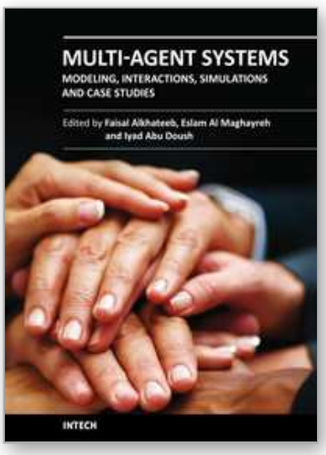
Wooldridge, M. & Jennings, N. R. (1995). Intelligent Agents: Theory and Practice, *The Knowledge Engineering Review* 10 (2) 115-152, ISSN: 0269-8889.

Wooldridge, M. (1996). Temporal Belief Logics for Modelling Distributed AI Systems, In: *Foundations of DAI*, G. M. P. O'Hare and N. R. Jennings, (Eds.), 269-286, Wiley Interscience, ISBN 0-471-00675-0, New York.

Wooldridge, M. (1999). Intelligent Agents, In: *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, G. Weiss (ed.), 27-77, The MIT Press, ISBN 0-262-23203-0, Cambridge, MA.

Zaharakis, I. D., Diplas, C. N., Kameas, A. D. & Pintelas, P. E. (1995). Specifying the Interaction with an Expert System Shell Using Interactive Multi Flow Graphs, *Proceedings of the 5th Hellenic Conference of Informatics*, pp. 601-613, Athens, Greece.

Zaharakis, I. D., Kameas A. D. & Nikiforidis, G. C. (1998). A Multi-agent Architecture for Teaching Dermatology, *Medical Informatics*, 23 (4) 289-307, ISSN: 1386-5056.



Multi-Agent Systems - Modeling, Interactions, Simulations and Case Studies

Edited by Dr. Faisal Alkhateeb

ISBN 978-953-307-176-3
Hard cover, 502 pages
Publisher InTech
Published online 01, April, 2011
Published in print edition April, 2011

A multi-agent system (MAS) is a system composed of multiple interacting intelligent agents. Multi-agent systems can be used to solve problems which are difficult or impossible for an individual agent or monolithic system to solve. Agent systems are open and extensible systems that allow for the deployment of autonomous and proactive software components. Multi-agent systems have been brought up and used in several application domains.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

I. D. Zaharakis (2011). The Agent Oriented Multi Flow Graphs Specification Model, Multi-Agent Systems - Modeling, Interactions, Simulations and Case Studies, Dr. Faisal Alkhateeb (Ed.), ISBN: 978-953-307-176-3, InTech, Available from: http://www.intechopen.com/books/multi-agent-systems-modeling-interactions-simulations-and-case-studies/the-agent-oriented-multi-flow-graphs-specification-model

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen