Goal Representation for BDI Agent Systems

Lars Braubach[1], Alexander Pokahr[1], Daniel Moldt[2], and Winfried Lamersdorf[1]

[1] Distributed Systems and Information Systems
Computer Science Department, University of Hamburg
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
{braubach, pokahr, lamersd}@informatik.uni-hamburg.de
[2] Theoretical Foundations of Computer Science
Computer Science Department, University of Hamburg
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
moldt@informatik.uni-hamburg.de

Abstract. Agent-oriented system development aims to simplify the construction of complex systems by introducing a natural abstraction layer on top of the object-oriented paradigm composed of autonomous interacting actors. One main advantage of the agent metaphor is that an agent can be described similar to the characteristics of the human mind consisting of several interrelated concepts which constitute the internal agent structure. General consensus exists that the Belief-Desire-Intention (BDI) model is well suited for describing an agent's mental state. The desires (goals) of an agent represent its motivational stance and are the main source for the agent's actions. Therefore, the representation and handling of goals play a central role in goal-oriented requirements analysis and modelling techniques. Nevertheless, currently available BDI agent platforms mostly abstract from goals and do not represent them explicitly. This leads to a gap between design and implementation with respect to the available concepts. In this paper a generic representation of goal types, properties, and lifecycles is developed in consideration of existing goal-oriented requirements engineering and modelling techniques. The objective of this proposal is to bridge the gap between agent specification and implementation of goals and is backed by experiences gained from developing a generic agent framework.

1 Introduction

When designing and building agent applications the developer is confronted with several intricate issues, ranging from general aspects such as development processes and tools to concrete design decisions like how agents should act and interact to implement a certain application functionality. These issues are addressed in the Jadex research project,[1] which aims to provide technical and conceptual support for the development of open multi-agent systems composed of rational

[1] http://vsis-www.informatik.uni-hamburg.de/projects/jadex

R.H. Bordini et al. (Eds.): PROMAS 2004, LNAI 3346, pp. 44–65, 2005.

and social agents. One main topic of the project is reviewing and extending concepts and software frameworks for developing goal-directed agents following the BDI model. With respect to goals in agent systems the topic poses several interesting questions, which can be categorised into representational, processing, and deliberation related issues.[2]

Representation:

1. *Which generic goal types and properties do exist?*
2. *Which goal states do exist during a goal's lifetime?*
3. Which structures can be used to represent goal relationships?

Processing:

4. *How does an agent create new goals and when does it drop existing ones?*
5. How does an agent reason and act to achieve its goals?
6. Which mechanisms do exist to delegate goals to other agents?

Deliberation:

7. *What are the possible agent's attitudes towards its goals?*
8. How can an agent deliberate on its (possibly conflicting) goals to decide which ones shall be pursued?

In the following the meaning of these questions will be shortly sketched. Regarding the representational aspect it is of interest which classifications of goals exist and which generic types of goals can be deduced from the literature and from implemented systems. Additionally, it is relevant which properties are exhibited by goals in general and specific goal types in particular. The second question refers to the goal lifecycle regarding the fact that goals can be in different states from the agent's point of view. On the one hand goals may differ in the agent's attitude towards them (see also question seven). This means that an agent e.g. sees some of its goals merely as possible options, which are currently not pursued in favour of other goals, and sees others as active goals, which it currently tries to achieve. On the other hand the goals may expose different processing states with respect to their type and achievement state. The third point focuses on the relationships between goals themselves, and between goals and other concepts. Relationships between goals are used for goal refinement purposes and for deliberation issues by making explicit how one goal (positively or negatively) contributes to another goal. The relationships to other concepts mainly influence creation and processing of goals, as discussed by the next two questions.

The aspect of goal processing comprises all mechanisms for goal handling during execution time. The initial question is how an agent comes to its goals and in what situations it may drop existing goals [11, 19, 23]. Intimately connected with this issue are deliberation aspects like the goal and intention commitment strategies, which define the degree of reconsideration an agent exposes. Extensive considerations about different intention commitment strategies can be found in descriptions of the IRMA agent architecture [6, 27]. Secondly, it is of importance which mechanisms an agent can use to try to achieve its goals. The process of

[2] This paper focuses on the *emphasised* questions.

plan selection and execution is a key element of BDI architectures and requires addressing further questions: How can the applicable plans be determined? Shall applicable plans be executed in parallel or one at a time? What mechanisms shall be used for the meta-level reasoning to select a plan for execution from the set of applicable plans? Partly, these questions are answered by proposed BDI architectures [6, 29] and by implemented systems [15, 16]. A complete discussion about the problems of this topic can be found in [8]. An important point of plan execution is that the agent should be able to recover from plan failures and have the possibility to try other means to achieve the goal it has itself committed to. Hence, a declarative goal representation would help to decouple plan from goal success resp. failure [37]. Another interesting point concerns goals in multi-agent systems (MAS) e.g. how an agent can delegate tasks to other agents. Goal delegation is one possibility of how this can be achieved. The topic has to address, besides the semantic meaning of goal delegation, issues of commitment, trust, and organisational structures [2, 20, 31].

Goal deliberation is part of the whole deliberation process, which comprises all meta-operations on the agent's attitudes such as belief revision and intention reconsideration. It is concerned with the manipulation of the goal structure of an agent, i.e. goal deliberation has the task to decide which goals an agent actively pursues, which ones it delays, and which ones it abandons. Necessary requirement for a goal deliberation mechanism to work is that the agent's attitudes towards its goals are clearly defined. Currently no general consensus exists how goal deliberation can be carried out. Instead, several approaches exist that address the topic with different strategies. The agent language 3APL introduces meta-rules for all of the agent's attitudes, which are executed during the interpreter cycle [10]. In contrast to this rule-based approach KAOS and Tropos allow the direct specification of contribution relationships between goals which form a basis for the decision process [9, 14]. In [33, 34] a mechanism based on pre- and post conditions for plans and goals is proposed and evaluated.

Considering these questions it is rather astonishing that available BDI multi-agent platforms such as JACK [15], JAM [16], or Jason [4] do not use explicit goal representations and therefore cannot address most of the aforementioned topics. One reason for this shortcoming is that most actual systems are natural successors of the first generation BDI systems (PRS [17, 13] derivates), which had to concentrate on performance issues and do without computationally expensive deliberation processes due to scarce computational resources. Additionally, the actual systems are mostly based on formal agent languages like AgentSpeak(L) [28] which focus on the procedural aspects of goals and treat them in an event-based fashion.

Nevertheless, the need for explicit goal representation is expressed in several recent publications [36, 37] and is additionally supported by the classic BDI theory, which treats desires (possibly conflicting goals) as one core concept [5]. The importance of explicit and declarative goal representation in the modelling area is underlined by BDI agent methodologies like Prometheus [24], Tropos [14] and requirements engineering techniques like KAOS [9, 18]. Additionally,

Winikoff et al. state in [37] "[...] by omitting the declarative aspect of goals the ability to reason about goals is lost", what means that the representation of goals is a necessary precondition when one wants reasoning about goals to become possible. Therefore, we claim that the usage of explicit goals should be extended from analysis- and design- to the implementation-level. Additionally, we think that this representation issue can be generalised and that one main objective of agent-oriented software development should be to support the continuity of concepts during the requirements, analysis, design, and implementation phase. This allows preserving the original abstraction level as far as possible throughout the development phases [21].

In this paper mainly generic goal representation issues for agent-oriented programming will be discussed with respect to the existing approaches coming from the requirements engineering and modelling area and from implemented systems. In the next section an example scenario is presented. Thereafter a generic model and lifecycle for goals is proposed and validated with respect to the given scenario in section 3. The model is elaborated further on to derive more specific goal types and representations. In section 4 the implementation of the proposed goal concepts for the Jadex agent system is sketched and finally, it is shown in section 5 that the concepts are well suited to be used in practical implementations by demonstrating how the example scenario can be realised. A summary and an outlook conclude the paper.

2 Example Scenario

In this section, a derivation of the so-called "cleaner world" scenario is described. It is based on the idea that an autonomous cleaning robot has the task to clean up dirt in some environment. This basic idea can be refined with respect to various aspects and already forms the foundation for several discussions about different agent and artificial intelligence topics (e.g. in [3, 12, 28, 30]).

In our scenario of the cleaner world the main system objectives are to keep clean a building at day, e.g. a museum, and to guard the building at night. To be more concise we think of a group of cleaning robots that are located in the building and try to accomplish the overall system goals by pursuing their own goals in coordination with other individuals. Therefore, four key goals for an individual cleaning robot were identified. First, it should clean its environment at day by removing dirt whenever possible. The cleaning robot therefore has to pick-up any garbage and carry it to a near waste bin. Secondly, it has to guard the building at night by performing patrols that should be based on varying routes. Any suspicious occurrences that it recognises during its patrols should be reported to some superordinated authority. Thirdly, it should keep operational by monitoring its internal states such as the charge state of its battery or recognised malfunctions. Whenever its battery state is low it has to move to the charging station. Fourthly, the robot should always be nice to other people that are close-by. This means that it should not collide with others and greet when this is appropriate.

These top-level goals of a cleaner agent can be further decomposed to more concrete subgoals. For example to clean up a piece of waste the robot first has to move to the waste and pick it up. Then it has to find a waste bin, move to the waste bin's location, and drop the waste into it. Similar refinements also apply to the other top-level goals.

3 Modelling Goals

The importance of goal representation is reflected through a variety of proposals for goal descriptions during the requirements acquisition, analysis, and design phases. In [35] three different kinds of goal criteria are stated that correspond to the distinctive features one would naturally deduce when considering a goal as a first class object; namely the object's type, the object's attributes, and the object's relations to other objects.

First characteristic is the goal type for which different taxonomies exist, which emphasise miscellaneous aspects. *System goals* represent high-level goals the software system needs to achieve to fulfil the system requirements and can be opposed to *individual goals* of single actors in the setting [9]. Another goal type distinction is made between so-called *hard* and *soft goals* [35]. Hard goals describe services the system is expected to deliver whereas soft goals refer to non-functional properties such as the expected system qualities like performance or excellence issues.

A very important classification relates to the *temporal behaviour of a goal* and additionally fits to the way in which humans tend to think and talk about goals. This classification is especially important for the design and implementation of agent based software, as it provides an abstraction for certain generic application behaviour. For example, a so-called *achievement goal* represents the common natural understanding of the word 'goal' as something to be achieved [9, 37]. In contrast, a *maintenance goal* is introduced to observe and maintain some world state as long as the goal exists [9].

Second characteristic of goals are their attributes that consist of properties relevant for all types of goals like name, description, priority, and other attributes that are type specific such as the target state specification for an achievement goal. Furthermore, goals can exhibit an arbitrary number of application specific attributes that are directly related to the problem domain like the desired location as part of a movement goal. Additionally, for implementing the Jadex BDI system several general goal properties were identified that are important for the interpretation of goals in the running system. Contrary to goals in natural language, which bear on a huge amount of implicit context and background knowledge, the semantics of executable goals, like the exclusion or retry mode for plan selection, has to be defined exactly [25].

Third characteristic of goals are their relationships to other objects, in first consequence to other goals. Such relationships between goals are typically hierarchical goal structures, which highlight refinement relationships with respect to the used refinement strategy. A common strategy used in several modelling

approaches are the AND/OR graphs [22]. An AND-refined goal demands that all its subgoals become satisfied while an OR-refined goal is fulfilled when at least one of the alternative subgoals is reached. An extensive discussion about goal relationships can be found in [35].

When talking about goals as objects it becomes apparent that they do not only exhibit these different characteristics, but additionally they need to be created in a suitable moment in the context of some actor to whom they belong. Only when new goal instances are generated during an agent's lifetime the agent will show rational behaviour in the sense that it proactively pursues its ideas [11]. And only when it exactly knows which goals actually exist and how the goals are interrelated, some deliberation mechanism can guide the agent to decide which goals should be pursued. We will now go on to discuss these issues with respect to the example scenario, thereby developing an explicit goal model and lifecycle.

3.1 Lifecycle

In the cleaner world scenario different goals can be identified for a cleaning agent. We will start our discussion with the cleanup-dirt goal, as it most closely matches the goal concepts commonly found in the literature. The desired behaviour of the agent is to pick up dirt whenever it sees it. This includes the statement of what to do (pick up dirt) and when to do it (sees dirt). Once the agent has achieved the goal, it can drop its intention towards it. To represent this goal in an agent application the developer should be able to specify in addition to the state to achieve, the condition (called production rule in [11]) when this goal should be created, therefore giving an answer to the question how an agent derives new goals.

When it notices some dirt in the environment and cannot clean-up the waste at the moment, e.g. because it already carries waste to the waste bin, it should be capable of memorising the new dirt positions to come back later and remove the litter. Hence, it should be able to form new still inactive clean-up goals (options) that should become active as soon as it is appropriate. Assuming that the environment changes during a time the agent cannot observe this area the agent might pursue a goal that is not appropriate any longer, e.g. some rubbish is blown away by the wind and the agent heads towards the memorised but outdated waste position. As soon as it can see the target position, it will notice that the waste has vanished and should drop the clean-up goal. Therefore, in addition to the conditions for goal creation, the representation of goals should allow the specification of the conditions under which a goal should be dropped.

In contrast to the cleanup-dirt goal, which is created and later dropped for each piece of waste, other goals (e.g. look for dirt, patrol) would be directly given to the agent when it is born and should persist during the lifetime of the agent. It can be noted that, although it is natural to say that the agent has both of these goals, only one of these goals is actively pursued depending on the daytime. Therefore, when representing such goals, the agent developer has to specify the context in which the goal should be pursued (e.g. day or night). Another thing that has to be captured by the goal representation is the fact that

when the agent sees some dirt it will form a new cleanup goal, which should be prioritised over the look-for-dirt goal. The agent should stop wandering around searching for dirt and cleanup the dirt it has found immediately. Therefore, the agent should be able to deliberate about its current goals to decide which one should be actively pursued and which ones should be dropped or inactivated (made to an option).

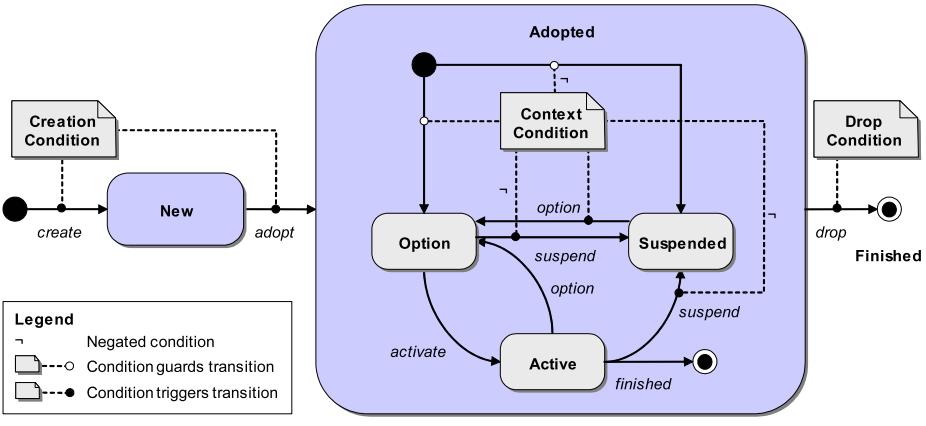

Fig. 1. Goal lifecycle

In Fig. 1 a proposal for a generic goal lifecycle that meets the requirements mentioned above is depicted in a UML statechart like fashion. It is shown that a goal can be in the states *New*, *Adopted* or *Finished*. The initial state of a newly created goal is *New*, what means that the goal exists as an idea but is not yet considered by the agent's deliberation mechanism. Therefore, the agent has to adopt the goal to pursue its new objective. By any means, the agent can always decide not to pursue the goal any more and drop it. The transitions between the different states can be either forced (not part of goal specification), e.g. a plan could create a new goal or drop a subgoal, or can be monitored by so-called conditions (specified as part of a goal). Conditions are annotated to several state transitions in two different ways to express either that the condition is used as a guard for the corresponding transition or that it represents the transition's trigger (see legend of Fig. 1). This means that a goal instance is created and adopted every time when the creation condition of this goal fires. Accordingly, it is dropped when its drop condition triggers.

Most interesting is the complex *Adopted* state which consists of the substates *Option*, *Active*, and *Suspended*. Adopting a goal makes this goal desirable to achieve for the agent and adds it to the agent's desire structure. The goal can be seen as an option that could possibly be pursued when the actual circumstances allow this. To be actively pursued the agent's deliberation mechanism has to activate the goal and so initiate the goal processing. The deliberation mechanism

can also deactivate the goal at any time by moving the goal to the option state again. Whenever the goal is an option or is active it can be suspended when the goal's context becomes invalid which is indicated by the goal's context condition. Here, a negation sign at the connection between condition and state transition indicates that the inverse of the condition is used as trigger for the transition. The suspension holds as long as the context stays invalid. A suspended goal is not actively pursued similar to an option, but in contrast to an option it cannot be activated by the deliberation mechanism due to its invalid context. When the context becomes valid again the goal is made an option to allow the deliberation component to reactivate the goal whenever appropriate.[3]

3.2 Types of Goals

As already mentioned, an important classification can be made with respect to the temporal behaviour of a goal. Unfortunately, there is no single exact set of suitable types of goals that can be used. Rather a multitude of different specifications and notions emerged from different sources such as methodologies or implemented systems (see Table 1).

Table 1. Several Different Goal Types

	KAOS	Gaia	JACK	PRS	JAM	Jadex
achieve	x	x	x	x	x	x
maintain	x	x		x	x	x
cease	x					
avoid	x					
optimise	x					
test			x	x		
query					x	x
perform					x	x
preserve			x	x		

The KAOS goal-oriented requirements engineering framework [9, 18] includes the already mentioned *achieve* and *maintain* goals. Additionally, KAOS introduces the negation of the aforementioned types called *cease* (as opposed to achieve) and *avoid* (as opposed to maintain). These two types of goals ease the description of goals in a natural way and semantically they can be traced back to the original types [35]. Furthermore, *optimise* goals for maximising or

[3] An interesting analogy to the goal lifecycle can be found in the operating systems area. The substates (*option*, *active*, *suspended*) of the adopted state resemble the states *ready*, *running*, *blocked* known from operating system processes [32]. Just like a blocked process, a suspended goal cannot be directly reactivated. In both cases a higher-level authority (the OS scheduler resp. the agent's deliberation mechanism) is responsible for selecting among the available options.

minimising some target value are proposed. The well-known Gaia methodology [38] does not introduce any goals at all, but uses liveness and safety properties for roles. Liveness properties describe states the agent has to bring about, whereas safety properties specify system invariants. In this way they are comparable with the achieve and maintain goal semantics.

The JACK agent system [15] offers in addition to *achieve* goal semantics the *test* and *preserve*[4] goal types. A test goal can be used to find out if a condition holds and a preserve goal is the passive version of a maintain goal in the sense that the goal controls a state and vanishes when this state is violated. In contrast to JACK, the C-PRS system [17] supports maintain goals at the implementation level. Besides *achieve* and *maintain* goals, the JAM interpreter [16] and the Jadex system [26] support *query* goals, which are similar to achieve goals. Query goals allow for an easy information retrieval from the beliefbase and when the result is not available the BDI mechanism will invoke plans for retrieving the needed information. The fourth type of goal that JAM and Jadex support is the *perform goal*, which is not related to some desired world state but to an activity. It ensures that an activity will be done in some future state [16].

In the rest of this paper we will concentrate on the *perform, achieve*, *query,* and *maintain* goal types. From Table1 one can see that the *achieve* and the *maintain* goal types are especially important, because they are in widespread use. *Cease* and *avoid*, on the other hand, exhibit the same execution semantics as *achieve* resp. *maintain*. The *optimise* goal belongs to the class of soft goals, which is outside the scope of this paper [35]. The *perform* goal is interesting, because it does not refer to a world state being achieved or maintained but to activities that should be performed. *Test* and *query* goals serve the same purpose to describe information acquisition, therefore only one of them is considered. Finally, The *preserve* construct is merely called a goal. In fact, it represents just a guarded action [17].

The following sections take a closer look at those interesting types of goals and their corresponding properties. Unlike the pure modelling approaches (KAOS, Gaia) it will be made explicit how goals following these models are processed at runtime using a BDI interpreter. One important aspect is therefore how their execution semantics relates to the generic goal lifecycle presented above. This is handled in a general way by the refinement of the active state, which reveals special information about the type of goal for goal processing. The example scenario is used as an evidence for the presented properties and behaviour, where appropriate.

Perform Goal. A *perform goal* specifies some activities to be done, therefore the outcome of the goal depends only on the fact if activities were performed [16]. Naturally, when no activities could be performed, e.g. because no plan was applicable in the actual context, the goal has *Failed*. Otherwise, when one or more plans have been executed the goal can enter the *Succeeded* state.

[4] It adds to the confusion about goal types that in JACK the *preserve* behaviour is obtained using the @maintain keyword.

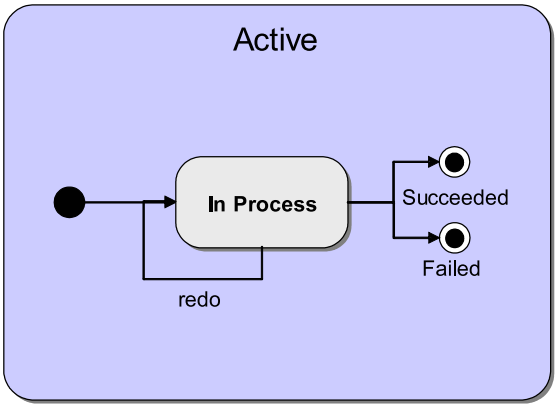

Fig. 2. Perform goal states

The refined active state of a perform goal is shown in Fig. 2. After being activated, the *In Process* state is entered, which triggers the internal plan selection and execution mechanism of the agent [26]. While plans are executing the goal stays in the *In Process* state. When the plan execution is done, i.e. no more plans are running or waiting for events, the *In Process* state is exited.

In the cleaner world example the two goals *patrol* and *do-greeting* should be modelled as perform goals, as they do not directly refer to a desired target state. While the *do-greeting* goal is finished once the greeting is performed, the *patrol* goal should not end when a patrol round is finished. Instead, the agent should continuously start new patrol rounds while the *patrol* goal is active. The *redo* property is an extension of the original JAM perform goal [16] and allows specifying that the activities of the goal should be performed iteratively. Therefore, when leaving the *In Process* state two state transitions may occur depending on the *redo* property. When *redo* is specified the goal re-enters the *In Process* state to re-start plan execution. When *redo* is not specified the goal enters one of the end states (*Failed*, *Succeeded*) causing the *Active* state to end. Looking back at the generic goal lifecycle (Fig. 1) one can see that exiting the *Active* state also causes the *Adopted* state to end (*finished* transition). Therefore, once the processing of the perform goal has stopped the goal is no longer adopted by the agent, because it is already reached or failed.

Achieve Goal. An *achieve goal* represents a goal in the classical sense by specifying what kind of world state an agent wants to bring about in the future. This target state is represented by a *target condition*. When an agent obtains a new achieve goal that shall be pursued (e.g. a cleanup goal) the agent starts activities for achieving the target state (e.g. no waste at given location). When the target state is already reached before anything has been done the goal can be considered as succeeded. Otherwise, for a yet unachieved goal the BDI mechanism is started and plans are selected for execution. Whenever during the plan execution phase the target condition switches to success all running plans

of that goal can be aborted and the goal is reached. In [37] the description of an achieve goal is enriched with an additional failure condition, which helps to terminate the goal when it is absolutely not achievable any more. The difference to the drop condition introduced in the generic goal lifecycle is that the drop condition does not determine the final state of the goal. In contrast, the failure condition indicates that the agent is unable to achieve the goal and therefore the goal has failed.

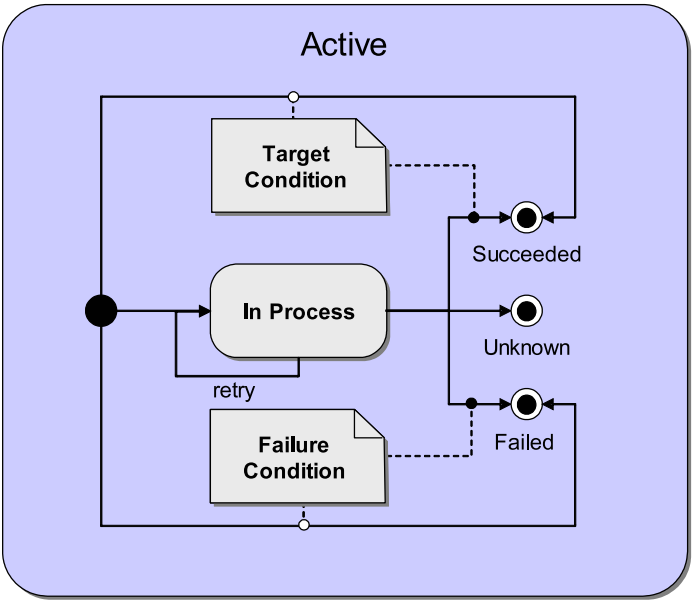

Fig. 3. Achieve / Query goal states

Fig. 3 shows the specific behaviour of an achieve goal. The main difference to the perform goal type is the target condition that specifies the desired world state to be achieved. An activated achieve goal will first check its target condition for fulfilment and enter the succeeded state directly when nothing needs to be done. Additionally, the failure condition will be checked to abort the goal when the condition is true. When none of them has fired the goal will enter the *In Process* state to start the execution of applicable plans. In contrast to the perform goal, plan execution may be terminated at any time when the target or failure condition become satisfied. In this case the goal is finished and moves to the *Succeeded* resp. *Failed* state.

When there are no more plans to execute and none of the executed plans could be completed successfully the goal moves to the *Failed* state. Another final state *Unknown* is entered when the execution is finished, some plans have been executed properly, but the agent cannot determine the truth-value of the

target condition (e.g. due to insufficient knowledge). Any of the three final states will cause the *finished* transition of the generic goal lifecycle (Fig. 1) to trigger. For example, when the given location is clean a cleanup goal is succeeded and can therefore be removed from agent's goal structure.

Query Goal. A *query goal* is used to enquire information about a specified issue. Therefore, the goal is used to retrieve a result for a query and does not necessarily cause the agent to engage in actions. When the agent has sufficient knowledge to answer the query the result is obtained instantly and the goal succeeds (e.g. an agent wants to find a waste bin and already knows the location). Otherwise, applicable plans will be tried to gather the needed information (e.g. searching for a waste bin).

The underlying model of the query goal resembles to a high degree the achieve goal [16]. The states of both goals are equal and are depicted in Fig. 3. Main difference between both goal types is that the query goal requires an informational result, which is captured by an implicit target condition testing if a result is available.

Maintain Goal. A *maintain goal* has the purpose to observe some desired world state and the agent actively tries to re-establish this state when it is violated. The perform, achieve, and query goal types represent goals that continuously cause the execution of plans while they are active. In contrast, an activated maintain goal may not instantly cause any plan to be executed. Fig. 4 shows that the maintain goal stays in the *Idle* state until the maintain condition is violated. Another difference is that there is no final state. Even when the *maintain condition* is currently satisfied the agent always has to monitor the environment for changes that may violate the condition. The maintain goal therefore always moves back to the *Idle* state when processing has been finished successfully.

In case the processing fails but the agent has no more applicable plan to execute, the *Unmaintainable* state is entered, which means that the agent knows that the condition is violated, but there is nothing it can do about it. Similar to the achieve goal, a maintain goal may be in the *Unknown* state when the agent cannot determine if the plan execution leads to the desired results. From the *Unknown* state a transition back to the *In Process* or *Idle* state may be done when the agent can determine the state of the maintain condition. From both the *Unknown* and the *Unmaintainable* state, the goal may periodically re-enter the *In Process* state to try out if the goal can be maintained now. This behaviour is obtained by specifying the *recur* flag. In contrast to the *retry* flag, which manages the sequential execution of applicable plans, the *recur* flag leads to a complete restart of goal processing, thereby again considering previously excluded plans.

Using the maintain condition alone may sometimes lead to undesirable behaviour, because of the event driven nature of goal processing in BDI agents. Consider the *maintain-battery-loaded* goal of the cleaner agent: When the condition to be maintained is specified as 'chargestate < 20%' the agent will move to

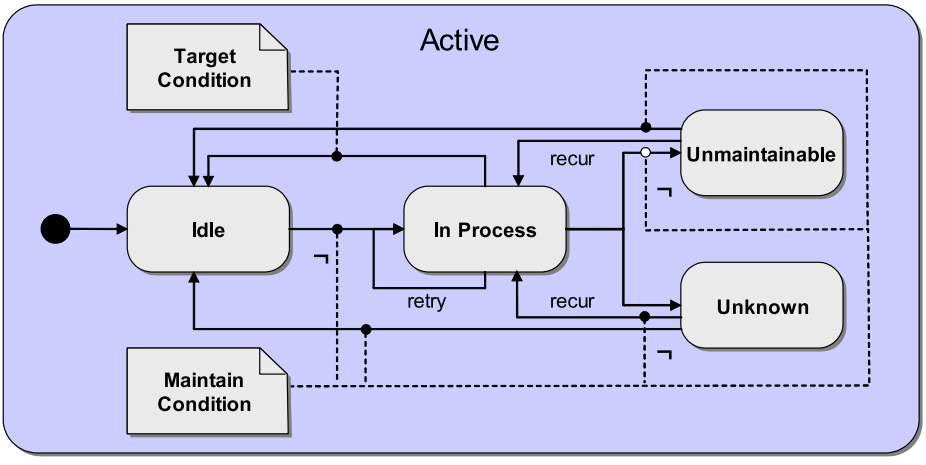

Fig. 4. Maintain goal states

the charge-station whenever the energy level drops below 20%. However, as soon as the level is back at 20% the agent will stop loading its battery, because the condition is satisfied again. Therefore, it is sometimes necessary to concretise the condition to be established whenever the maintain condition is triggered. In our model this can be specified by an optional *target condition*, which specifies when the transition to the idle state is allowed. The semantics of this extended type of maintain goal is therefore: Whenever the maintain condition is violated select and execute plans in order to establish the (more specific) target condition. In the example the maintain condition 'chargestate > 20%' can be refined to the target condition 'chargestate=100%' to make sure that the cleaner agent will always do a full recharge.

All of the specific types of goals (perform, achieve, query, maintain) inherit the same generic lifecycle presented in section 3.1. Therefore, in addition to the properties specific to a goal type (such as failure condition for achieve goals) the specification of any goal can be enriched by the generic goal properties such as creation, context, and drop condition. This makes it possible e.g. to easily specify a maintain goal that should only be pursued in a given context.

4 Goal Realisation in Jadex

The last section presented a generic model for goals in BDI agents and identified four goal types with distinct execution behaviour. In the following we will shortly sketch how this execution behaviour is realised in the generic agent framework Jadex. The next section will then show how applications like the cleaner example scenario can be easily implemented when such an abstract goal representation is available at the implementation level.

The Jadex agent framework [26, 7] is built on top of the JADE plattform [1] and provides an execution environment and an API to develop agents using

beliefs, goals, and plans as first class objects. Jadex adopts well established application development technologies such as XML, Java, and OQL to facilitate an easy transition from conventional object-oriented programming to BDI agent programming.

To implement an agent the developer has to create two types of files: One XML file is used to define the agent by declaratively specifying among other things the beliefs, goals, and available plans. In addition to this agent definition file (ADF), for each plan used by the agent the plan body has to be implemented in a separate Java class. Plan implementations may use the Jadex API e.g. to send messages, manipulate beliefs, or create subgoals (for details see [25]). An expression language is used throughout the ADF to establish the connection between the declarative elements in the ADF and the object-oriented plan implementations. The language follows a Java syntax, but is extended to support OQL constructs for querying the belief base.

The goal tags in the XML file are read by the interpreter to create instances of the goals, which implement the state machines presented in section 3. The instatiated goal objects themselves take care of their lifecycle by throwing so called goal-events (leading to the execution of plans) whenever they enter the *In Process* state and by automatically performing the corresponding state transitions when goal conditions are triggered or the execution of a plan has finished. The goal conditions and parameters, which are evaluated at runtime, are specified using the Java/OQL like expression language.

At runtime the system keeps track of the instantiated goals, which may be created either as independent top-level goals or dispatched as subgoals inside of a plan. Goal processing is initiated whenever the active state of a goal is entered. Before a goal is reached, several plans may try to process the goal, even at once, when specified so. Thereby, plans only have access to a copy of the original goal object called *process* goal, to ensure a level of isolation between running plans and their associated subgoal-hierarchies. When the active state of a goal is exited (e.g. because the goal is suspended), all associated process goals are dropped leading to a termination of the corresponding plans and subgoals created by those plans. For each goal, a history of process goals is kept to remember the executed plans together with the outcome. This information is used to determine plans which should be excluded from the applicable plan list, when the goal needs to be processed again.

5 Example Implementation

The cleaner world scenario is realised as a simulation setting using two different kinds of agents. Besides the cleaner agents an environment agent acts as substitute for the real surrounding. Using an agent as environmental representation has the advantage that the setting can be easily distributed over a network of computers having cleaner agents working in the same environment located on different platforms.

The cleaner agents use vision and movement plans that interact with the environment agent following a domain dependent ontolology in which the relevant concepts and actions like waste, waste bin and pick-up resp. drop waste are defined. They update their internal beliefs with respect to the sensed environmental changes and request actions in the environment that may fail under certain conditions e.g. when two cleaners try to pick up the same piece of waste simultaneously.

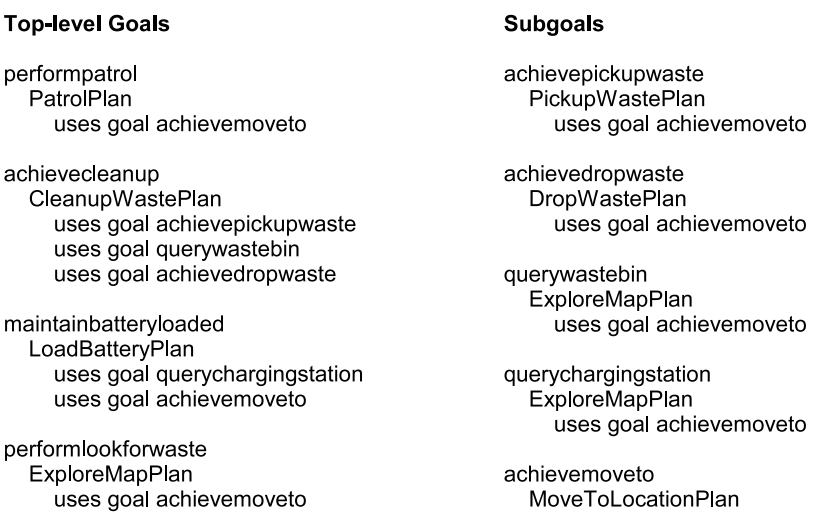

Fig. 5. Goal - plan overview

In Fig. 5 a brief overview of the relationships between the used goals and plans is given. On the left hand side the agents' top-level goals are shown whereas on the right hand side the subgoals that are used from within plans are depicted. For each goal at least one plan is defined that is responsible for pursuing the goal. As introduced in section 2 a cleaner agent has top-level goals for performing patrols (performpatrol), cleaning-up waste (achievecleanup) and monitoring its battery state (maintainbatteryloaded). To avoid the agent doing nothing when it currently has no duty, a goal template for searching for waste is also defined (performlookforwaste).

To handle the performpatrol goal a cleaner agent has a patrol plan that accesses a predefined route from the beliefbase and steers the agent to the actual patrol points by using the achievmoveto subgoal. Somewhat more complex is the CleanupWastePlan that is used in response to an active achievecleanup goal. It employs three different subgoals for decomposing the goal into the separate tasks of picking up a piece of waste (achievepickup), searching for a non-full waste bin (querywastebin) and finally dropping the waste into the wastebin (achievedropwaste). To be able to resume a suspended cleanup goal the plan also tests if the agent is already carrying a piece of waste. In the case that the

agent already possesses the waste the pickup procedure can be omitted. For re-establishing a violated maintainbatteryloaded goal the LoadBatteryPlan tries to find a charging station, heads towards it and consumes as many energy as needed. To find a suitable station a query subgoal (querychargingstation) is used that immediately returns a result when the agent already knows a station. When this is not the case, the ExploreMapPlan is used to systematically search for a yet unknown charging station. This plan is also used in the context of the performlookforwaste goal to discover new waste in the environment.

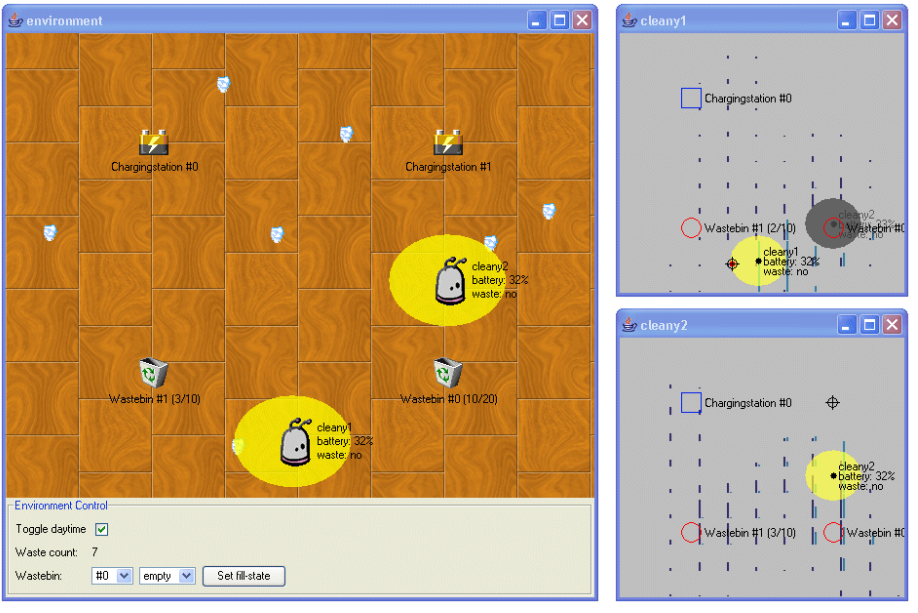

Fig. 6. Cleaner World Example Snapshot

A cleaner agent has three initial goal instances that drive its actions from birth. An instance of the performlookforwaste resp. the performpatrol goal lets the agent move around to search for waste or to observe the environment, depending on the daytime. These two goals are only active, when the agent has no other important things to do. An instance of the maintainbatteryloaded has highest priority and monitors the agent's battery state during its lifetime. In addition, several goal types are declared for goals that get instantiated and adopted under certain conditions. In the following sections some example goal declarations are explained. More implementation details can be found in the freely downloadable Jadex package, which includes a runnable implementation of the cleaner world example.[5] In Fig. 6 a snapshot of the running application is presented, which shows the global environmental view as well as the local views of two cleaner agents.

[5] available for download at `http://vsis-www.informatik.uni-hamburg.de/projects/jadex`

5.1 The Perform-Patrol Goal

Fig. 7 shows the perform-patrol goal as it is specified in the XML agent descriptor of a cleaner agent. The goal is of type *performgoal* and is given the name perform-patrol. The attribute *redo* was already introduced in the refined perform goal state chart (see Fig. 2) and causes the goal to be continuously executed as long as applicable plans are available. The *exclude* attribute is a special flag that in this case tells the BDI plan selection mechanism that plans should not be excluded from the applicable plans list once they have been executed. Therefore, the agent will continue to patrol while the goal is active using any patrol plans it has.

```
<performgoal name="performpatrol" redo="true" exclude="never">
   <contextcondition>
      !$beliefbase.is_loading && !$beliefbase.daytime
   </contextcondition>
</performgoal>
```

Fig. 7. Peform-patrol goal

The example scenario demands that the agent should only be on patrol at night. Our system does not yet capture the (positive or negative) contribution between goals, but the agent has to be prevented somehow from continuing to patrol while it tries to reload its battery. It is assumed that the agent knows if it is day or night and if its battery state is low and has to be reloaded. Using these two boolean beliefs (daytime, is_loading) the developer can specify the *contextcondition* of the goal, where $beliefbase refers to the belief base of the agent. The context condition was introduced in the generic goal lifecycle (Fig. 1) and defines when the goal can or cannot be active. The perform patrol goal may therefore only be active when the agent is not loading its battery and it is not daytime. In a similar way, a perform-look-for-waste goal is defined with a context condition that is only valid at daytime.

5.2 The Achieve-Cleanup Goal

One purpose of the cleaner agent is to remove all pieces of waste it notices. The achieve-cleanup goal (Fig. 8) is an *achievegoal* that is instantiated for every single piece of waste to clean up. The goal contains a *parameter* waste specifying which piece of waste to clean up. The given default *value* of the waste parameter is specified by a select statement that always evaluates to the piece of waste that is nearest to the agent when the goal is instantiated. The known pieces of waste (belief wastes) are sorted by distance (order by clause) to the current location (belief my_location).

For the agent to keep cleaning up every piece of waste it notices, the *creation-condition* as introduced in the generic goal lifecycle (Fig. 1) is used to trigger creation of new goal instances whenever needed. A cleanup goal will be created whenever the agent knows that there is some waste (belief wastes) and that it is not currently cleaning (belief is_cleaning). The reason for the second part of the

```
<achievegoal name="achievecleanup">
  <parameter name="waste" class="Waste">
    <value>
      select any $waste from $beliefbase.wastes order by
      $waste.location.getDistance($beliefbase.my_location)
    </value>
  </parameter>
  <creationcondition>
    $beliefbase.wastes.length>0 && !$beliefbase.is_cleaning
  </creationcondition>
  <contextcondition>
    !$beliefbase.is_loading && $beliefbase.daytime
  </contextcondition>
  <dropcondition>
    !$beliefbase.carrieswaste
    && (!$beliefbase.containsFact("wastes", $goal.waste)
    || (select any $waste from $beliefbase.wastes
      order by $waste.location.getDistance(
        $beliefbase.my_location)) != $goal.waste)
  </dropcondition>
  <targetcondition>
    (select any $wastebin from $beliefbase.wastebins
    where $wastebin.contains($goal.getParameter("waste"))) !=null
  </targetcondition>
</achievegoal>
```

Fig. 8. Achieve-cleanup goal

condition is that there is currently no deliberation mechanism telling the agent which cleanup goal to achieve first when there is more than one present at the same time. Therefore, the is_cleaning belief is used to assure that only one cleanup goal at a time is created. As with the perform-patrol goal a context condition is used to constrain under which circumstances the goal may be active: The agent should pursue cleanup goals only when it is not loading its battery and only at daytime. The goal is achieved when the waste is contained in one of the known waste bins as described in the *target condition*.

In our example implementation we also added a rather complex *dropcondition* for the cleanup goal, which is not necessary for correct operation, but helps to improve the performance of the cleaner agent. To allow opportunistic cleanup of new pieces of waste and to avoid unnecessary movement of the cleaner, an existing cleanup goal is dropped when the agent comes to know that the piece of waste to be picked up is no longer there or another piece of waste is closer to the agent.

5.3 The Query-Wastebin Goal

The query-wastebin goal shows how a goal to query for information can be realised in Jadex (see Fig. 9). Assuming that the agent does not completely know its environment, the objective of the goal is to find a waste bin that is not

full and near to the agent. This goal is created by a plan as a subgoal of the achieve-cleanup goal once the agent has picked up some dirt (cf. sect. 2).

```
<querygoal name="querywastebin" exclude="never">
  <parameter name="result" class="Wastebin" optional="true">
    <value evaluationmode="on_demand">
      select any $wastebin from $beliefbase.wastebins
      where !$wastebin.isFull() order by
      $waste.location.getDistance($beliefbase.my_location)
    </value>
  </parameter>
</querygoal>
```

Fig. 9. Query-wastebin goal

It is modelled as *querygoal* and has a *parameter* result. This parameter is bound to the nearest not full waste bin, if any, and is evaluated on_demand what means that the select expression is evaluated whenever the parameter value is accessed. The *targetcondition* of the query goal is not stated and therefore the default target condition for query goals is used. Hence the goal succeeds when a result is retrieved, i.e. a not full waste bin nearby was found. The implicit target condition allows for opportunistic goal achievement (see Fig. 3), that is, the goal succeeds without the execution of any plan if the agent already knows the location of a not full waste bin.

```
<maintaingoal name="maintainbatteryloaded">
  <maintaincondition>
    $beliefbase.my_chargestate > 0.2
  </maintaincondition>
  <targetcondition>
    $beliefbase.my_chargestate == 1.0
  </targetcondition>
</maintaingoal>
```

Fig. 10. Maintain-battery-loaded goal

5.4 The Maintain-Battery-Loaded Goal

The cleaning agent has to stay operational; therefore it has to monitor its internal state and will occasionally move to the charging station to reload its battery. The specification of the maintain-battery-loaded goal is given in Fig. 10. The goal is a *maintaingoal* and therefore includes a *maintaincondition* and a *targetcondition* as present in the refined maintain goal state chart (see Fig. 4). The maintain condition monitors the battery state (belief my_chargestate) and

triggers plan execution whenever the charge state drops below 20%. The refined target condition causes the battery to be always reloaded to 100% before the goal moves back to the idle state.

6 Conclusions and Outlook

This paper provides two main contributions. First, the way of how an agent attains and manages its goals is analysed and a generic lifecycle is proposed that models the different states of goals in BDI agent systems. Secondly, the generic goal lifecycle is refined into different goal types which capture commonly required agent behaviour. Both of these contributions are backed by the cleaner world example at the conceptual as well as implementation level.

The example shows that the proposed goal model is well suited for a natural description of an agent-based system. The continuous usage of abstract concepts in the design and implementation phases considerably simplifies the development of software agents compared to the current practice of using object-oriented techniques. Additionally, it helps to preserve the abstraction level throughout the whole development process. The system is easier to design, as the involved goal concepts are closer to the way that humans think and act. The transition to the implemented system is largely simplified, because only minor refinements of design specifications are necessary to obtain an executable system. Moreover, the development is less error-prone, as large portions of complex agent behaviour, such as goal creation and processing, are already implemented in the underlying agent architecture. Finally, the types of goals available in the agent language have the additional effect that they may guide the agent developer in its analysis and design decisions, because they represent a natural and abstract means for describing the application domain.

This work is also the result of practical considerations when realising the proposed goal model in an efficient and easy to use software framework. The model includes those goal types and properties that frequently occured in the researched systems and methodologies and that have practical relevance for agent systems we have built so far.

The presented goal model does not cover all important aspects of goals as they are presented in the introduction. One point that was not addressed by this paper affects the relations between goals such as hierarchies for goal decomposition. In this field, especially concerning the requirements and modelling phases, a lot of research has already been done and it has to be evaluated if these concepts can be successfully transferred to the design and implementation phase of MAS. Another important aspect of goals that was covered only marginally in this paper is goal deliberation. With the help of deliberation mechanisms, the agent is able to select between different goals, detect goal conflicts and handle them appropriately. The precondition for goal deliberation is the explicit and declarative representation of goals, which is not reflected in actual agent systems and agent languages. Therefore, the conceptualization of the introduced goal model is the foundation for further explorations of different deliberation mechanisms.

References

1. F. Bellifemine, G. Rimassa, and A. Poggi. JADE – A FIPA-compliant agent framework. In *4th Int. Conf. Practical Applications of Agents and Multi-Agent Systems (PAAM-99)*, pages 97–108, London, UK, December 1999.
2. F. Bergenti, L. Botelho, G. Rimassa, and M. Somacher. A FIPA compliant Goal Delegation Protocol. In *Workshop on Agent Communication Languages (AAMAS 2002)*, 2002.
3. R. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifiable multi-agent programs. In *Proceedings of the First International Workshop ProMAS*, pages 43–49, Australia, 2003.
4. R. H. Bordini and J. F. Hübner. *Jason User Guide*, 2004. `http://jason.sourceforge.net/`.
5. M. Bratman. *Intention, Plans, and Practical Reason.* Harvard University Press, 1987.
6. M. Bratman, D. Israel, and M. Pollack. Plans and Resource-Bounded Practical Reasoning. In *Philosophy and AI: Essays at the Interface*, pages 1–22. The MIT Press, 1991.
7. L. Braubach, A. Pokahr, and W. Lamersdorf. Jadex: A Short Overview. In *Net.ObjectDays 2004: AgentExpo*, 2004. (to be published). `http://vsis-www.informatik.uni-hamburg.de/papers/jadex_node.pdf`.
8. P. Busetta, N. Howden, R. Rönnquist, and A. Hodgson. Structuring BDI Agents in Functional Clusters. In *Intelligent Agents VI, ATAL '99*. Springer, 2000.
9. A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, April 1993.
10. M. Dastani, F. de Boer, F. Dignum, and J.J. Meyer. Programming Agent Deliberation: An Approach Illustrated Using the 3APL Language. In *Proceedings of AAMAS'03*, 2003.
11. Frank Dignum and Rosaria Conte. Intentional Agents and Goal Formation. In *Agent Theories, Architectures, and Languages*, pages 231–243, 1997.
12. J. Firby. An Architecture for A Synthetic Vacuum Cleaner. In *Proc. of the AAAI Fall Symp. Series Workshop on Instantiating Real-World Agents*, Raleigh, NC, October 1993.
13. M. Georgeff and A. Lansky. Reactive Reasoning and Planning: An Experiment With a Mobile Robot. In *Proceedings of the 1987 National Conference on Artificial Intelligence (AAAI 87)*, pages 677–682, Seattle, Washington, July 1987.
14. F. Giunchiglia, J. Mylopoulos, and A. Perini. The Tropos Software Development Methodology: Processes, Models and Diagrams. In *Proc. of AAMAS02*. ACM Press, 2002.
15. N. Howden, R. Ronnquist, A. Hodgson, and A. Lucas. JACK Intelligent Agents - Summary of an Agent Infrastructure. In *Proc. 5th ACM Int. Conf. on Autonomous Agents*, 2001.
16. M. Huber. JAM: A BDI-Theoretic Mobile Agent Architecture. In *3rd Annual Conf. on Autonomous Agents (AGENTS-99)*, pages 236–243, New York, May 1–5 1999. ACM Press.
17. F. Ingrand, R. Chatila, R. Alami, and F. Robert. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 43–49, Minneapolis, April 1996.
18. E. Letier and A. van Lamsweerde. Deriving operational software specifications from system goals. In *Proc.of the 10th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 119–128. ACM Press, 2002.

19. M. Luck and M. d'Inverno. Motivated Behaviour for Goal Adoption. In *Multi-Agent Systems: Theories, Languages and Applications - 4th Australian Workshop on Distributed Artificial Intelligence*, pages 58–73. Springer-Verlag, 1998.
20. Á. Moreira, R. Vieira, and R. Bordini. Extending the operational semantics of a BDI agent-oriented programming language for introducing speech-act based communication. In *Proc. Declarative Agent Languages and Technologies (DALT-03), held with AAMAS-03*, 2003.
21. J. Mylopoulos. Requirements-Driven Information Systems Development. Invited Talk, AOIS'99 at CAiSE'99, Heidelberg, Germany, 1999.
22. N. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, 1971.
23. T. J. Norman and D. Long. Goal creation in motivated agents. In *Intelligent Agents, Proc. of ATAL'95*, pages 277–290. Springer-Verlag, 1995.
24. L. Padgham and M. Winikoff. Prometheus: A methodology for developing intelligent agents. In *3rd Int. Workshop on Agent Oriented Software Engineering (AOSE02)*, July 2002.
25. A. Pokahr and L. Braubach. *Jadex User Guide*, 2004. `http://vsis-www.informatik.uni-hamburg.de/projects/jadex/download.php`.
26. A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: Implementing a BDI-Infrastructure for JADE Agents. *EXP – in search of innovation*, 3(3):76–85, 2003.
27. M. Pollack. The Uses of Plans. *Artificial Intelligence*, 57(1):43–68, 1992.
28. A. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In *7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, 1996.
29. A. Rao and M. Georgeff. BDI Agents: from theory to practice. In *Proc. of the 1st Int. Conference on Multi-Agent Systems (ICMAS'95)*, pages 312–319. The MIT Press, 1995.
30. S. Russell and P. Norvig. *Artifical Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1995.
31. M. Somacher, M. Tomaiuolo, and P. Turci. Goal Delegation in Multiagent System. In *Proc. Tecniche di Intelligenza Artificiale per la ricerca di informazione sul Web, Siena*, 2002.
32. A. Tanenbaum. *Modern Operating Systems*. Prentice Hall PTR, 2001.
33. J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and Avoiding Interference Between Goals in Intelligent Agents. In *Proceedings of IJCAI 2003*, August 2003.
34. J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and Exploiting Positive Goal Interaction in Intelligent Agents. In *Proceedings of AAMAS'03*, 2003.
35. A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proc. RE'01 - Int. Joint Conference on Requirements Engineering*, pages 249–263. IEEE, 2001.
36. M. Winikoff, J. Harland, and L. Padgham. Linking Agent Concepts and Methodology with CAN. `http://citeseer.ist.psu.edu/497423.html`.
37. M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative & Procedural Goals in Intelligent Agent Systems. In *Proc. of KR03*. Morgan Kaufmann Publishers, 2002.
38. M. Wooldridge, N. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.