Goal-Oriented Agent Patterns with the PRACTIONIST Framework

V. Morreale[a] G. Francaviglia[a] F. Centineo[a] M. Puccio[a]

M. Cossentino[bc]

[a] *R&D Laboratory - ENGINEERING Ingegneria Informatica S.p.A., Italy*
[b] *SET-Université de Technologie Belfort - Montbéliard, France*
[c] *ICAR - Italian National Research Council*

Abstract

When developing BDI agent-based systems, some design patterns such as incompatible intentions, multiple strategies, intention decomposition, etc. would be very useful for specifying some desired agent behaviours. As BDI agent programmers, our desire would be to have a framework that natively supports such common patterns.

The PRACTIONIST framework provides a goal-oriented approach for developing agent systems according to the BDI model. In this paper we first describe the goal model of PRACTIONIST agents and how they use such a model to reason about goals during their deliberation process and means-ends reasoning. Then, we show how some useful BDI agent patterns can be directly and actually implemented with our framework, which natively supports such design-level solutions. In other words, in our framework we wanted to solve some common design problems, by providing some built-in solutions that programmers can easily adopt when developing their intentional agents.

1 Introduction

The BDI [12] is one of the most interesting agent models and derives from the philosophical tradition of the practical reasoning, which states that agents decide, moment by moment, which actions to perform in order to pursue their goals. The practical reasoning involves a deliberation process, to decide what states of affairs to achieve, and a means-ends reasoning, to decide how to achieve them.

When developing BDI agent-based systems, some common design problems are often addressed at several levels of abstraction (i.e. organization, agent, task and so forth). According to [14] design patterns are explicit formulations of good proven solutions to recurring problems that arise within some contexts. Thus patterns make easier the reuse of good software design. A design pattern explains the insight and good practices that have evolved to solve a given problem and it provides a concise definition of common elements, context, and essential requirements for a solution [8]. Moreover, patterns reduce the development time, communicate knowledge and can help people to learn a new design paradigm [13]. The adoption of design patterns also improves the quality of software, as they are validated by the experience rather than from testing [6].

Some common design patterns such as incompatible intentions, multiple strategy, intention decomposition, etc. would be very useful to specify some desired behaviours of intentional agents. Unfortunately, some BDI agent frameworks do not directly support such patterns and usually require too much work to actually implement them. As BDI agent programmers, our desire would be to have a framework that natively support such patterns.

We believe that the explicit representation of goals and the ability to reason about them play an important role in the definition of several intentional design patterns.

Therefore our PRACTIONIST framework [10] adopts a goal-oriented approach to develop BDI agents and stresses the separation between the deliberation process and the means-ends reasoning, with the abstraction of goal used to formally define both desires and intentions during the deliberation phase. In PRACTIONIST a goal is considered as an analysis, design, and implementation abstraction compliant to the semantics described in this paper. In other words, agents can be

programmed in terms of goals, which will be related to either desires or intentions according to whether some specific conditions are satisfied or not.

Actually, other BDI agent platforms natively use the concept of goal (e.g. JACK [2] and JAM [5] use goals instead of desires). However, the actual implementations of mental states differ somewhat from their original semantics: desires (or goals) are treated as event types (such as in AgentSpeak(L) [11]) or procedures (such as in 3APL [4]) and intentions are executing plans. Therefore the deliberation process and means-ends reasoning are not well separated, as being committed to an intention (ends) is the same as executing a plan (means). In our view, this reduces the opportunity of defining and implementing useful design patterns.

Moreover, since some available BDI agent platforms do not support the explicit representation and the implementation of goals or desires with their properties and relations, the ability to reason about goals is lost, along with the ability to know if goals are impossible, achieved, incompatible with other goals, and so forth [15]. Thus, the support for the *commitment strategies* of agents and their ability to autonomously drop, reconsider, replace or pursue goals is strongly decreased.

Only a few BDI agent platforms deal with declarative goals, such as JADEX [1], where goals are explicitly represented according to a generic model, enabling the agents to handle their life cycle and reasoning about them. Nevertheless, the model defined in JADEX does not deal with relations among goals.

In the following sections we show that some useful BDI agent patterns can directly and actually be implemented with the PRACTIONIST framework, which natively supports some design solutions. In other words, in our framework we wanted to solve some of most common design problems at the agent level, by providing some built-in solutions that programmers can easily adopt when developing their agents.

Therefore, in this paper we first give a brief overview of the general structure of PRACTIONIST agents and their execution model (section 2). Then we introduce the definition of the goal model and describe how PRACTIONIST agents are able to reason about available goals according to their goal model, current beliefs, desires, and intentions (section 3). Finally, some agent-level design patterns are presented and the implementation with the PRACTIONIST framework is briefly illustrated (section 4). The last section outlines some conclusions and our intentions about further and future work.

2 PRACTIONIST Agents

The PRACTIONIST framework, defined on top of JADE[1], supports the development of BDI agents endowed with the following elements (figure 1):

- a set of *perceptors* that listen to some relevant external stimuli (*perceptions*);
- a set of *beliefs* (represented through prolog-like clauses) that each agent has got about both its internal state and the external environment;
- a *goal model* including: *(i)* a set of *goals* each agent could pursue, which represent some states of affairs to bring about or activities to perform and can be related to either *desires* or *intentions*; and *(ii)* a set of *goal relations* to use during its deliberation process and means-ends reasoning;
- a set of *plans* that are the means to achieve the intentions;
- a set of *actions* an agent can perform to act over its environment; and
- a set of *effectors* that actually execute such actions.

In the PRACTIONIST framework plans represent an important container in which developers define the actual behaviours and strategies of agents. Each agent may own a declared set of plans (the *plan library*), each specifying the course of *acts* an agent can perform while pursuing its intentions, or to handle incoming perceptions, or to react to changes of its beliefs.

[1] http://jade.tilab.com

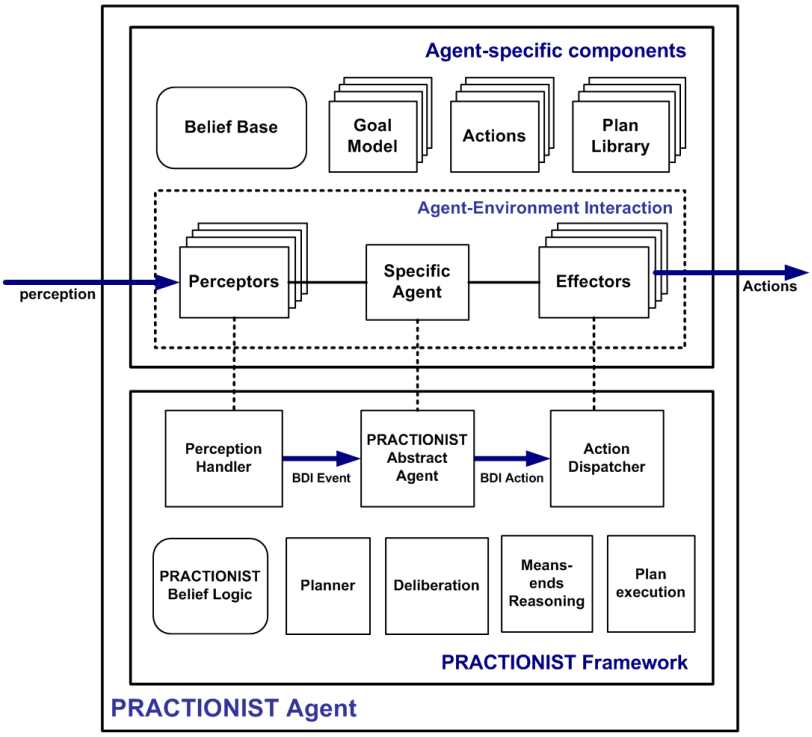

Figure 1: PRACTIONIST Agent Architecture.

PRACTIONIST plans have a set of properties, which agents can use during their means-ends reasoning and the actual execution of their activities. Some of these properties are: the *trigger event*, which defines the event (i.e. goals, perceptions, and belief updating) each plan is supposed to handle; the *context*, a set of conditions that must be believed true before performing the plan; the *body*, which include the acts the agent performs during the execution of the plan. Within the body several acts are possible, such as *sending* messages, *desiring* to bring about some states of affairs or perform some action, modifying beliefs, and so forth. Further details about the structure of PRACTIONIST agents and its components can be found in [10].

During its main cycle an agent performs the following steps:

1. through the perceptors, it searches for perceptions coming from the environment and transforms them into *events*, which in turn are put into an event queue;

2. it selects and extracts an event from such a queue, according to some logic;

3. it handles the selected event through the following *means-ends reasoning* process: *(i)* the agent figures out the *practical* plans, which are those plans whose trigger event matches the selected event; *(ii)* among practical plans, the agent detects the *applicable* ones, which are those plan whose context is believed true, and selects one of them (the *main plan*); *(iii)* it builds the *intended means*, containing the main plan and the other alternative practical plans. Each intented means is put within a stack according to the following criteria: if the event that has generated the intended means is related to an intention to pursue a goal, it is put on top of the stack in which there is the intended means that is committed to that intention; otherwise, a new stack is created with the new intented means.

It should be noted that every intended means stack can contain several intended means, each able to handle a given event, possibly through several alternative means. Moreover all intended means stacks are *concurrently* executed, so that each PRACTIONIST agent can perform several activities in parallel.

3 The PRACTIONIST Goal Model

In the PRACTIONIST framework, a goal is an objective that an agent could pursue and we use it as a means to transform desires into intentions through the satisfaction of some properties. In other words, our agents are programmed in terms of goals, which then will be related to either desires or intentions according to whether some specific conditions are satisfied or not.

More formally, a PRACTIONIST *goal* g is defined as a pair $g = \langle \sigma_g,\ \pi_g \rangle$, where σ_g is the *success condition* of the goal and π_g is the *possibility condition* stating whether g can be achieved or not. In the PRACTIONIST framework both conditions are local properties of goals and are defined as operations that have to be implemented for each goal (`possible` and `succeeded` in figure 4).

In order to describe the goal model, we first provide the following definitions:

- a goal g_1 is *inconsistent* with a goal g_2 if and only if when g_1 succeeds, then g_2 fails;
- a goal g_1 *entails* a goal g_2 (or equivalently g_2 is *entailed by* g_1) if and only if when g_1 succeeds, then also g_2 succeeds;
- a goal g_1 is a *precondition* of a goal g_2 if and only if, to be possible to pursue g_2, g_1 must succeed;
- a goal g_1 *depends* on a goal g_2 if g_2 is precondition of g_1 and g_2 must be successful while pursuing g_1.

Therefore a dependence is a stronger form of precondition. Both definitions let us specify that some goals must be successful before (and during, in case of dependency) pursuing some other goals. Moreover, when two goals are inconsistent with each other, it might be useful to specify that one is preferred to the other. Indeed, since several goals can be pursued in parallel, there is no need to prefer some goal to another goal if they are not inconsistent with each other.

The *goal model* of PRACTIONIST agents contains the set of goals the agent could pursue and all existing relations (i.e. inconsistence, entailment, precondition, and dependence) among such goals. A more formal definition of the goal model can be found in [9].

This goal model is used by PRACTIONIST agents when reasoning about goals during their deliberation process and the means-ends reasoning. In PRACTIONIST, desires and intentions are mental attitudes towards goals, which are in turn considered as descriptions of objectives. Thus, referring to a goal, an agent can just relate it to a *desire*, which it is not committed to because of several possible reasons (e.g. it believes that the goal is not possible). On the other hand, a goal can be related to an *intention*, that is the agent is actually and actively committed to pursuing it.

Suppose that an agent α starts its deliberation process and generates a goal $g = \langle \sigma_g,\ \pi_g \rangle$ as an option. Therefore the agent *desire*s to pursue the goal g. However, since an agent will not be able to achieve all its desires, it checks if it believes that the goal g is *possible* (i.e. if it believes that π_g is true) and not *inconsistent* with active goals (i.e. those goals that the agent is currently committed to).

If both conditions hold the desire to pursue g will be promoted to an *intention*. Otherwise, in case of inconsistence among g and some active goals, the desire to pursue g will become an intention only if g is preferred to all inconsistent goals, which will in turn be dropped.

In any case, if the desire to pursue g is promoted to an *intention*, before starting the means-ends reasoning, the agent α checks if it believes that the goal g *succeeds* (that is, if it believes that the success condition σ_g holds) or whether the goal g *is entailed* by some of the current active goals. In case of both above conditions do not hold, the agent will perform the means-ends reasoning, by either selecting a plan from a fixed plan library or dynamically generating a plan and finally executing it (details on this means-ends reasoning can be found in [10]).

Indeed, if the goal g succeeds or is entailed by some current active goals (i.e. some other means is working to achieve a goal that entails the goal g), there is no reason to pursue it. Therefore, the agent does not need to make any means-ends reasoning to figure out how to pursue the goal g.

Otherwise, before starting the means-ends reasoning, if some declared goals are precondition for g, the agent will first desire to pursue such goals and then the goal g.

PRACTIONIST agents adopt a *single-minded intention commitment* strategy. Thus, it will continue to maintain an intention until it believes that either such an intention has been achieved

or it is no longer possible to achieve the intention. Moreover the agent checks if some dependee goal does not succeed. If so, it will desire to pursue it and then continue pursuing the goal g.

In order to be able to recover from *plan failures* and try other means to achieve an intention, if the selected plan fails or is no longer appropriate to achieve the intention, then the agent selects one of applicable *alternative plans* within the same intended means and executes it.

If none of the alternative plans was able to successfully pursue the goal g, the agent takes into consideration the goals that *entail* g. Thus the agent selects one of them and considers it as an option, processing it in the way described in this section, from deliberation to means-ends reasoning.

If there is no plan to pursue alternative goals, the achievement of the intention has failed, as the agent has not other ways to pursue its intention. Thus, according to agents beliefs, the goal was *possible*, but the agent was no able to pursue it (i.e. there are no plans).

4 BDI Agent Patterns

The PRACTIONIST framework, with the execution model of agents, the goal model and the way such a model is used by agents, easily allows the direct implementation of some common intentional agent level design patterns. Some of them are described in this section, with a structure that reflects a restricted version of the classical structure of design patterns presented by GoF [3]. Each subsection is named with the pattern *name*, which conveys the essence of the pattern succinctly. The *intent* states what the pattern does, its rationale, what particular design issue or problem it addresses. The *motivation* is a scenario that illustrates a design problem and how the entities in the pattern solve such a problem. The *structure* is a graphical representation of the pattern using a notation based on the Unified Modelling Language (UML). Then in the *participants* section we provide more details about the entities involved in the design pattern and their responsibilities. Finally, the *implementation* describes how the proposed patterns can be implemented with the PRACTIONIST framework.

It should be noted that, as stated above, intentions are mental attitudes towards goals, that is an intention always concerns with being committed to pursuing some goal. Thus, according to the definitions and the agent reasoning model provided in section 3, declaring some relations among goals is equivalent to state the corresponding relations among the intentions of pursuing each of those goals. Therefore, although most of the presented patterns refer to intentions and relations among intentions, the solutions below are provided in terms of goals and relations among goals, given the semantics discussed in previous sections.

The proposed patterns are described using the *Tileworld* environment, as a system with a highly parameterized environment that could be used to investigate the reasoning in agents. The Tileworld consists of a grid of cells on which tiles, obstacles and holes (of different size and point value) can exist. Each agent can move up, down left or right within the grid to pick up and move tiles in order to fill the holes. Each hole has an associated score, which is achieved by the agent that has filled that hole. The main goal of the agent is to score as many points as possible.

4.1 Dynamic Strategy Selection

Intent

An agent's intention can be achieved through a family of strategies, which have the same purpose but work in different operative conditions: the best strategy should be dynamically selected by the agent at run time.

Motivation

One of the key features of practical reasoning agents is the clear separation among the deliberation process and the means-ends reasoning. Usually developers want to provide the agents with different strategies to achieve a given intention and let the agent be the responsible of applying the "best" strategy according to its beliefs and the current environment conditions. This would enable the development of more flexible agents by adopting a more modular and declarative approach.

Moreover, with the increasing complexity and maintenance cost of advanced software systems, in recent years attention has fallen on self-* systems and particularly on the autonomic

computing approach and autonomic systems. In [7] authors argue that adopting a design approach that supports the definition of a space of possible behaviours related to the same function is one of the ways to make a system autonomic. Then the system should be able to autonomously select at runtime the proper behaviour on the basis of the current situation.

In the Tileworld example, a player agent must find the holes to fill within the environment. In order to achieve such an intention, the agent could use several strategies according to the environment conditions, such as *(i)* finding a hole by *randomly* moving within the grid, *(ii)* finding a hole by means of a *breath first* research, or *(iii)* finding the hole with the *greatest value*.

Obviously some strategy is better than others in terms of agent scores, but perhaps it cannot be applicable in certain environment conditions (e.g. very rapidly changing conditions). Moreover, when the agent is trying to find a hole with a given strategy, if the environment conditions change, it can perceive this and dynamically change or adapt its research strategy according to the new situation.

Structure

The *agent* is provided with some *plan*s, each implementing a different `Strategy` to pursue a given *goal*. During the means-ends reasoning the *agent* performs the internal process `StrategySelector` to figure out the best *applicable* `Strategy` *plan*, according to the *context* of declared *plan*s and the current state of the world reflected into its *belief*s.

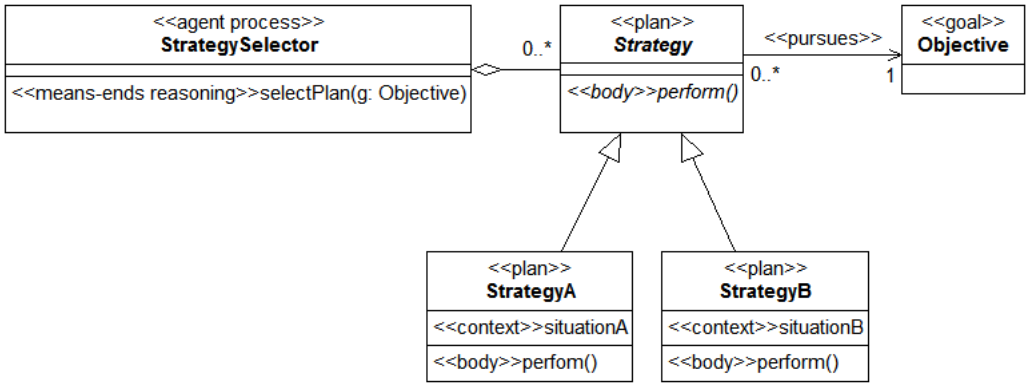

Figure 2: The structure of the *Dynamic Strategy Selection* pattern.

Participants

- **Objective**: some *goal*s the *agent* can pursue during its life-cycle by means of several strategies.
- **Strategy**: an abstract *plan* common to all concrete strategy *plan*s that can be adopted to pursue the same `Objective`.
- **StrategyA**, **StrategyB**: two concrete *plan*s the *agent* can use to pursue the same `Objective` in different ways.
- **StrategySelector**: an *agent process* that, in the context of the *means-ends reasoning*, has the responsibility of selecting a given concrete `Strategy` to pursue the `Objective`, if the conditions expressed by its *context* hold, at least according to the agent's *belief*s. This provides the *agent* with some meta-level reasoning capabilities to select the best applicable strategy under any operational condition.

Implementation

The `StrategySelector` process is provided within the PRACTIONIST framework with a default behaviour that checks the context of each *practical* plan (see section 2), by invoking

the `applicable` method of the class `Plan`. Therefore, each concrete strategy should be implemented as a sub-`Plan` of an abstract `Plan` which is *practical* for the considered *goal*. Each strategy sub-`Plan` should at least override both the `applicable` and the `body` methods.

4.2 Intention Decomposition

Intent

An agent's intention can be decomposed into a set of sub-intentions. Therefore, the main intention will be achieved as soon as all its sub-intentions are achieved.

Motivation

Often an agent is not able to achieve some high-level intentions. However, if the agent is aware that such an intention can be achieved if some other intentions are achieved, it can commit to them and indirectly bring about the main intention.

Actually, several analysis and design methodologies adopt a functional decomposition principle to describe and represent the expected functionalities of a system. Therefore, the ability to reason about intention decompositions can provide the agents with the capability of looking for alternative ways of achieving those intentions which it does not have a direct strategy/plan for.

In the Tileworld example, a player agent has to score as many points as possible. In order to "score points", the agent can "find a tile", "move to the (tile's) position", "pick up the tile", "move to the (hole's) position", and finally "fill the hole". All these objectives can be considered as contributors of the main goal "score points". Therefore, at a given moment, if the agent is committed to "scoring points", it could autonomously commit itself to pursuing all the above-mentioned contributors.

Structure

The agent is provided with the *entailment* relation `MainThroughABC` between the *goal* `MainObjective` and its sub-*goal*s (i.e. `ObjectiveA`, `ObjectiveB`, `ObjectiveC`).

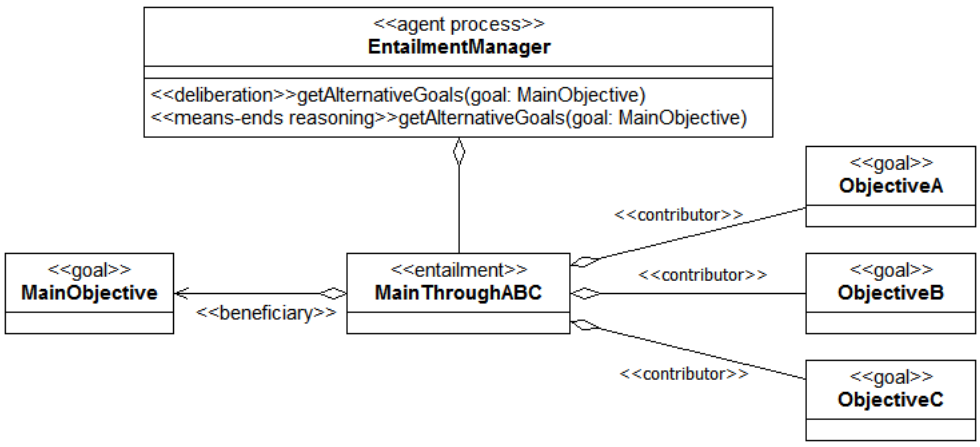

Figure 3: The structure of the *Intention Decomposition* pattern.

This relation will be analysed by the agent during its *deliberation* process in order to figure out the alternative intentions (i.e. pursuing *contributor* goals) when it is not able (due to some reason) to directly achieve the main intention (i.e. pursuing the main *goal*). In this case, all *contributor*s will be automatically pursued in place of the *beneficiary*.

Moreover, the above relation will be also analysed by the agent during its *means-ends reasoning*. Indeed, if the agent is committed to pursuing the `MainObjective` and the agent is already committed to pursuing `ObjectiveA`, `ObjectiveB`, and `ObjectiveC`, there is no reason

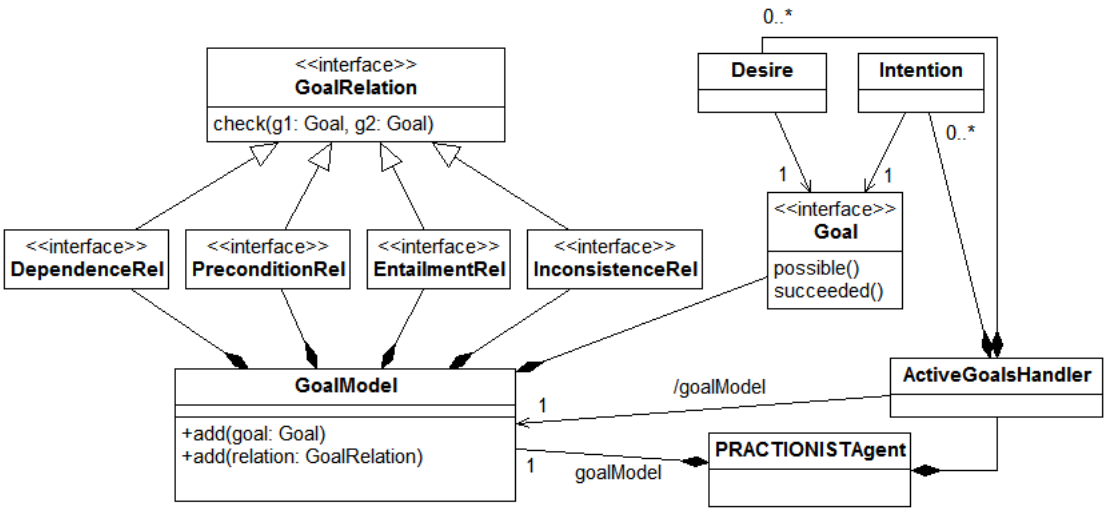

Figure 4: The structure of the support for the goal model in the PRACTIONIST framework.

to actually look for a plan to achieve the *beneficiary*, as it will be achieved as soon as all its *contributor*s are achieved.

Participants

- **ObjectiveA**, **ObjectiveB**, **ObjectiveC**: some *goal*s the *agent* can pursue during its life-cycle.
- **MainObjective**: a *goal* that the *agent* can pursue by pursuing all *contributors*.
- **MainThroughABC**: a relation stating that the *goal* `MainObjective` succeeds if all its *contributors* (i.e. `ObjectiveA`, `ObjectiveB`, `ObjectiveC`) succeed.
- **EntailmentManager**: an *agent process* that has the responsibility of providing the set of *contributors* of a given *goal*.

Implementation

The PRACTIONIST framework provides the support for the definition/handling of agent goal models and the capabilities for reasoning about goals.

Figure 4 shows the actual structure of the `GoalModel` that each agent owns (`PRACTIONISTAgent` is the abstract class that has to be extended when developing PRACTIONIST agents). Such a model stores information about declared goals (with their internal properties, i.e. success and possibility condition) and the four types of relations these goals are involved in. The interface `GoalRelation` provides the super interface for all goal relations supported by the framework (i.e. `EntailmentRel`, `InconsistencyRel`, `DependencyRel`, and `PreconditionRel`) and defines the operation `check`, whose purpose is to check each specific relation.

In order to exploit the features provided by the goal model and understand if a goal the agent desires to pursue is inconsistent with or implied by some active goals, the agent must have information about such active goals and whether them are related to either desires or intentions. Therefore, each PRACTIONIST agent owns an `ActiveGoalsHandler` component, which, with the aid of the `GoalModel`, has the responsibility of keeping track of all executing intended means stacks with the corresponding waiting and executing goals. Thus, at any given time, the `ActiveGoalsHandler` is aware of current desires and intentions of the agent, referring them to active goals.

The PRACTIONIST framework fully supports this pattern, as developers should only define the *entailment* relation between a goal and the set of its contributors (by implementing the interface `EntailmentRel`) and then add such a relation into the agent's `GoalModel` (through the method `add`). Then the agent will use it as described in section 3.

4.3 Mutually Exclusive Intentions

Intent

An agent could have two incompatible intentions, which cannot be achieve simultaneously.

Motivation

When developers want to specify that an agent has two intentions that are inconsistent with each other, the agent itself should not work to simultaneously achieve both intentions. It should choose the preferred one (if any) and perform only the activities to achieve it.

In the Tileworld environment, there are several inconsistent objectives that the agent must avoid to pursue at the same time. As an example, a player agent should not simultaneously intend to go toward two or more different holes. Thus, the agent should work to achieve only one of them and drop the other intention.

Structure

The agent is provided with an *inconsistence* relation between the two goals representing potential intentions (figure 5).

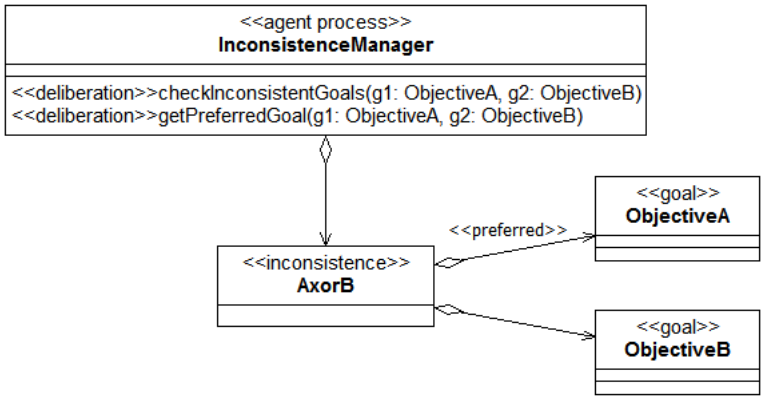

Figure 5: The structure of the *Mutually Exclusive Intentions* pattern.

This relation will be analysed by the agent during the deliberation phase in order to guarantee that at any moment it is not trying to achieve both intentions (i.e. pursuing the `ObjectiveA` and pursuing the `ObjectiveB`), by maintaining the preferred one and dropping the other one.

Participants

- **ObjectiveA**, **ObjectiveB**: some *goal*s the *agent* can pursue during its life-cycle.
- **AxorB**: a relation stating the *inconsistence* between the *goal* `ObjectiveA` and the *goal* `ObjectiveB`.
- **InconsistenceManager**: the *agent process* that has the responsibility of checking whether two declared goals are inconsistent with each other or not and if there is a preference between them. Then, the *agent* must guarantee that at any moment inconsistent or incompatible goals (e.g. `ObjectiveA` and `ObjectiveB`) are not pursued simultaneously.

Implementation

This pattern can be easily adopted when developing BDI agents with the PRACTIONIST framework. With reference to figure 4, developers should only define the inconsistence relation (by implementing the interface `InconsistencyRel`) and add such a relation into the agent's `GoalModel` (through the method `add`). Then the agent will use it during the deliberation process as described in section 3.

It should be noted that with the PRACTIONIST framework developers can also specify the preference between two inconsistent goals.

4.4 Necessary Intention

Intent

An agent must achieve an intention I_d before trying to achieve a given intention I. Sometimes the intention I_d must be maintained even while the agent is pursuing the intention I.

Motivation

Often it is useful to specify that some agent intentions must be achieved so that it is possible working to achieve other intentions. In this case, when an agent is committed to a given intention, if there are some intentions that must be achieved before, the agent should first perform activities to achieve them. Sometimes, such intentions must be achieved even when working to achieve the dependent intention.

In the Tileworld example, a player agent aims at maximizing its score by filling holes. However, a player agent should find a tile and hold it before working to actually fill a hole. Therefore, the objective "fill a hole" requires that the goals "find a tile" and "hold a tile" have been achieved.

Structure

The agent is provided with some *dependence* relation between some goals. These relations will be analysed by the agent during the deliberation phase in order to pursue dependee goals before (and sometimes while) pursuing the dependent goal.

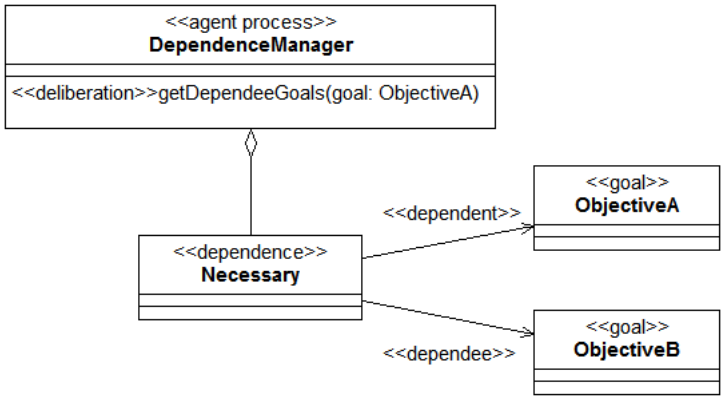

Figure 6: The structure of the *Necessary Intention* pattern.

Participants

- **ObjectiveA**, **ObjectiveB**: some *goal*s the *agent* can pursue during its life-cycle.
- **Necessary**: a relation stating that the *goal* `ObjectiveA` *depends on* the goal `ObjectiveB`, that is the latter should be achieved before (and in some cases even maintained while) pursuing the former.
- **DependenceManager**: the *agent process* that has the responsibility of checking if there is some *dependence* relation between `ObjectiveA` and `ObjectiveB`, in order to let the agent guarantee that the dependee goal is achieved before (and in some cases even maintained while) pursuing the dependent goal.

Implementation

The PRACTIONIST framework supports the adoption of this pattern. With reference to figure 4, developers should only define either the *dependence* or the *precondition* relation (by implementing either the interface `DependenceRel` or `PreconditionRel`) according to whether the dependee goal must be maintained while pursuing the dependent goal or not. Then the relation has to be added into the agent's `GoalModel` (through the method `add`). Thus the agent will use it during the deliberation process as described in section 3.

5 Conclusions and Future Work

In this paper we described how a declarative representation of goals can support the definition of desires and intentions of PRACTIONIST agents. This can also enable the detection and the resolution of conflicts among agents' objectives and activities. Indeed, unlike several BDI and non-BDI agent platforms, the PRACTIONIST framework provides each agent with the ability to figure out if goals are impossible, already achieved, incompatible with other goals, and so forth. This in turn supports the *commitment strategies* of agents and their ability to autonomously drop, reconsider, replace or pursue intentions related to active goals.

Moreover, programmers can implicitly specify different behaviours for several circumstances, without having to explicitly code such behaviours, letting agents figure out the right activity to perform on the basis of the current state and the relations among its potential objectives.

In order to address some specific recurring design problems when developing intentional agents, we presented and discussed some agent-level design patterns (i.e. dynamic strategy selection, intention decomposition, inconsistent intentions, and necessary intention), which can provide some guideline to take advantage from the goal model and the meta-level reasoning presented in this paper. Actually such patterns can be easily and directly implemented with the PRACTIONIST framework, thus providing developers with high-level solutions for some recurrent design problems. Moreover, it is well known that one of the most relevant effects of the introduction of design patterns is the improvement of the quality of the developed software.

It should be noted that although we refer to them with the term "design patterns", they are not general purpose patterns, but they are rather BDI-oriented solutions to recurrent problems we faced and solved. Results obtained by the applications of these patterns in some ongoing real projects are very encouraging. Then we plan to improve our work in this direction with the introduction of more patterns and the development of supporting tools integrated in the design tool we are developing. This could help the designer in identifying the proper pattern and easily applying it in the context of its design.

Finally we are currently working to define a complete methodology that supports the development of application using the PRACTIONIST framework. This methodology is based on a goal-oriented approach, where goals are a central concept in the design and the implementation of systems where agents should be able to both react to changes of the environment while pursuing their objectives.

Acknowledgments. This work is partially supported by the Italian Ministry of Education, University and Research (MIUR) through the project PASAF.

Moreover, the authors would like to thank *Anto* for her important contribution.

References

[1] Lars Braubach, Alexander Pokahr, Winfried Lamersdorf, and Daniel Moldt. Goal representation for BDI agent systems. In *Second International Workshop on Programming Multiagent Systems: Languages and Tools*, pages 9–20, 7 2004.

[2] P. Busetta, Ralph Rnnquist, Andrew Hodgson, and Andrew Lucas. JACK intelligent agents - components for intelligent agents in java. *Agentlink News*, January 1999.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[4] K. V. Hindriks, F. S. De Boer, Hoek Wiebe van der, and J. Jc Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999. Publisher: Kluwer Academic Publishers, Netherlands.

[5] Marcus J. Huber. Jam: a bdi-theoretic mobile agent architecture. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 236–243, New York, NY, USA, 1999. ACM Press.

[6] Prechelt L., Unger B., Philippsen M., and Tichy W. Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance. *IEEE Transaction on Software Engineering*, 28(6):595–606, 2002.

[7] A. Lapouchnian, S. Liaskos, J. Mylopolous, and Y. Yu. Towards requirements-driven autonomic systems design. *Proceedings of the 2005 workshop on Design and evolution of autonomic application software*, pages 1–7, 2005. ACM Press, New York, NY, USA.

[8] J. Lind. Patterns in agent-oriented software engineering. In *AOSE Workshop at AAMAS 2002*, Bologna, Italy, 2002.

[9] V. Morreale, S. Bonura, G. Francaviglia, F. Centineo, M. Cossentino, and S. Gaglio. Reasoning about goals in BDI agents: the PRACTIONIST framework. In *Proceedings of Joint Workshop "From Objects to Agents"*, Catania, Italy, 2006.

[10] V. Morreale, S. Bonura, G. Francaviglia, M. Cossentino, and S. Gaglio. PRACTIONIST: a new framework for BDI agents. In *Proceedings of the Third European Workshop on Multi-Agent Systems (EUMAS'05)*, page 236, Brussels, Belgium, 2005.

[11] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Rudy van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996.

[12] Anand S. Rao and Michael P. Georgeff. BDI agents: from theory to practice. In *Proceedings of the First International Conference on Multi—Agent Systems*, pages 312–319, San Francisco, CA, 1995. MIT Press.

[13] L. Sabatucci, M. Cossentino, and S. Gaglio. Building agents with agents and patterns. In *Proceedings of Joint Workshop "From Objects to Agents"*, Catania, Italy, 2006.

[14] Yasuyuki Tahara, Akihiko Ohsuga, and Shinichi Honiden. Agent system development method based on agent patterns. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 356–367, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[15] Michael Winikoff, Lin Padgham, James Harland, and John Thangarajah. Declarative & procedural goals in intelligent agent systems. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning*, pages 470–481, Toulouse, France, 2002.