A Goal Deliberation Strategy for BDI Agent Systems

Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf

Distributed Systems and Information Systems,
Computer Science Department, University of Hamburg
{pokahr, braubach, lamersd}@informatik.uni-hamburg.de

Abstract. One aspect of rational behavior is that agents can pursue multiple goals in parallel. Current BDI theory and systems do not provide a theoretical or architectural framework for deciding how goals interact and how an agent can decide which goals to pursue. Instead, they assume for simplicity reasons that agents always pursue consistent goal sets. By omitting this important aspect of rationality, the problem of goal deliberation is shifted from the architecture to the agent programming level and needs to be handled by the agent developer in an error-prone ad-hoc manner. In this paper a goal deliberation strategy called Easy Deliberation is proposed allowing agent developers to specify the relationships between goals in an easy and intuitive manner. It is based on established concepts from goal modeling as can be found in agent methodologies like Tropos and requirements engineering techniques like KAOS. The Easy Deliberation strategy has been realized within the Jadex BDI reasoning engine and is further explained by an example application. To fortify the practical usefulness of the approach it is experimentally shown that the computational cost for deliberation is acceptable and only increases polynomially with the number of concurrent goals.

1 Introduction

Goal-directedness is one important characteristic of rational agents, because it allows agents to exhibit pro-active behavior [19] and it is argued that the BDI (belief-desire-intention) model [3] is well suited to describe this kind of agents [16]. Typically, goal-directed agents should be capable of pursuing multiple goals simultaneously. As a consequence the agent's goals can interact positively or negatively with each other [18]. Positive interaction means that one goal contributes to the fulfillment of another one, whereas negative contribution indicates a conflict situation in which one goal hinders the other. Such contribution relationships between goals are commonly used in modeling agent applications, e.g. in the Tropos methodology [7] and in the requirements engineering technique KAOS [10]. Despite their usefulness, most implemented agent systems based on the BDI model do not support any mechanism for handling goal relationships at the architectural level. Hence, the cumbersome task of ensuring that the agent will never process any conflicting goals at the same time is left to the agent developer.

The main aspect of goal deliberation is "*How can an agent deliberate on its (possibly conflicting) goals to decide which ones shall be pursued?*" [5]. Considering this question from an architectural point of view it is of interest how a goal deliberation strategy can be integrated into a BDI infrastructure. Thereby, the agent infrastructure has the tasks to activate the strategy at certain points in time and to provide a clearly defined interface by specifying the possible operations for conflict resolution and exploiting positive goal interactions. These operations are constrained by the attitudes supported by the agent architecture. E.g. only when the architecture distinguishes between goals and desires the deliberation process can resort to both concepts.

Tackling the question from a strategy-centric point of view it is necessary to address at least the following issues:

1. *What are the important influence factors that can be used to drive the decision process?* As influence factors all of the agents attitudes such as the active goals or plans can be considered. Additionally, several approaches utilize meta-information about these attitudes such as resource requirements [17,18].
2. *When and how often shall the agent deliberate about its goals?* Generally, the strategy could require that the agent engages in the deliberation process in regular intervals (e.g. time or cycle driven) or on demand (e.g. when a new goal was created) or in a mixture of both.
3. *About what goal set shall the agent deliberate?* The options range from deliberation between just two goals to the consideration of all goals of an agent.

The approach presented in this paper proposes a deliberation strategy called Easy Deliberation which allows for specifying the relationships between goals for conflict detection. At runtime an extended BDI system ensures that the constraints of the concrete deliberation settings, as specified by an agent developer, are respected and only consistent goal sets are pursued at any one time. Main design rationale behind the strategy is the ease of use for agent developers requiring minimal specification overhead.

The remainder of this paper is structured as follows: In Section 2 explicit goal representation as necessary prerequisite for goal deliberation is discussed. Section 3 presents the conceptualization, realization and experimental evaluation of the Easy Deliberation strategy. A brief review of related work is introduced in Section 4. The paper concludes with a summary and an outlook on future work.

2 Explicit Goal Representation

Realizing a goal deliberation strategy has the necessary prerequisite that an agent is aware of its goals at any one time. In classical agent languages such as AgentSpeak(L) [14] and current BDI systems such as JACK [9] or Jason [2] this prerequisite is not fulfilled. The main reason for this shortcoming is that goals are represented in the transient form of events, which causes an agent to only know about its goals at the moment they need to be processed. As a consequence an agent e.g. cannot easily defer the processing of a certain goal, because there is no semantics behind the event representing the agent's intended desire. Hence, in the several papers [5,16] this implicit representation was criticized and different enhancements were proposed.

In this paper we build on the explicit representation of goals as described in [5]. In short, it consists of a generic goal lifecycle (cf. Fig. 1) that exactly describes the states and transition relationships of goals at runtime and forms the basis for different goal types such as perform, achieve, query, maintain. Adopted goals can be in either of the substates *Option*, *Active* or *Suspended*, whereby only active goals are currently pursued by the agent. Options and suspended goals represent inactive goals, where options are inactive, because the agent explicitly wants them to be, e.g. because an option conflicts with some active goal. In contrast, suspended goals currently must not be pursued, because their context is invalid. They will remain inactive until their context is valid again and they become options.

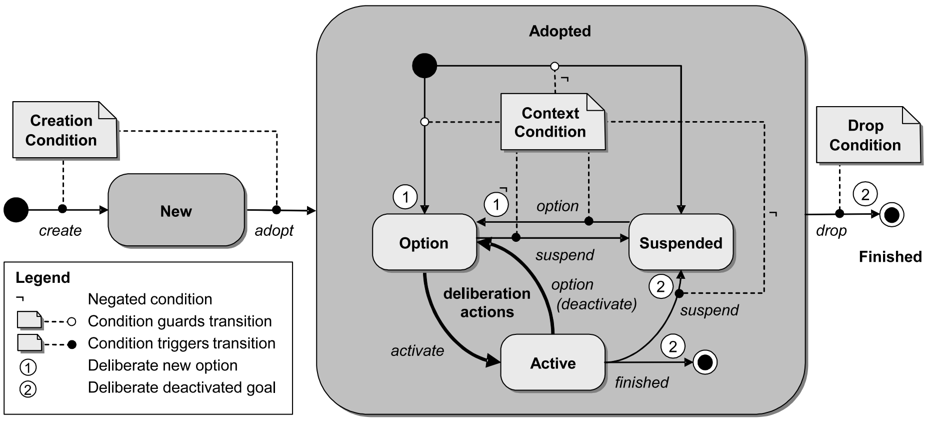


Fig. 1. Goal lifecycle (adapted from [5])

Additionally, some basic properties common to all goal types have been defined. Among those the most important ones are: A creation condition that defines when a new goal instance is created; a context condition that describes when a goal's execution should be suspended (to be resumed when the context is valid again); and a drop condition that defines when a goal instance is removed. At runtime, goal state changes occur, whenever one of the aforementioned conditions triggers or the agent intentionally changes the state, e.g. by exploiting a goal deliberation mechanism for this purpose.

3 The Easy Deliberation Strategy

Integrating a goal deliberation strategy requires that the agent can engage into the deliberation process whenever the strategy demands. Additionally, the operations available to the deliberation mechanism need to be clearly defined (cf. architectural viewpoint in Section 1). As a foundation for the definition of available operations for goal deliberation strategies, the generic goal lifecycle is used (see Fig. 1). For goal deliberation only adopted goals are of relevance as they represent the goals an agent is aware of.

From this lifecycle the operations for *activating* an option and *deactivating* an active goal, i.e. making it an option again, are derived as interface for goal deliberation, i.e. these transitions should be under control of the deliberation strategy (bold transitions in Fig. 1). This set of operations should not be considered as being the only possibility, alternative strategies might incorporate other actions such as *drop*.

3.1 Strategy Conceptualization

The Easy Deliberation strategy is conceived to allow real-time goal deliberation even when an agent pursues a multitude of goals simultaneously. The strategy is based on practical considerations derived from example applications and ideas from goal modeling as can be found in the agent methodology Tropos [7] and the requirements engineering technique KAOS [10], which both propose directed contribution links between goals. According to Section 1, the strategy will be explained by answering the characteristic questions:

1. *What are the important influence factors that can be used to drive the decision process?* The strategy is based only on information about goals, intentionally factoring out the plan level. Two main concepts are used to describe deliberation information within goal type declarations: *cardinalities* and *inhibition arcs*. Cardinalities can be used to constrain the maximum number of active goals of a specific type at runtime, whereas inhibition arcs are used to declare negative contribution relationships between two goals on type level as well as on instance level. On the type level it can be specified that a goal of a given type inhibits goals of the referenced type. For a finer-grained specification instance-level relationships between goals can be defined by attaching constraints to the inhibition links, which determine the goal instances affected by the inhibition. The strategy requires the inhibition links forming a directed acyclic graph to avoid infinite deliberation loops.
2. *When and how often shall the agent deliberate about its goals?* The deliberation process is initiated on demand. In Fig. 1 the triggering state transitions are depicted. Generally two different situations can arise, in which deliberation becomes necessary: First, a goal can become an option either when a new goal is adopted or when the context of a suspended goal becomes valid again. In these cases the deliberation process needs to decide whether the new option can be *activated* and additionally what the consequences of the activation are, i.e. which other active goals need to be *deactivated* to avoid having conflicting goals (*1: Deliberate new option*). Second, an active goal can become inactive when it gets suspended, finished or dropped. In this case, the deliberation has to determine which options have been possibly inhibited by the deactivated goal. For each of these options it needs to be checked whether it can be reactivated (*2: Deliberate deactivated goal*).
3. *About what goal set shall the agent deliberate?* The deliberation process only has to consider a local subset of the agent's goals, derived from the goal that triggered the deliberation with its state transition (see above). For goal types with cardinality, all instances of the goal type have to be considered. In addition, all goals with incoming and outgoing inhibition relationships to the triggering goal have to be taken into account.

In the following both goal deliberation actions will be described more formally. Given that all goals of an agent are in one of the states option, active or other defined by the sets $\Gamma_o, \Gamma_\alpha, \Gamma_\omega$, respectively, the full goal set of an agent is comprised of $\Gamma = \Gamma_o \cup \Gamma_\alpha \cup \Gamma_\omega$with $\Gamma_o \cap \Gamma_\alpha = \Gamma_o \cap \Gamma_\omega = \Gamma_\alpha \cap \Gamma_\omega = \emptyset$. A goal $\gamma \in \Gamma$ is defined as a tuple $\langle gt, s \rangle$ with gt being the user defined goal template in which creation, context and drop condition among other things are specified and $s \in \{option, active, other\}$

being the actual state of the goal. For simplicity reasons other aspects of concrete goal instances such as parameter values are not considered here.

The *Deliberate new option* action is responsible for activating an option $\gamma_o = \langle gt_o, option\rangle \in \Gamma_o$, if allowed in the current context. Therefore, first it has to be checked, if the goal can be activated by testing cardinality and inhibitions with the predicate $p_{act}(\gamma_o)$ defined as:

$$p_{act}(\gamma_o) : \Gamma_o \rightarrow \{true, false\}, p_{act}(\gamma_o) = \forall\gamma \in \Gamma_\alpha(\gamma \nrightarrow \gamma_o) \wedge \mid \Gamma_\eta \mid < f_{card}(gt_o)$$

$$\text{with } \Gamma_\eta = \{\gamma = \langle gt, active\rangle \in \Gamma_\alpha \mid gt = gt_o \wedge \gamma_o \nrightarrow \gamma\}$$

$$\text{and } f_{card}(gt_o) : \Gamma \rightarrow \mathbb{N} \text{ (cardinality function)}$$

$$\text{and } \rightarrow \subseteq \Gamma \times \Gamma \text{ (inhibition relation)}$$

The predicate $p_{act}(\gamma_o)$ is to true, when there is no active goal that inhibits goal γ_o, i.e. no pair (γ, γ_o), $\gamma \in \Gamma_\alpha$ is part of the inhibition relation $\rightarrow \subseteq \Gamma \times \Gamma$, and when the number of hindering goals in the set Γ_η is lower than the allowed cardinality of this goal defined by the function $f_{card}(\gamma_o)$. In the set of hindering goals are only those active goals which have the same template as the considered option $gt = gt_o$ and which are not inhibited by the option $\gamma_o \nrightarrow \gamma$ (because these active goals will be subsequently be made to an option). If the goal could be activated it needs to be determined if other currently active goals need to be deactivated. The set of active goals to be deactivated Γ_{inh} is defined as $\Gamma_{inh} = \{\gamma \in \Gamma_\alpha \mid \gamma_o \rightarrow \gamma\}$, which includes all goals the newly activated goal inhibits.

Thus, if an option can be activated the set of adopted goals changes so that the option is made to an active goal and all newly inhibited active goals become options:

$$\Gamma_{new} = \Gamma \setminus \{\gamma_o\} \cup \{\langle gt_o, active\rangle\} \setminus \Gamma_{inh} \cup \Gamma_{opt}$$

$$\text{with } \Gamma_{opt} = \{\langle gt, option\rangle \mid \langle gt, s\rangle \in \Gamma_{inh}\}$$

The *Deliberate deactivated goal* action has to compute for a just deactivated goal $\gamma_o = \langle gt_o, option\rangle \in \Gamma_o$ the set of options Γ_{test} for which it needs to be checked whether they can be reactivated:

$$\Gamma_{test} = \{\gamma \in \Gamma_o \mid gt = gt_o \vee \gamma_o \rightarrow \gamma\} \text{ with } \gamma = \langle gt, option\rangle$$

This set is composed of all options which have the same template as the considered goal $gt = gt_o$, because possibly cardinality allows for another goal of this type being activated. Additionally, all options need to be considered, which were inhibited by the deactivated goal $\gamma_o \rightarrow \gamma$. Note, that this is not the same set as Γ_{inh} because in this case inhibited options instead of active goals are considered. Of course, such goals will only be activated if the deactivated goal was the only inhibitor. As result of performing this action new *Deliberate new option* actions are produced for every option for which the deactivated goal was a *necessary condition* being not activated.

3.2 Realization

The newly conceived deliberation strategy is designed in terms of operations (*Deliberate new option*, *Deliberate deactivated goal*) which operate on the internal state of the agent. These operations have to be performed at proper times, e.g. when a new goal is adopted or an active goal is suspended or dropped (cf. Fig. 1). Therefore, these operations should not be executed continuously in each interpreter cycle. Instead, they should be activated whenever the need for goal deliberation arises.

To allow such flexible activation of goal deliberation operations a new interpreter architecture is proposed, which does not rely on a fixed interpreter cycle. The basic idea of the architecture is to break up the traditional BDI interpreter cycle [15] into a small set of self-contained meta-actions, which are invoked as needed, rather than being executed in a fixed sequence. The resulting set of meta-actions roughly corresponds to the steps of the original interpreter (see Fig. 2).

```
initialize-state();
repeat
    options := option-generator(event-queue);
    selected-options := deliberate(options);
    update-intentions(selected-options);
    execute();
    get-new-external-events();
    drop-successful-attitudes();
    drop-impossible-attitudes();
end repeat
```

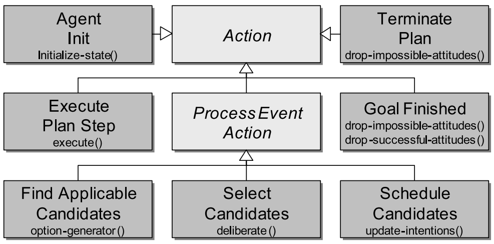


Fig. 2. Abstract interpreter (from [15]) and basic meta actions

The basic mode of operation of the proposed interpreter is depicted in Fig. 3. The interpreter is based on a data structure called *Agenda* where all meta-actions to be processed are collected. The interpreter continuously selects the next entry from the agenda and executes it, thereby changing the internal state of the agent. The execution of an action may further lead to the creation of new actions (*direct effects*), which are inserted into the agenda. Moreover, state changes may cause *side effects*, e.g. when a goal has to be dropped due to a changed belief. These side effects are also inserted to the agenda. More details of this architecture can be found in [11,12].

The presented interpreter architecture has been realized in the Jadex BDI reasoning engine [4,13], which establishes a rational agent layer on top of the JADE platform [1]. In Jadex, an agent type is described within an XML-file that adheres to a BDI metamodel specified in XML schema. In addition, for each plan used by the agent, a plan body has to be implemented in an ordinary Java class.

To integrate the Easy Deliberation strategy into the Jadex system, the basic set of interpreter meta-actions is extended with the newly defined Easy Deliberation actions (*Deliberate new option* and *Deliberate deactivated goal*). The creation of these actions is accomplished through conditions that guard the identified state transitions in the goal

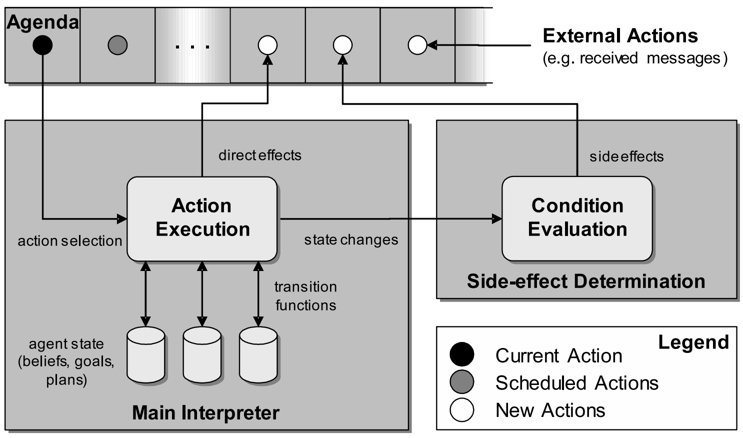


Fig. 3. Interpreter architecture

lifecycle. In order to allow the specification of user-defined deliberation settings in applications the Jadex BDI metamodel has been extended to incorporate the cardinality and inhibition settings directly within the XML agent specifications.

3.3 Example Application

To illustrate the strategy an example application called "cleaner world" is outlined (cf. [5]). The basic idea is that an autonomous robot has at daytime the task to look for waste in some environment and clean up the located pieces by bringing them to a near waste-bin. At night it should stop cleaning and instead patrol around to guard its environment. Additionally, it always has to monitor its battery state and reload it at a charging station when the energy level drops below some threshold. From this scenario the four corresponding top-level goals *PerformLookForWaste, PerformPatrol, AchieveCleanupWaste* and *MaintainBatteryLoaded* are derived. Initially, a cleaner possesses a *PerformLookForWaste*, a *PerformPatrol*, and a *MaintainBatteryLoaded* goal, whereas *AchieveCleanupWaste* goals are created for every piece of waste it discovers. To ensure correct operation several constraints must be met and are modeled by specific deliberation settings as described next (cf. Fig. 4):

– Only one *AchieveCleanupWaste* goal must be active at the same time to avoid the cleaner running to different pieces of waste concurrently. Therefore, the cardinality of this goal type is restricted to one.
– The agent must pursue exactly one of the top-level goals at the same time, whereby *MaintainBatteryLoaded* is the most important goal inhibiting all other goals. The *AchieveCleanupWaste* goal inhibits the *PerformLookForWaste* goal to force the agent to clean up known waste before looking for new. These inhibition relationships are introduced at the type-level, i.e. they always apply to all instances of, e.g., *AchieveCleanupWaste* goals. Note, that no deliberation is necessary to decide between *PerformLookForWaste* and *PerformPatrol*, as these goals have different contexts (at daytime vs. at night).

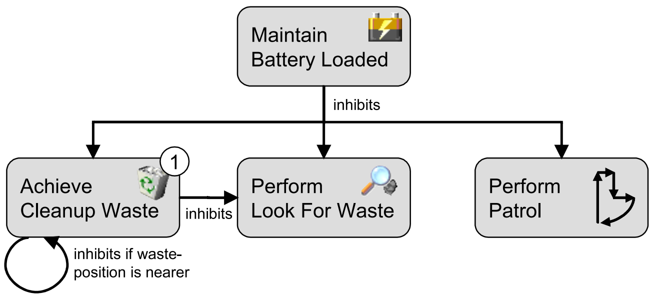


Fig. 4. Constraints between goals of a cleaner agent

```
<maintaingoal name="MaintainBatteryLoaded">
  [omitted parameter and condition specs. for brevity]
  <deliberation>
    <inhibits ref="AchieveCleanupWaste"/>
    <inhibits ref="PerformLookForWaste"/>
    <inhibits ref="PerformPatrol"/>
  </deliberation>
</maintaingoal>

<achievegoal name="AchieveCleanupWaste">
  [omitted parameter and condition specs. for brevity]
  <deliberation cardinality="1">
    <inhibits ref="PerformLookForWaste"/>
    <inhibits ref="AchieveCleanupWaste">
    $beliefbase.my_location.getDistance($ref.waste.location) >
    $beliefbase.my_location.getDistance($goal.waste.location)
    </inhibits>
  </deliberation>
</achievegoal>
```

Fig. 5. Cleaner agent XML fragment

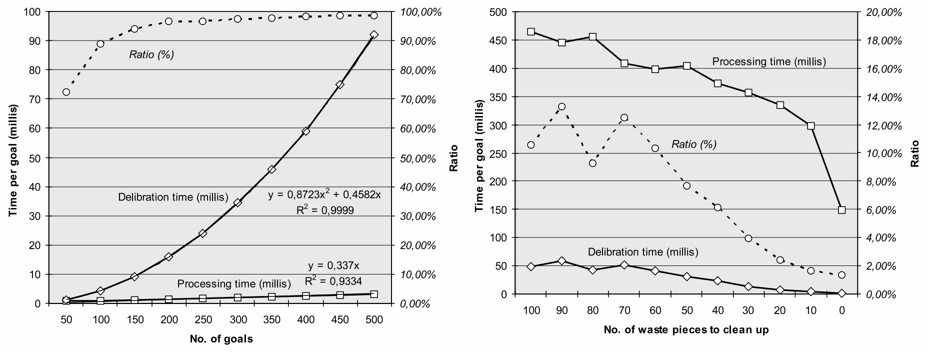
– For improved performance, the cleaner should always clean up the nearest piece of waste first. Hence, an instance-level inhibition arc for the *AchieveCleanupWaste* goal is introduced. An *AchieveCleanupWaste* goal instance inhibits another one, when its waste position is nearer to the agent. Note, that this constraint is not sufficient to replace the cardinality condition introduced earlier, because two or more waste pieces could have exactly the same distance from the cleaner.

The design decisions concerning the deliberation settings of the modeled goals can be directly mapped to the implementation. The extended Jadex XML schema allows deliberation settings to be embedded into the agent's goal specifications (see Fig. 5). The Jadex interpreter then uses these specifications to execute the agent, thereby automatically respecting all modeled dependencies between the goals.

3.4 Evaluation

For agents in dynamic domains, deliberation strategies are only useful, when they provide fast and efficient results, still allowing the agent to quickly react to changes in the environment. The Easy Deliberation strategy was designed to be computationally inexpensive, by only considering bilateral goal relationships. Therefore, the cost for deliberation should increase at most quadratically with the number of concurrent goals of an agent. To verify this analytical expectations, an empirical evaluation was performed.

Figure 6 (a) shows the results from an artificial test case, in which an increasing amount of concurrent goals with instance-level inhibition links has to be processed by an agent. This represents a worst-case scenario, where every present goal competes with any other goal. To obtain generalizable results, Application-specific code is omitted, i.e. no complex actions are performed to achieve the goals. The data we were interested in concerns the pure time for goal deliberation, the remaining time for goal processing (including e.g. plan selection and execution) and the ratio between them. The first thing to note is that the cost of goal processing increases linearly with the number of goals (as shown by the trend function y with regression coefficient R^2). This is due to more plan instances being created, which have to be considered in the plan selection process. Also one can see, that the cost of goal deliberation grows quadratically as expected. Not surprisingly, the ratio between goal processing and deliberation approximates to 100% very fast. With more than 100 concurrent goals, the agent spends 90 percent of its time thinking about which goals to pursue. Nevertheless, the absolute costs of deliberation are low (less than 100 ms even for 500 concurrent goals on a standard desktop PC[1]).



(a) Artificial test case (b) Cleaner example

Fig. 6. Evaluation results

To collect also data related to practice, a second evaluation was performed in the cleaner example presented above. In this case not only the speed of the reasoning engine was measured, but also the costs incurred by the application. Therefore, in Fig. 6 (b) the time needed for processing a single goal is about 100 times higher, due to the need for computing distances between pieces of waste (for deliberation) and for performing actions like moving the robot around. In this setting the robot starts with 100 cleanup goals which are processing in the order enforced by the deliberation settings. One can see that the deliberation cost decreases faster, as the robot cleans up more and

[1] Pentium 4(HT) 3 GHz, 512 MB RAM, WindowsXP, Sun Java 1.4.

more waste. Moreover, in this practical example, the relative time spent for goal deliberation does not exceed 14 percent of the total execution time, although the agent starts with a large number of goals (100). Even though a generalization of these results for other application domains cannot easily be drawn, this example is an indication for the overhead incurred by using explicit goal deliberation being acceptable, when used in the right context.

3.5 Discussion

The Easy Deliberation strategy has been used in several example applications and is sufficiently expressive for a wide variety of settings. Nevertheless, due to its simplicity it exhibits several conceptual limitations:

- The strategy does only consider bilateral relationships. Hence, it is impossible to specify e.g. that two goals together are more important than another single goal.
- Conflicts between subgoals cannot always be resolved optimally, e.g. when a conflict between subgoals could be resolved by replacing one of the subgoals with another non-conflicting subgoal [16].
- Conflicts at plan level are not considered, which means that inconsistencies between plans e.g. because of access to conflicting resources are not detected.
- Positive interactions between goals are not considered, which means that the strategy cannot identify and exploit potentially common subgoals.

Although some of these limitations indicate that the strategy cannot be applied universally to all kinds of problems, it is a straight-forward and easily understandable mechanism, due to reusing ideas from modeling approaches. The reason for choosing inhibition links instead of using utility values is that it allows to adopt a local view and frees the agent developer from establishing a global ordering between all goals. Our practical experiences have shown that the explicit declaration of goal deliberation information makes agent specifications simpler and more readable because concerns are clearly separated. The overhead in many practical settings is low, because a typical application consists of several different agents each deliberating only about small sets of related goals. Moreover, empirical evaluations reveal that the strategy only incurs low computational costs in general.

4 Related Work

The topic of goal deliberation within a single agent has not attracted much attention in the BDI agent community yet. One reason for this deficiency is that most implemented systems do not explicitly support goals and desires. Instead, these systems use a transient representation of goals as a type of event rendering the consideration about goals impossible [16]. In the area of planning agents a considerable amount of work has been devoted to plan scheduling. Main objectives of plan scheduling concern avoiding conflicts in plan execution and exploiting common steps via plan merging [6,8]. These approaches are different in that they require agents to have complete plans and do not support real-time decision control about goals and plans [17].

Our work concerning the Easy Deliberation strategy is similar to the work of Thangarajah et al., who propose strategies for detecting and resolving conflicts [17] as well as for exploiting positive goal interaction [18]. The influence factors of the conflict resolution strategy from [17] are annotated meta-data to plans and goals called "interaction summaries" containing information about their effects, pre- and in-conditions. This information is used at runtime to defer the adoption of possibly conflicting goals resp. the execution of plans. Compared to Easy Deliberation, the strategy greatly differs in the amount and the kind of deliberation data used and the resulting behavior. Our approach is designed to manage with minimal deliberation information based on agent modeling techniques providing an easy usable mechanism. In contrast, Thangarajah et al. require more detailed information that in return allows for handling conflicts also at plan level. Furthermore, besides ensuring that only conflict free goals are pursued, our strategy also respects the intended order of processing and is suitable for all goal types due to the underlying generic goal lifecycle.

5 Conclusion and Outlook

This paper motivates the need for goal deliberation strategies. To release the agent developer from the burden of ensuring that an agent always pursues consistent goal sets, an agent needs explicit information allowing it to deliberate about its goals, and autonomously select an appropriate goal set based on the current situation. In this paper the requirements for goal deliberation are discussed and a set of characteristic questions for conceiving a specific goal deliberation strategy is proposed.

The Easy Deliberation strategy is developed based on concepts from agent modeling techniques. It is designed to be intuitive to use with little specification effort and enables an agent to deliberate about its goals by activating and deactivating certain goals. The realization introduces two strategy specific meta-actions that are added to the underlying BDI interpreter architecture, by determining their activation points. During agent execution, the strategy enforces that only conflict free goals are pursued, additionally respecting the relative order of goal importance. Practical experiences with different applications indicate that the strategy considerably simplifies agent development and only incurs a low computational overhead.

Future work is devoted to the further investigation of deliberation strategies. We intend to experiment with alternative strategies, e.g. based on the work of Thangarajah et al. for comparing the effectiveness of different approaches in typical application domains. Especially, it is interesting to evaluate the advantages of detecting also plan conflicts and possibly extend the Easy Deliberation strategy in this respect.

References

1. F. Bellifemine, G. Caire, and G. Rimassa. JADE: The JADE platform for mobile MAS applications. In *Net.ObjectDays 2004: AgentExpo*, 2004.
2. R. Bordini and J. Hübner. *Jason User Guide*, 2004.
3. M. Bratman. *Intention, Plans, and Practical Reason.* Harvard University Press, Cambridge, Massachusetts, 1987.

4. L. Braubach, A. Pokahr, and W. Lamersdorf. Jadex: A BDI Agent System Combining Middleware and Reasoning. In M. Klusch R. Unland, M. Calisti, editor, *Software Agent-Based Applications, Platforms and Development Kits*. Birkhäuser, 2005.
5. L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. Goal Representation for BDI Agent Systems. In *Proceedings of the Second Workshop on Programming Multiagent Systems (ProMAS04)*, 2004.
6. B. Clement and E. Durfee. Identifying and resolving conflicts among agents with hierarchical plans. In *AAAI Workshop on Negotiation*, 1999.
7. F. Giunchiglia, J. Mylopoulos, and A. Perini. The Tropos Software Development Methodology: Processes, Models and Diagrams. In *Proc. of 1st Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS'02)*, 2002.
8. J. Horty and M. Pollack. Evaluating new options in the context of existing plans. *Artificial Intelligence*, 127(2):199–220, 2001.
9. N. Howden, R. Rönnquist, A. Hodgson, and A. Lucas. JACK Intelligent Agents-Summary of an Agent Infrastructure. In *Proc.of the 5th ACM Int.Conf. on Autonomous Agents*, 2001.
10. E. Letier and A. van Lamsweerde. Deriving operational software specifications from system goals. *SIGSOFT Softw. Eng. Notes*, 27(6):119–128, 2002.
11. A. Pokahr, L. Braubach, and W. Lamersdorf. A BDI Architecture for Goal Deliberation. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, 2005.
12. A. Pokahr, L. Braubach, and W. Lamersdorf. A Flexible BDI Architecture Supporting Extensibility. In *The 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2005)*, 2005.
13. A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI Reasoning Engine. In J. Dix R. Bordini, M. Dastani and A. Seghrouchni, editors, *Multi-Agent Programming*. Kluwer, 2005.
14. A. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In R. van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996.
15. A. Rao and M. Georgeff. BDI Agents: from theory to practice. In *Proc. of the 1st Int. Conf. on MAS (ICMAS'95)*, 1995.
16. J. Thangarajah, L. Padgham, and J. Harland. Representation and Reasoning for Goals in BDI Agents. In *Proc. of the 25th Australasian Computer Science Conf. (ACSC2002)*, 2002.
17. J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and Avoiding Interference Between Goals in Intelligent Agents. In *Proc. of the 18th Int. Joint Conf. on AI (IJCAI 2003)*, 2003.
18. J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and Exploiting Positive Goal Interaction in Intelligent Agents. In *Proc. of in the 2nd Int. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003)*, 2003.
19. M. Wooldridge and N. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.