Software Self-Reconfiguration: a BDI-based approach (Extended Abstract)

Fabiano Dalpiaz
DISI - University of Trento
38100 Trento, Italy
dalpiaz@disi.unitn.it

Paolo Giorgini
DISI - University of Trento
38100 Trento, Italy
pgiorgio@disi.unitn.it

John Mylopoulos
DISI - University of Trento
38100 Trento, Italy
jm@disi.unitn.it

ABSTRACT

Software self-reconfiguration is the capability of software systems to change autonomously their current configuration to a better one. This is a more and more requested feature, particularly for software systems that operate in critical domains when human intervention is not possible or not convenient. The Belief-Desire-Intention (BDI) architecture proposes a structured Monitor-Diagnose-Compensate cycle that partially meets self-reconfiguration requirements. We propose a realization of the abstract BDI control loop and we draw generic solutions to support the self-reconfiguration process. We aim at supporting traceability and runtime monitoring of requirements and we base our solution on Tropos goal models to structure agents' internal state.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Algorithms, Design

Keywords

BDI, self-reconfiguration, Multi-Agent Systems

1. INTRODUCTION

Modern software systems are an interplay of several interacting subsystems characterized by a high level of computational complexity. As a consequence, their development and management is getting more and more difficult for system developers and administrators.

Coping with complexity is not a trivial challenge, and cannot be addressed at design time only. Tight integration prevents designers from anticipating all possible interactions between system's components. Errors and failures not addressed at design-time should be handled through runtime reconfiguration, either by system administrators or the system itself. The second solution – *self-reconfiguration* – is the only possible option to reduce the workload of system administrators.

Cite as: Software Self-Reconfiguration: a BDI-based approach (Short Paper), Fabiano Dalpiaz, Paolo Giorgini and John Mylopoulos, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

The Belief-Desire-Intention (BDI) paradigm [5] is a natural solution for the development of multi-agent systems with self-reconfiguring capabilities. A BDI agent cyclically (i) updates its beliefs according to sensed events; (ii) identifies a set of options that react to events; (iii) selects plans to achieve the options and creates new intentions; (iv) executes a step of an intention; and (v) revises its intentions.

We introduce a realization of the BDI abstract interpreter [7] to support self-reconfiguring systems, exploiting well established techniques from Requirements Engineering and Databases. We provide an elaborate goal specification using Tropos [2], perform monitoring and problem determination on the basis of [6], enact failure handling taking inspiration from Sagas [3]. We have tested the feasibility of our approach by means of an implementation on Jason [1].

2. A BDI CONTROL LOOP FOR SELF-RECONFIGURATION

We introduce a revised version of the BDI control loop aimed at fully supporting self-reconfiguration. We model agent goals as goal trees, in which top-level goals are and/or-decomposed to sub-goals. The leaves of a goal tree – plans – are linked to goals by means-end decomposition. Some goals and plans are enriched with a *compensation plan*, which defines a reaction to failures in terms of (i) a failure handling plan and (ii) a reconfiguration of the system by switching to an alternative plan. Failures are detected only when compensation plans are defined, otherwise they are propagated bottom-up in the goal tree. Soft-goals, introduced by the *Non-Functional Requirements* (NFR) framework [4] to represent non-functional requirements, are used here as criteria to select alternatives.

We have revised the BDI agent control [7] as follows:

- *Intentions filtering* bases on policy compliance to verify if a desire is eligible to become an intention, and explicitly deals with delegations from other agents, creating a new intention only if the delegatee agrees to carry out the service on behalf of the delegator;
- *Planning* handles three different types of intentions: new top-level goals to be achieved, goal failures, and messages to be sent. Top-level goals require to explore the goal tree checking for the applicable plan that best contributes to softgoals. Goal failures imply failure compensation and the choice of the best alternative plan (Algorithm 1). Messages are sent to the recipient, specifying `no-plan` when no applicable plan is available for a delegated goal.

- *Reconsideration* is required in three cases: (i) a plan failed; (ii) the plans to achieve a goal ended but the goal is not achieved (declarative goal); (iii) a plan is performing below a minimum threshold (e.g., current contribution to soft-goals is not sufficient).

Algorithm 1 Reconfiguration algorithm

```
RECONFIGURE(i : intent)
 1  g ← i.trigger
 2  π ← EVTBESTREACT(g)
 3  if π ≠ NIL
 4    then return π
 5  while not done
 6  do if i = NIL
 7       then done ← true
 8       else  i ← POP(i)
 9             g_c ← i.trigger
10             π ← EVTBESTREACT(g_c)
11             if π = NIL
12               then continue
13             P ← STARTEDTOCOMPENSATE(g_c)
14             π_c ← NIL
15             for  each pl ∈ P
16             do if pl.compens ≠ NIL
17                  then π_c ← π_c + pl.compens
18             return (π_c + π)
19  return NIL
```

We describe now Algorithm 1, which defines the self-reconfiguration process enacted by an agent. First, it searches for a same-depth alternative to the failed goal/plan g (lines 1-2). If an alternative is found, the identified plan is returned (lines 3-4). Otherwise (*while* cycle in lines 5-18) backtracking is performed, searching for an alternative bottom-up in the goal tree. The cycle is repeated until the intention is empty (lines 6-7). We apply a POP operation to the intention for backtracking (line 8), and we seek an alternative at that depth (lines 9-10). If an alternative is not found we backtrack another level (lines 11-12). Otherwise (an alternative is found), we assign to P the set of plans started but not compensated yet (line 13), we create a plan π_c to compensate all these plans (lines 14-17), and we return the concatenation of the compensation and the reconfiguration plans (line 18). If we reach the top-level goal without finding alternatives we return NIL (line 19).

3. IMPLEMENTATION IN JASON

We implemented the major features of our approach on top of Jason [1], extending the class Agent to define self-reconfiguring agents compliant with our framework. We override a number of methods: `initAg` to initialize data structures, `selectEvent` to perform planning, and `selectOption` to choose the best alternative. Each agent contains a Java data structure associated to each top-level goal instance, fundamental to monitor the execution of the goal-driven agent and to enact compensation/self-reconfiguration. We keep Jason's syntax, and exploit plan labels and annotations to represent goals that belong to goal trees. Code 1 shows a simple example of the mapping to Jason. We use predicates to define soft-goals and their relative weight (the sum of all weights must be 1): in line 1, $s1$ is declared as a soft-goal with weight 1. The contribution from plans to soft-goals is expressed through prolog-like rules: lines 2 and 3 define the contribution of $p1$ and $p2$ to $s1$. The annotation `[tgoal]` (line 4) represents top-level goals triggering softgoal-based planning over goal trees, `[goal]` identifies other goals (line 5), and `[plan]` represents plans (line 7). The pre-condition of top-level goals is used to evaluate the contributions to soft-goals before planning.

Code 1 Goal models for self-reconfiguration in Jason.

```
1 softgoal(s1,1.0).
2 contrib(p1,s1,V) :- V=0.7.
3 contrib(p2,s1,V) :- V=0.3.
4 @g[tgoal]  +!g :
  contrib(p1,s1,V_p1_s1) &
  contrib(p2,s1,V_p2_s1)
  <- !g1; !g2.
5 @g1a[goal] +!g1 : true <- !g3.
6 @g1b[goal] +!g1 : true <- !g4.
7 @p1[plan]  +!g3 : true <- act1; act2.
8 @p2[plan]  +!g4 : true <- act3.
9 -!g3 : true <- compens1.
```

Currently, failure handling mechanisms based on goal trees and compensation techniques are fully implemented, and the revised BDI control loop is supported by the Jason reasoning cycle and the extended Agent class. Other features of our framework are not completely supported: we are currently working to define the policy engine supporting fine-grained meta-level reasoning, handle goal delegation through interaction protocols, and checking under-performance.

4. ACKNOWLEDGEMENTS

This work has been partially funded by EU Commission, through the SERENITY project and by MIUR, through the MEnSA project (PRIN 2006).

5. REFERENCES

[1] R. H. Bordini, M. Wooldridge, and J. F. Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.

[2] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[3] H. Molina and K. Salem. Sagas. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1987.

[4] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: a process-oriented approach. 18(6):483–497, 1992.

[5] A. Rao and M. Georgeff. An abstract architecture for rational agents. *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, pages 439–449, 1992.

[6] Y. Wang, S. McIlraith, Y. Yu, and J. Mylopoulos. An automated approach to monitoring and diagnosing requirements. *Proceedings of the 22nd IEEE/ACM international conference on Automated software engineering*, pages 293–302, 2007.

[7] M. Wooldridge. *Reasoning About Rational Agents*. MIT Press, 2000.