Capturing Task and Dependency Aspects in Agent Oriented Requirement Specifications

Kalyan Chakrawarthy and Rushikesh K Joshi

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Powai, Mumbai 400076, India
{kalyan,rkj}@cse.iitb.ac.in

Abstract. A method of capturing task and dependency aspects in an agent oriented requirements methodology based on Formal Tropos is presented. The emphasis is on capturing functional aspects in formal specifications through aspect extensions. With the help of a few examples, the benefit of aspect extensions in terms of reducing scattering from specifications are demonstrated.

1 Introduction

Development of modularity concerns is an important issue in software engineering as the development and evolution of software is closely tied to the inherent modularity abstractions. Traditional software decomposition methodologies based on object orientation are considered as significant improvements over their procedural counterparts. However, it is known that in some object oriented paradigms, certain concerns do not get modularized. Examples of such concerns have been discussed in [3] and [14]. The problems of separation of concerns have been addressed at programming level by several researchers. Some examples of such programming techniques are Encapsulators [13], Advices and Pointcuts [12], Composition Filters [2], Filter Relations [10, 9] and Context Relations [16].

Even at the requirements level, the problems of crosscutting concerns can arise due to the characteristics of abstractions used in modeling the requirements. We address this problem in the context of an agent oriented requirements specification methodology. The notion of aspects has earlier been applied in agent oriented programming systems. A few examples of such aspects are mobility aspects [18], interaction aspects [5], learning aspects [6], autonomy aspects [7] and collaboration aspects [11]. These solutions however do not address the problem of identifying and modeling concerns during the requirement phase of software development. Some examples of early aspects in requirements phase are the non-functional aspects demonstrated in a matrix based approach of Rashid et al. [15], and V-graph based approach of Yu et al. [20] for identifying non-functional aspects. A notion of *functional aspects* in i^* [19] based Agent oriented requirement modeling was discussed in [1, 8].

In this paper, we propose an aspect extension to Formal Tropos [4], and an extension to the metamodel of TAOM [17], a visual modeling representation

for Tropos. The proposed model is mainly intended for capturing *functional* aspects in agent oriented requirement specifications. The aspects outlined in this work capture generic concerns at task and dependency level. This method reduces a scattering caused by distribution of a single concern into multiple similar tasks or dependencies. Such a concern is captured through an aspect, which is subsequently translated into specific independent tasks or dependencies at the lower level. When aspect dependencies and aspect tasks are used, the size of the specification is reduced and a concern which otherwise is distributed across multiple tasks or dependencies is captured with in a single aspect.

The next section provides the motivation for the work through an example case. In Section 3, a method of capturing aspects through a visual model is outlined. Section 4 develops this model further into a formal description based on Formal Tropos. Translations from high level task and dependency aspects to native specification in FT are also described in Section 4.

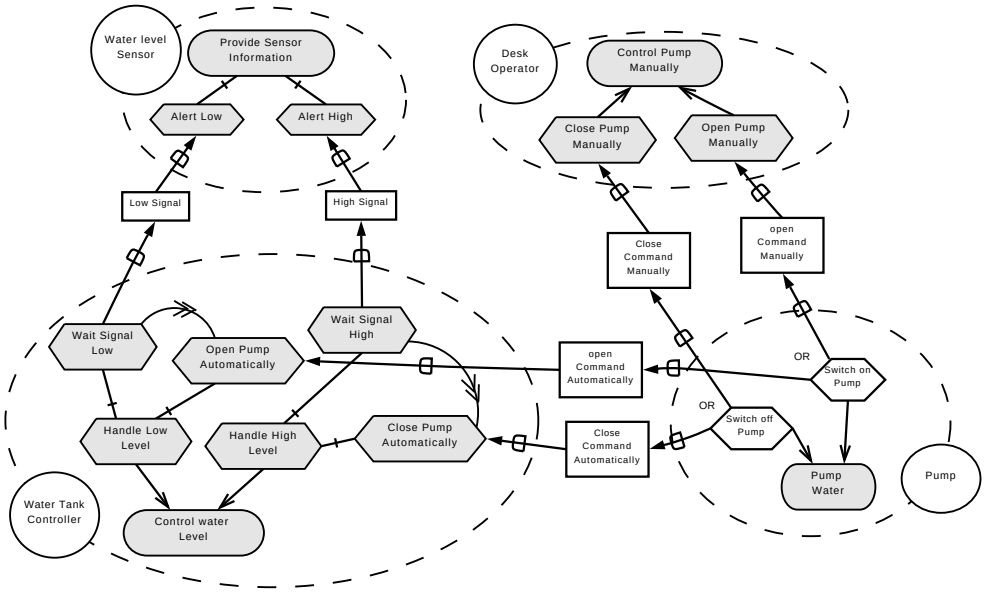

Fig. 1. Strategic Rational Model of the WLCP Problem

2 Development of an Aspect

This section illustrates the behavior of crosscutting concerns in an agent-centric requirement through an example specification of Water Level Controller Program (WLCP). WLCP is a problem about water level control in a Water Tank in between certain limits by controlling Pump. The requirements are represented

through an agent modeling method based on i^*. The i^* modeling framework uses the notions of actors, goals, tasks, resources and dependencies, making requirement specifications agent-centric as opposed to the traditional object-centric method. Crosscutting concerns may span across multiple actors, tasks, goals and dependencies in an agent-centric approach, while they span across classes in an object-centric approach.

2.1 The WLCP Problem

As shown in Figure 1, the WLCP specification consists of four agents: Water Tank Controller, Water Level Sensor, Desk Operator and Pump. The figure represents the i^* strategic rationale model of the specification. *Water Tank Controller* is an agent which controls the water level in the tank by operating the pump depending on the level of water in the tank. *Water Level Sensors* sense the water level in the tank and alert the controller when water level crosses specified limits, *Water Level Low* and *Water Level High*. *Desk Operator* can operate the pump manually by overriding automatic pump operations whenever needed. *Pump* provides water to the tank by taking commands from *Water Tank Controller* and *Desk Operator*.

From the figure, it can be observed that if handling of the pump is conceived as concern *Handle Pump*, the tasks belonging to this concern come from two agents *Water Tank Controller* and *Desk Operator*. In this case, it may not be possible to contain the concern in a single entity without losing the naturality of agent abstractions captured in this model. Now we outline an aspect based solution to this problem.

2.2 Agents and Aspects

Aspects need to be distinguished from agents. The properties that differentiate agent with aspects in the proposed model are given below.

- Agents in the system have the knowledge of other agents but not of aspects.
- Aspects may refer to agents without their knowledge and capture commonalities amongst the agents.
- On top of the commonalities they also capture the refinements.
- Aspects do not have goals to be fulfilled.

Methods such as matrix based analysis or V-graph [19] based analysis may be used for identification of crosscutting concerns. Once the crosscutting concerns are identified, a crucial problem is to decipher the entities and properties that are related to the concern, after which, a suitable aspect entity can be created in the formal description. The formal descriptions for aspect tasks and aspect dependencies are discussed later in the next section. The main activities in developing aspect tasks are listed below.

- Creation of an aspect-entity corresponding to the crosscutting concern identified.

– Relocation of relevant tasks belonging to the crosscutting concern from agents to the newly created aspect.
– An assessment of common properties and differences of the relocated tasks.

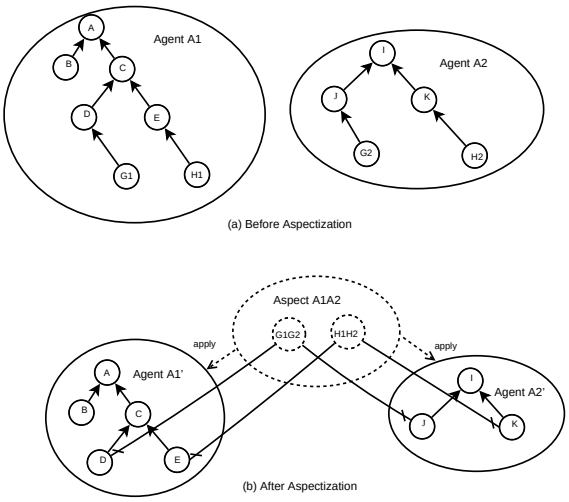

Fig. 2. A Method of Aspectization

The conceptual description of the approach is captured in Figure 2. An aspect *A1A2* has been extracted to represent the commonality in two agents *A1* and *A2*. The commonality in this case is among two pairs of tasks G1,G2 and H1,H2. The original tasks are removed from the agents. Each pair represents a single aspect task in the aspect. The aspect is applied to the two agents. The decomposition remains the same as shown. A similar method is applied for developing dependency aspects.

3 Visual Modeling of Task and Dependency Aspects

The aspectized representation of the concern *Operating Pump* is presented in Figure 3. As shown in the figure, an aspect is represented by a dotted circle. Representation of agents, goals, tasks and resources are as in i^* modeling language. An *apply to* relation between aspects and agents is also introduced. This relation captures the crosscutting linkages between aspects and agents. The relation gets elaborated further in the detailed specification through an extension on top of formal tropos. The concern *Handle Pump* which was scattered across agents *Water Tank Controller* and *Desk Operator* has been turned into a single aspect. The corresponding extensions to the TAOM metamodel are described in

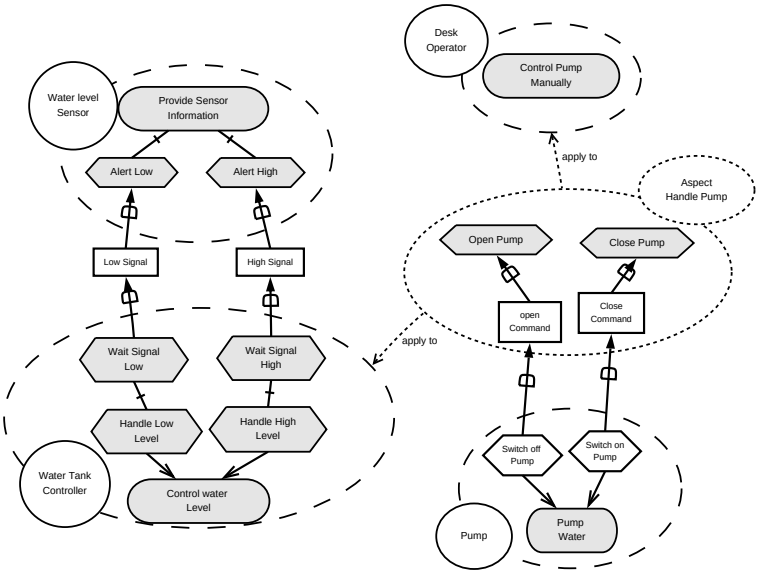

Fig. 3. Visual Model for Aspect HandlePump

Figure 4. The figure depicts an aspect as a separate entity which encapsulates crosscutting tasks and dependencies in a single entity. The goals and tasks of a crosscutting concern are contained in a single aspect and they contribute to the fulfillment of the respective agents.

4 Extensions to Formal Tropos

In this section, extensions to Formal Tropos [4] for specifying task and dependency aspects are presented. The core extensions are outlined below.

specification:= (entity | actor | **aspect** | int-element | **aspect-int-element**
| dependency | global - properties)*
aspect := **aspect** name [attributes] [aspect-creation-properties] [aspect-invar-properties]
[aspect-fulfillment-properties]
aspect-int-element := type name mode **aspect** name [attributes]
[aspect-creation-properties] [aspect-invar-properties] [aspect-fulfill-properties]
aspect-creation-properties := property-category event-category aspect-temporal-formulae
aspect-temporal-formula := **apply to** Agent-name-list temporal-formulae

4.1 Task Aspects

The main feature in the aspect task specification is that the temporal formulae in its properties include pointers to the agents to which they are applied. Every

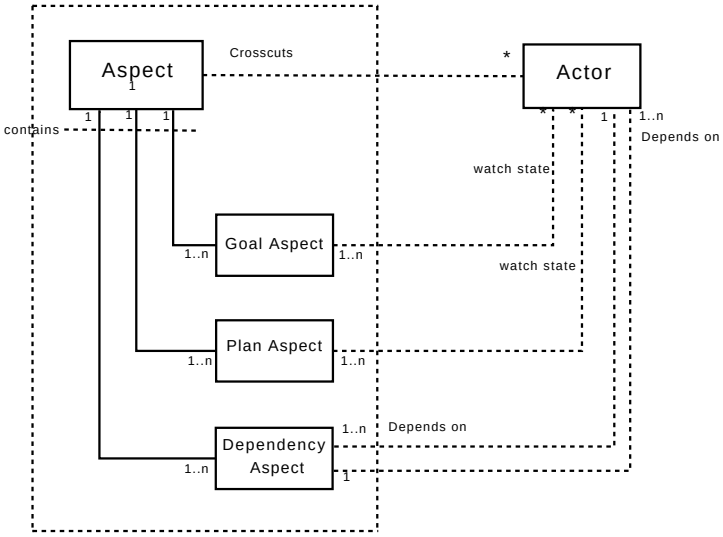

Fig. 4. The Aspect Metamodel in UML

intentional element that is part of an aspect specifies the name of the aspect to which it belongs. An example is demonstrated in Figure 5. The left column in the figure represents a snapshot of the non-aspectized specification based on Formal Tropos for the WLCP problem discussed in earlier sections. Only the entities that are aspectized are shown. As we can see from the formal tropos specification, the closely related and similar tasks *close pump automatically, close pump manually, open pump automatically* and *open pump manually* belonging to concern *Handle_pump* are scattered in two agents. A specification of an aspect begins with declaring the aspect as in a declaration given below:

```
Aspect HandlePump
```

The next step is to aspectize the tasks that scattered across multiple agents. In this case, tasks *OpenPump* and *ClosePump* are scattered in agents *Water_Tank_Controller* and *Desk_Operator*. The task *OpenPump* has two manifestations *Open_Pump_Automatically* and *Open_Pump_Manually*, one in each agent. In the figure, the right column represents the aspectized formulation of the two tasks into one aspect task *OpenPump*. The references to crosscutting agents are provided through a directive *apply to* in the temporal formulae of creation, invariant and fulfillment properties.

4.2 Dependency Aspects

In the WLCP specification, the dependencies for two pairs of resources for opening and closing of the pump are spread across two agents as it can be

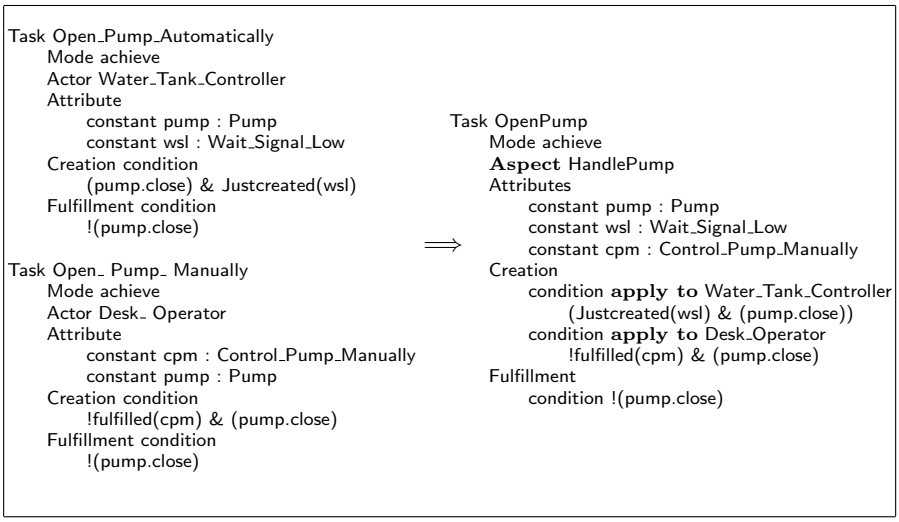

Fig. 5. Aspectization of Tasks *Open_Pump_Automatically* and *Open_Pump_Manually*

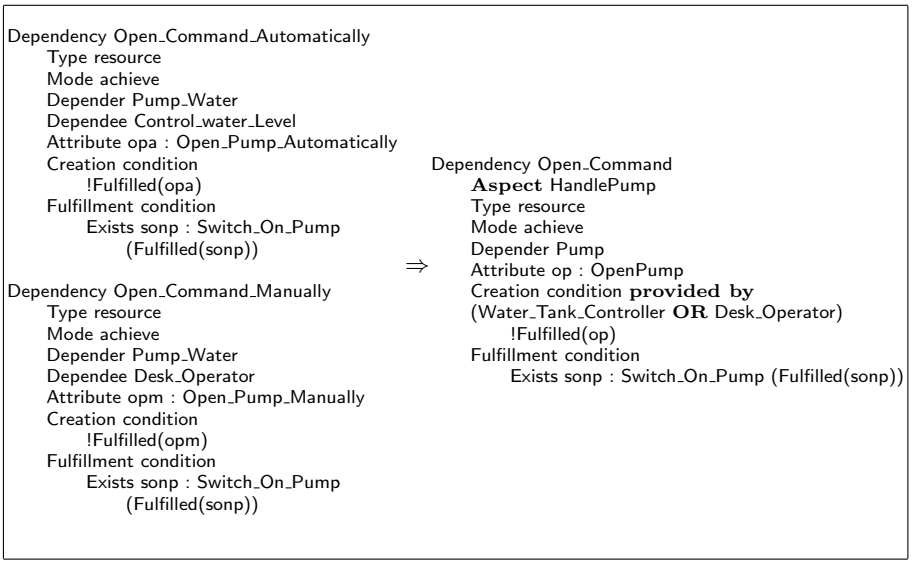

Fig. 6. Aspectizing *Open_Command_Automatically* and *Open_Command_Manually*

seen in Figure 1. There is similarity between the two dependencies within each pair. The four resources are collapsed into two aspects *Open_Command* and *Close_Command*. Figure 6 shows a snapshot of the aspectization of dependencies, left column representing the non-aspectized resource dependencies *Open_Command_Automatically* and *Open_Command_Manually*, and right column representing the dependency aspect *Open_Command*. Dependency aspects do not need to specify the *Dependee* clause as there are multiple dependees with which it is concerned. The dependees are referenced using the directive *provided by*. The specific combinatorial formulae for the dependencies are also captured independently for every agent.

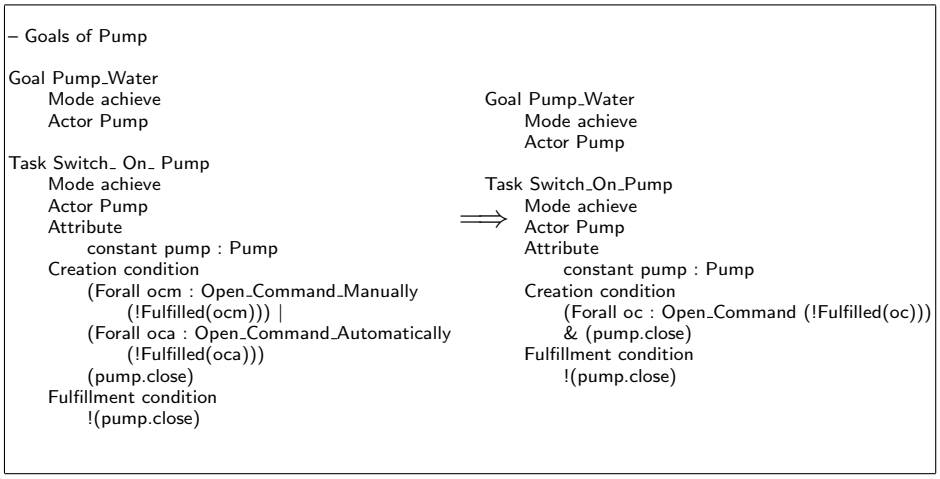

Fig. 7. Tasks References Before and After Aspectization

After the creation of task aspects and dependency aspects, all references to the original tasks and dependencies are redirected to the newly created task aspects and dependency aspects. In Figure 7 the left column represents the snapshot of pre-aspectized agent *Pump*, and the right column represents the post-aspectized specification. For example, all the references to dependencies *Open_Pump_Automatically* and *Open_Pump_Manually* are replaced by a single dependency aspect *Open Pump*.

A scheme for translating the aspectized specification into a low level specification in Formal Tropos is captured through an example in Figure 11. A numbering scheme is used to generate multiple tasks as dictated by the aspect specifications. If there are m aspect tasks that crosscut with k agents, the total number of different low level tasks generated are $m * k$. The temporal formulas for respective agents are replicated for the specific low level tasks. Similarly the desired low level dependencies are established.

```
procedure task_generator
    input A[1..m]: set of aspect tasks
    output n, T[1..n]: set to represent generated tasks
    precondition A ≠ φ, T = φ
    postcondition A = φ, T ≠ φ
    begin
        i ⇐ 0
        for each t in A
            a_t[] ⇐get_crosscutting_agents(t)
            for each a in a_t
                T[i] ⇐create_task(t, a)
                i ⇐ i + 1
            end
        end
    end
```

```
procedure create_task
    input t: aspect task, a: agent
    output g_t^a: task
    precondition g_t^a = φ
    postcondition g_t^a ≠ φ
    begin
        g_t^a ⇐ declare_task(t, a)
        g_cp ⇐ set_creation_properties(t, a)
        g_ip ⇐ set_invariant_properties(t, a)
        g_fp ⇐ set_fulfillment_properties(t, a)
        g_at ⇐ set_attributes(g_cp, g_ip, g_fp)
        g_t^a ⇐ g_t^a + g_at + g_cp + g_ip + g_fp
    end
```

Fig. 8. Translating Aspectized Tasks

- get_crosscutting_agents(t): returns the agents referred in aspecttask(t)
- declare_task(t, a): defines a task of type t in agent a.
- set_creation_properties(t, a), set_invariant_properties(t, a), set_fulfillment_properties(t, a): determine creation, invariant, fulfillment properties of aspect task t referring agent a.
- set_attributes(g_{cp}, g_{ip}, g_{fp}): create attributes for task(t, a) using the creation, invariant and fulfillment conditions belonging to task t in agent a.

```
Task OpenPump
    Mode achieve
    Aspect HandlePump
    Attributes
        constant pump : Pump
        constant wsl : Wait_Signal_Low
        constant cpm : Control_Pump_Manually
    Creation
        condition apply to Water_Tank_Controller
            (Justcreated(wsl) & (pump.close))
        condition apply to Desk_Operator
            !fulfilled(cpm) & (pump.close)
    Fulfillment
        condition !(pump.close)
```

⇒

```
Task OpenPump_000
    Mode achieve
    Actor Water_Tank_Controller
    Attributes
        constant pump : Pump
        constant wsl : Wait_Signal_Low
    Creation
        condition (Justcreated(wsl)) & (pump.close)
    Fulfillment
        condition !(pump.close)

Task OpenPump_001
    Mode achieve
    Actor Desk_Operator
    Attributes
        constant cpm : Control_Pump_Manually
        constant pump : Pump
    Creation
        condition !Fulfilled(cpm) & (pump.close)
    Fulfillment
        condition !(pump.close)
```

Fig. 9. Translating Aspectized Tasks: An Example

```
procedure dependency_generator
    input ad[]: dependency_aspects
    output d_t^a[]: dependencies
    precondition ad ≠ φ
    postcondition ad = φ
    begin
        for each t in ad
            a_t[] ⇐ get_dependees(t)
            for each n in a_t
                d_t^a[] ⇐ create_dependency(t, n)
            done
        done
    done
```

Fig. 10. Translating Aspectized Dependencies

- get_dependees(t): returns the agents the dependency aspect t crosscuts.
- create_dependency(t, n): creates dependency between depender specified in t and dependee n

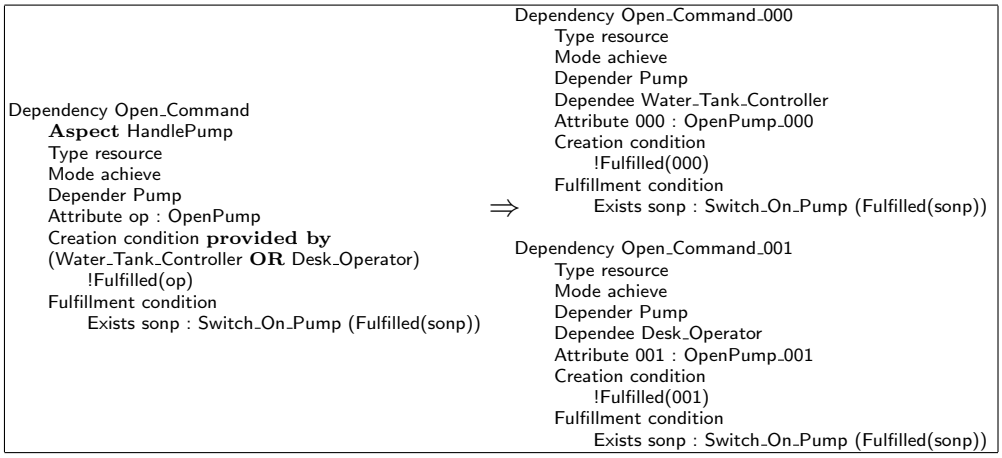

Fig. 11. Translating Aspectized Dependencies: An Example

5 Conclusions

An approach to capturing of early-aspects in an agent oriented specification methodology based on i^* and Formal Tropos was proposed. The proposed extensions include representation of aspects as separate entities. Aspects are collections of aspect tasks and aspect dependencies that crosscut with multiple actors. Each aspect task captures some commonality and some difference amongst

a group of tasks that it aspectizes. The approach was demonstrated through an example formal specification. It was found that if the specification can capture the commonalities effectively, it can cause reduction in the number of tasks that otherwise are quite similar to each other. Through the temporal formulae, the aspects note the differences between the behaviors when applied to different agents.

References

1. Satyashil Awadhare. Extensions to tropos for requirements engineering. Master's thesis, Department of Computer Science & Engineering, Indian Institute of Technology Bombay, 2004.
2. L. Bergmans and M. Akşit. Composing crosscutting concerns using composition filters. *Comm. ACM*, 44(10):51–57, October 2001.
3. Robert E. Filman and Daniel P. Friedman. Aspect-oriented programming is quantification and obliviousness. In Mehmet Akşit, Siobhan Clarke, Tzilla Elrad, and Robert E. Filman, editors, *Aspect-Oriented Software Development*. Addison-Wesley, Reading, MA, 2004.
4. Ariel Fuxman, Lin Liu, John Mylopoulos, Marco Roveri, and Paolo Traverso. Specifying and analyzing early requirements in tropos. *Requir. Eng*, 9(2):132–150, 2004.
5. Alessandro F. Garcia and Carlos J. P. de Lucena. An aspect-based object-oriented model for multi-agent systems. In Peri Tarr and Harold Ossher, editors, *Workshop on Advanced Separation of Concerns in Software Engineering (ICSE 2001)*, May 2001.
6. Raj P. Gopalan, Tariq Nuruddin, and Yudho Giri Sucahyo. A seamless integration of association rule mining with database systems. *CoRR*, cs.DB/0106055, 2001.
7. Zahia Guessoum and Jean-Pierre Briot. From active objects to autonomous agents. *IEEE Concurrency*, 7(3):68–76, 1999.
8. Rushikesh K. Joshi. Early aspects in agent oriented modeling. ITPAR Workshop on Knowledge and Logic Oriented Software Engineering, Tata Institute of Fundamental Research, Mumbai, 19-21 January 2005, 2005.
9. Rushikesh K. Joshi, Maureen Mascarenhas, and Yogesh Murarka. Filter objects for java. *Softw, Pract. Exper*, 33(6):509–522, 2003.
10. Rushikesh K. Joshi, N. Vivekananda, and D. Janaki Ram. Message filters for object-oriented systems. *Softw. Pract. Exper.*, 27(6):677–699, 1997.
11. Elizabeth A. Kendall. Role model designs and implementations with aspect-oriented programming. In *OOPSLA*, pages 353–369, 1999.
12. Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. In J. Lindskov Knudsen, editor, *ECOOP 2001 — Object-Oriented Programming 15th European Conference*, volume 2072 of *Lecture Notes in Computer Science*, pages 327–353. Springer-Verlag, Budapest, Hungary, June 2001.
13. Geoffrey A. Pascoe. Encapsulators: A new software paradigm in smalltalk-80. 21(11):341–346, November 1986. OOPSLA '86 Conference Proceedings, Norman Meyrowitz (editor), September 1986, Portland, Oregon.
14. Awais Rashid and Lynne Blair. Aspect-oriented programming and separation of crosscutting concerns. *The Computer Journal*, 46(5):527–528, September 2003.
15. Awais Rashid, Ana M. D. Moreira, and João Araújo. Modularisation and composition of aspectual requirements. In *AOSD*, pages 11–20, 2003.

16. Linda M. Seiter, Jens Palsberg, and Karl J. Lieberherr. Evolution of object behavior using context relations. *IEEE Trans. Software Eng.*, 24(1):79–92, 1998.
17. Angelo Susi, anna Perini, and John Mylopoulos. The tropos metamodel and its use. Technical report, May 09 2005.
18. Naoyasu Ubayashi and Tetsuo Tamai. Separation of concerns in mobile agent applications. In A. Yonezawa and S. Matsuoka, editors, *Metalevel Architectures and Separation of Crosscutting Concerns 3rd Int'l Conf. (Reflection 2001), LNCS 2192*, pages 89–109. Springer-Verlag, September 2001.
19. E. S. K. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings: 3rd IEEE International Symposium on Requirements Engineering*, pages 226–235. IEEE Computer Society Press, 1997.
20. Yijun Yu, Julio Cesar Sampaio do Prado Leite, and John Mylopoulos. From goals to aspects: Discovering aspects from requirements goal models. In *RE*, pages 38–47. IEEE Computer Society, 2004.