Experiencing AUML for the WINK Multi-Agent System

Sonia Bergamaschi
Gonzalo Galadi
Francesco Guerra
Maurizio Vinci

WOA 03 - Dagli Oggetti agli Agenti
September 10 - 11, 2003
Tanka Village
Villasimius, Sardinia, Italy

Outline

- Background
- Applying AUML Diagrams: Conceptual Level
- Applying AUML Diagrams: Implementation Level
- Conclusions

Background

Case study

The Web-linked Integration of Network-based Knowledge project WINK
Partners: Gruppo Formula (Bologna site), dbgroup@unimo@Alenia Spazio (Turin site)
Goal: supporting project management for large-scale, multi-supplier and one-of-a-kind production
Solution: integration network process
Test case: Alenia Spazio’s programs
An activity within the: EUTIST-AMI cluster Contract
Number IST-2000-28221
**Background**

- Research on agent-based software development has been a very important discipline for agent technology.
- The field is characterised by a variety of approaches, surveyed in [Igl98].
- Desired features:
  - It should be possible to distinguish a conceptual and an implementation or development phase during the overall process.
  - Different models to catch different views of the system-to-be should be defined (agents, environments, interactions and organisations).
  - They support not only agent-based systems, but also ensure that real applications envisage a blend of object-oriented services and agent technology.

---

**Choice**

- We have chosen AUML:
  - It leverages the expressive power of an existing language, by extending it.
  - It leverages on existing knowledge, proposing to designers and developers to learn extensions of something they already know.
  - It offers a common language for understanding and exchanging design.
  - It is intended to be neutral with respect to the methodology followed during the design/implementation process.
- Main references: Bauer 2001 a and b, Huget 2002 a and b.
Applying AUML Diagrams

Conceputal level

- The conceptual level is used for depicting the agents and classes that compose the system together with the relationships intercurring among them.
- Besides classes, we distinguish agents, identified with the stereotype `<agent>`.
- UML defines the relationships of ASSOCIATION, GENERALIZATION, AGGREGATION AND DEPENDENCY between classes.
- In AUML, we need to clarify what these relationships mean when agent are involved (both for agents-classes and classes-classes relationships).

Defining relationships (Huget)

- Agents-agents relationships:
  - Association: the linked agents are acquaintant and can exchange messages.
  - Generalisation: the definition of an agent can be derived from other agents.
  - Aggregation: an agent is aggregate of another if it has a recursive architecture.
  - Dependency: it is a mutual dependency between the linked agents.
- Agents-classes relationships:
  - Association: the agent uses the connected classes for their execution.
  - Generalisation: not possible.
  - Aggregation: the agent is defined as an aggregation of several classes.
  - Dependency: the agent needs the class either in its code or during its execution.

Example Class Diagram
Class Diagrams: Observations (I)

1) Relationship between agents: the Query Agent and the Wrapper Agent are acquaintance and can exchange messages. This can be rendered by using an association relationship. Further, one could claim there is the Query Agent relies on the Wrapper Agent to achieve its task. This can be rendered by using a dependency. Further, the QA and WA can engage in more complex interactions, such as coordination and cooperation. For the time being, it seems to us there is no way to capture this using AUML.

Class Diagrams: Observations (II)

2) Relationships between agents and classes: we propose to distinguish classes into service classes (or simply services) and behaviour classes (or simply behaviours)

- The meaning of the relationships becomes as follows:
  - Association: the association relationship connects an agent to the service classes it exploits. It means that the agent uses the connected service classes for its execution.
  - Aggregation: the aggregation between agents and behaviour classes implies that agents are defined as an aggregation of several behaviours, i.e. an agent can show the set of connected behaviours (no matter whether they are statically or dynamically acquired).
  - Dependency: the dependency between agents and service classes is possible and means that one agent needs the services during its execution since the agent would possibly exploit the service execution. A dependency between agents and behaviour classes is also possible, but it would have the same semantics as aggregation.

Applying AUML Diagrams

Implementation Level

- This level of abstraction develops the implementation phase
- Usually (see Gaia and Tropos) we disclose details about:
  - the agency view, which deals with agent's knowledge, belief, intentions, plans and behaviours;
  - the environmental view, which models how agents react to external changes;
  - the interactional view, where interaction protocols are specified;
  - The organizational view, which gives details on organizations to which an agent belongs.
Detailed Class Diagram

- Helps us specify details about the agency, environmental and organizational view

Observation

- For the organizational view, Huget proposed to add a capabilities compartment (free format textual description of what agents are able to do)
- But we have defined behaviour classes that capture this kind of knowledge
- This results in a more compact notation

Interactional View

The Interactional view presents how the agents interact
- The notions encompassed in AUML protocol diagrams are the following:
  • agents and their roles
  • agent lifelines and their connectors, synchronization (Huget)
  • Messages: conditions, cardinality, type of messages, repetition, sending messages until delivery (Huget), broadcast (Huget)
  • Protocols: nested protocols, interleaved protocols
  • triggering actions (Huget)
  • exception management (Huget)
  • time management (Huget)
  • atomic transaction (Huget)

Example

- Example of interaction between different roles and lifelines
- Diagram showing message flow and interactions
Conclusions

- The choice of AUML has revealed a clear improvement towards understanding the nature of multi-agent systems and how they combine with other technologies.

- We have changed something in order to design our system in a more expressive way.

- What really lacks when working with AUML are development tools that support the graphical representation and the translation into code. Future work could possibly be spent for writing plug-in modules to add AUML notation to existing IDE.

- In our exercise, we have not tackled the design of the system topology. A proposal for Configuration and Deployment Diagrams has been done in [Pogl02]. This will be addressed in future work.