Implementation Level Issues in MAS Modelling

Massimo Cossentino\textsuperscript{1}, Agostino Poggi\textsuperscript{2}, Giovanni Rimassa\textsuperscript{2}, Paola Turci\textsuperscript{2}

\textsuperscript{1}Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
Consiglio Nazionale delle Ricerche (CNR)
\textsuperscript{2}AOT Lab - DII, Università di Parma
Implementation Issues

- Issues connected with the transition from the design phase to the implementation phase
  - Diagrammatic notations and tools which should support developers
    - Refinements of the design models
    - Deployment of the system
    - Need of reducing the prototype implementation time

- Three main research lines:
  - Representation of the interactions between agents and their relationship with the architecture of the single agent
  - Deployment diagram of MASs
  - Pattern reuse
Two dimensional plane: Component vs. Interconnection

- The diagram is placed at the:
  - Agent-object boundary in the component dimension
    - Looking at the diagram one can recognize
      » Agent-level entities
      » Object-oriented artefacts with which the entities are expressed
  - Agent-society boundary in the interconnection dimension
    - Looking at the diagram one can see
      » The part of the social structure the agent participate in as a whole
      » Which internals parts of each agent get actually involved in the ongoing interaction

Parametric diagram

- Cannot be independent from the agent architecture; concrete agent architectures as a parameter
- The aim is to support
  - Well-known agent architectures
  - Custom agent architectures expressed through an object-oriented model
Choosing the Perspective: UML Activity Diagrams

- Our work lies within the AUML initiative. We look at standard UML diagrams as the basis for our notation.
  - properties that our diagram should have
    - Showing interactions at a detailed level
    - Showing agents and objects
    - Showing inter-agent and intra-agent links
- Among the various UML diagrams, we decided to use the Activity Diagram, because:
  - Combines structural and behavioural aspects
  - Can have precise constraints (e.g. temporal)
  - Can be recursively expanded with sub-activities
  - Provides flexible clustering with swim lanes
Different Levels of Detail

- Exploiting the recursive aggregation of activities allowed by UML activity diagram
- Global vision of the system, low level of detail
  - One swim lane for each agent
  - Activities within each swim lane group the dynamics of an agent
    - Links between activities represent
      » Message exchange, when they cross swim lanes
      » Task creation or activation, when they connect activities within the same swim lane
  - At this level, the activities granularity is comparable to an agent behaviour
Diagram for FSM-Based Agents

genericAM

ExecutorBehaviour

/msg(AddNewAnnotation:request)

notify-newAnnotation

content language=CoMMA-RDF
ontology=CoMMA-user

InformPushDocumentBehaviour

InformPushDocumentHandlerBehaviour

Always active:

genericUPM

NewAnnotationHandlerBehaviour

/new(FIPARequestInitiatorBehaviour)

request-matchUserProfile

content language=CoMMA-RDF
ontology=CoMMA-ontology

PushDocumentHandlerBehaviour

<<datastore>>
UserProfileDataBase

Sends messages to all known UPM:

A list of User Profiles:

genericLocalUPA

/NewAnnotationMatcherBehaviour

/content language=CoMMA-RDF
ontology=CoMMA-user

SenderPushDocumentsBehaviour

/new(PushEngineBehaviour)

PushEngineBehaviour

<<datastore>>
UserProfileDataBase

genericUserIC

SenderPushDocumentsBehaviour

/new(FipaRequestResponderBehaviour)

FIPARequestResponder

/new(NotifyUPMBehaviour)

NotifyUPM

/new(FipaRequestResponderBehaviour)

FIPARequestResponder

/new(INformPushDocumentsBehaviour)

/new(INformPushDocumentsBehaviour)

InformPushDocumentBehaviour

InformPushDocumentHandlerBehaviour

PushDocumentHandlerBehaviour

request-PushDocuments

content language=CoMMA-RDF
ontology=CoMMA-user

<Interested>

[not interested]

/topic=FIPARequest
content language=CoMMA-RDF
ontology=CoMMA-user

/topic=FIPARequest
content language=CoMMA-RDF
ontology=CoMMA-user
Different Levels of Detail

- ... a more detailed activity diagram (PASSI MABD diagram)
  - One different swim lane for each agent behaviour
  - Each activity addresses a method level entity
  - Is modelling methods as activities a proper approach?
    - Shows the object-oriented idioms to be used to effectively represent and implement agent oriented concepts
      » Agent has control over the life cycle of its tasks, which cannot be accessed from the outside
      » Tasks can use agent internal data as a common repository, to share information among them

- A specific CASE tool should support the zooming operation
PASSI –
Multi-Agent Behaviour Description
Why Multi-Agent Deployment Modelling?

- In UML 2.0, it is the only form of implementation diagram
- Relevance in MAS
  - Has to represent hosts, resources, physical agents, their acquaintance graphs and MAS platforms.
    - Deployment diagram is very useful to model highly distributed MAS
      - To visualize the system’s current topology and distribution of components and agents
      - To reason about the impact of changes on the topology
  - Lack of concepts and notations to fully capture the multi-agent system deployment
- Starting point: Agent UML (AUML)
  - Strictly related to the nearest antecedent technology
    - ... an agent is an active object plus “something else” (thus OO provides a good starting point)
    - Commonly used and popular
  - Focus on a new subset of an agent-based UML extension for the specification of the deployment
AUML Deployment Diagram Notation

UserProfileServer: AppServer

- UserProfile Archivist
  - acquaintance

- UserProfile Manager

UserPlatform: APInstance

- Mobile
  - location=home
  - JoeAgent: Personal Agent

Joe’sPC: Client

- Mobile
  - location=home
  - JoeAgent: Personal Agent

Mary’sPC: Client

- MaryAgent: Personal Agent

Reason: obtain information about Mary

Mobility relationship

Node

Concrete agent

Connection

Acquaintance relationship
We want to be precise and rigorous
- The aim is to provide complete semantics for all modelling notations used in the AUML deployment diagram
  - Mapping the new entities in UML model elements

The definition of the stereotype agent of the metaclass Class, as a starting point
- Agent Class has the attribute isActive always true
- Agent Class defines a set of elements, Agents, instances of Agent Class

When we speak of ComponentInstance we refer to its executable part
- A ComponentInstance will host concrete elements - concrete agents
  - “concrete agent” denotes an instance that originates from an implementation of an Agent Class.
MASs benefit from sound methodology and notation but there is more ...

- APIs, libraries and infrastructure are also very important to achieve a success
- Component and Deployment Diagrams tightly related to concrete software infrastructures and middleware
  - Consider the agent oriented middleware standards
  - Find useful abstractions to model them

- We define the stereotype `agentPlatform` of the metaclass Component
  - Component, seen more as an infrastructure for agents deployed in it rather than as a mere agent container
Pattern reuse is a very challenging and interesting issue

- Focus on the definition and the implementation aspects
  - Goal: provide a strong automation support to the designer/developer

Each pattern is composed of:

- A model of the structure of the involved elements (class diagram)
- A dynamical representation of the agent behaviours (activity diagram)
- The implementation code
Using Patterns During the Design

Request Participant pattern – *behaviour-sized* pattern
Patterns Implementation

- ... from *platform independent representation* to *platform specific model*
  - Restrictions of the possible implementation platforms
    - FIPA platforms
  - Java as the implementation language

- Code generation
  - We aim at going beyond the generation of the Java skeleton of the agent (and its tasks)
    - Introduction of significant parts of the inner code of the methods
    - *Code Reuse Library*: a library of patterns by class and activity diagrams with associated related code
Discussion and Conclusions

- Advantages:
  - Diagrams can be used within the existing tools of UML
  - Agent-based system characteristics are well supported
  - Pattern reuse cuts down the time and cost

**but ...**

- More work needs to be done to gather more comments within the AUML proponents and users

- Open issues
  - Other agent architectures should be considered (e.g. BDI)
  - **non-social situatedness**
    - Agent surroundings $\rightarrow$ non-agent entities
      - Representing non-agent entities and their connections with agents requires further study
        - To integrate them in the UML metamodel