MODEL-DRIVEN
AGENT ORIENTED SOFTWARE ENGINEERING

Dr Nikolaos Spanoudakis
Teaching/Research Staff
School of Production Engineering and Management
Applied Mathematics and Computers Laboratory
Technical University of Crete
Tel. : +30 28 21 03 77 44
Email: nikos@science.tuc.gr
www : users.isc.tuc.gr/~nspanoudakis
Tutorial Summary

- Motivation and Background
  - Software Engineering
  - Model-driven engineering
- Agent Oriented Software Engineering
- AOSE Methodologies
  - Protocols Specification
- Model-Driven AOSE with ASEME
  - The Domain-Specific Language
  - The process
  - The tools
Motivation and Software Engineering

- I know how to program, look, this is a simple “Hello world”, I did it in 10 seconds!!

- Why do I need Software Engineering?
- Why should I even spend my time drawing models?
Teamwork needs

- Process
- Vocabulary
- Role assignment
- Documentation
- Planning
- Progress Assessment
- ...
The Scientific Domain

- The **Software Engineering** (IEEE, 1990) discipline:
  - applies a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software
  - studies such approaches
A software process provides answers to the following questions (Boehn, 1988):
- “What shall we do next?”
- “How long shall we continue to do it?”

A software methodology is “A predefined and organized collection of techniques and a set of rules which state by whom and in what order the techniques are used” (Tolvanen, 1998)
SE models abstractions

(Kleppe, 2009)

Run a nuclear power plant

Autonomous Negotiation

Business Process

Warehouse

Agent

Aspect

Design pattern

Virtual zero line

Entity bean

Component

Object/Class

Method

Subroutine

Loop

If statement

XML

Association

Database table

Hardware zero line

Memory location

Accumulator

Instruction

Business Concepts

EASSS 2016 - N. Spanoudakis - MoD-AOSE 26 July 2013
**Waterfall and Iterative Process**

- **Requirements**
- **Analysis**
- **Design**
- **Implementation**
- **Verification-Validation**
- **Maintenance**

- **Determine objectives, alternatives, constraints**
- **Evaluate alternatives, identify and resolve risks**
- **Commit to new iteration**
- **Review**
- **Plan next phases**
- **Develop and verify next level product**

**Foundations of SE**

EASSS 2016 - N. Spanoudakis - MoD-AOSE  
26 July 2013
Mainstream SE

- RUP
- UML
- Statecharts

Jet
- Moves
- Takes passengers
- Needs fuel
- Flies
- Has wings

Vehicle
- Moves
- Takes passengers
- Needs fuel

Flying Vehicle
- Flies

Ground Vehicle
- Has wheels

Jet
- Has wings

Helicopter
- Has rotor blades

Boat
- Floats

26 July 2013
Agility (Fowler and Highsmith, 2001)

- New development needs
  - addressing continuously changing requirements
  - continuous evaluation
  - motivated individuals
  - less bureaucracy

- Representatives
  - XP (Beck, 2000)
  - Scrum (Schwaber and Beedle, 2002)
Trends in Software Engineering (2)

- **Modularity**
  - Give importance to interfaces
  - Heterogeneous programs interoperability
  - Assembling components

- **Service Oriented Computing**
  - Service description
  - Service discovery
  - Service orchestration
Autonomic Systems

- Self-configuration: Automatic configuration of components;
- Self-healing: Automatic discovery, and correction of faults;
- Self-optimization: Automatic monitoring and control of resources to ensure the optimal functioning with respect to the defined requirements;
- Self-protection: Proactive identification and protection from arbitrary attacks.
- ... (more self-* properties)
Model-Driven Engineering (MDE)

Proposes the systematic use of models as primary engineering artifacts (Beydeda et al., 2005) allowing for:
- portability
- interoperability
- reusability

Model Driven Architecture (MDA) of OMG (2001)
- Computation Independent Model (CIM)
- Platform Independent Model (PIM)
- Platform Specific Model (PSM)

Domain-Specific Languages (DSLs) and Metamodelling
Model Driven Engineering (MDE)

- Model driven engineering relies heavily in model transformation
- and to metamodeling (Jouault and Bézivin, 2006)
Model Transformation

- Four types of transformations (Langlois et al., 2007)
  - model to model (M2M)
  - model to text (M2T)
  - text to model (T2M)
  - text to text (T2T)
The metamodel allows

- To write formally a transformation process
- To use the Eclipse Modeling Framework which automates a number of tasks, and,
- allows the use Java (and other languages - DSLs) for manipulating models and for writing transformations
The ecore metametamodel

of Eclipse Modeling Framework (EMF)
A class diagram contains one or more classes

Classes have a name and visibility

Moreover, classes can reference other classes

Finally, they can have a super class

EClass with name: Class

EAttribute with name: name

EReference with name: super, containment: false, lowerBound: 0, upperBound: 1, eReferenceType: Class

EDataType with name: EString
Sample model (graphical editor)
Sample model (text-XML editor)
Now, let’s create a metamodel

- Sequence diagrams are common in SE
- Each object has one or more ordered timelines and can send and receive messages in a timed sequence
- Messages can be method invocations or return messages
All classes must be contained by another except the Model class.
A UML sequence diagram model
<?xml version="1.0" encoding="UTF-8"?>
<AUML_AIP:Model
  xmi:version="2.0"
xmi:schemaLocation="http://www.example.org/AUML_AIP UML_SD.ecore">
  <objects name="X">
    <timelines/>
  </objects>
  <objects name="Y">
    <timelines/>
  </objects>
  <objects name="Z">
    <timelines/>
    <timelines
      after="/@objects.2/@timelines.0"/>
  </objects>
  <messages
    origin="/@objects.0/@timelines.0"
    destination="/@objects.1/@timelines.0"
    type="call"
    name="do(X,Y)"/>
  <messages
    origin="/@objects.1/@timelines.0"
    destination="/@objects.2/@timelines.0"
    type="call"
    name="give(Z)"/>
A DSL is a computer language specialized to a particular application domain.

The line between general-purpose languages and DSLs is not always sharp, e.g.:

- Parallel programming language (Ada)
- Query language SQL
- 3D models representation (CAD)

We use EMF and metamodels for defining DSLs.
Computer-Aided Software Engineering (CASE) tools improve productivity and quality in software development (Baik and Boehm, 2000)

Tools aid the software development process and lead to code generation

Model transformation prevents errors of understanding or copying
Knowledge Engineering

- Is about
  - Knowledge Representation
  - Inference Engines and Tools

- Trends
  - Ontology
    - Development
    - Merging
    - Sharing
  - Web (OWL, RDF)
I have the knowledge on how to program an agent (method/know-how, programming language, etc). Look, here is my ground-breaking argumentation-based negotiating agent...

Why do I need AOSE?
What is it about?
Agents vs Objects

**Agent**
- Beliefs (world state)
- Sensors
- Environment
- Goals
- Select Action
- Effectors
- Plans

**Class**
- Class
- Properties
- Methods

26 July 2013
Agents vs Objects (2)

Society of Agents (Multiagent Architecture)

High-level Dynamic Interactions between Agents

Interactions with the Environment

Environment

(Zambonelli, 2010)

Traditional Software Architecture

Object (component) → Object (component)

Functional Dependencies Between Objects/Components

Object (component) → Object (component)

Object (component) → Object (component)

Object (component) → Object (component)
Agent Oriented Programming (AOP)

- Events
  - From the environment
  - From peers
- Goals
  - Select next goal
- Plans
  - Select plan
  - Revise plan
- Beliefs
  - Environment state
  - Belief revision
- Knowledge
  - Domain knowledge
- Actions
  - Act on environment
  - Send message
Research a methodology for defining software that is (Wooldridge and Jennings, 1995; Weiss, 2003):

- Autonomous
- Social
- Reactive
- Pro-active
- Adaptive
- Persistent

The domain is maturing (more than 15 years)

- OOD invented around 1982, first UML version in 1997

Most important for Multi-Agent Systems (MAS)
AOSE Considerations

- What, how many agents?
- How to structure of agent?
- Model of the environment?
- Communication?
- Relationships?
- Coordination?
- Protocols?

(Hexmoor and Brainov, 2002)
Intrinsic Attributes (Weiss, 1999)

- Life span (temporary, permanent)
- Knowledge level (reactive, deliberative)
- Construction (declarative method, procedural method)
- Mobility (static, mobile)
- Adaptiveness (rigid, self-learning, learning)
- Models (self-model, world-model, other agents models)
Extrinsic Attributes (Weiss, 1999)

- Locality (local, remote)
- Autonomy (autonomous, controlled)
- Sociality (autistic, informed, responsible, team member)
- Friendliness (cooperative, antagonistic, truthful, benevolent)

Interactions
- How (directly, through facilitators or mediators or non-agents)
- Whom (other agents, people, the world)

Semantics (declarative or linear communications)
Software Agent Types

Sycara and Zeng (1996)

And since then...

Or even Intelligent Personal Assistant (IPA)
Platforms

- 1990: AGENT-0 (Shoham)
- 1993: PLACA (Thomas; AGENT-0 extension with plans)
- 1996: AgentSpeak(L) (Rao; inspired by PRS)
- 1997: 3APL (Hindriks et al.)
- 2000: JACK (Busetta, Howden, Ronnquist, Hodgson)
- 2000: CLAIM (Amal El FallahSeghrouchni)
- 2001: JADE (Bellifemine, Poggi, Rimasa)
- 2002: Jason (Bordini, Hubner; implementation of AgentSpeak)
- 2003: Jadex (Braubach, Pokahr, Lamersdorf)
- 2008: 2APL (successor of 3APL)

(extended from Koen Hindriks, 2011)
Some AOSE Methodologies

- **Multi-agent Systems Engineering** (Deloach et al., 2001, 2006)
- **The Gaia Methodology** (Wooldridge et al., 2000 ; Zambonelli et al., 2003)
- **Agent UML** (Odell et al., 2000)
- **Vowels** (Ricordel and Demazeau, 2002)
- **PASSI** (Burrafato and Cossentino, 2002)
- **Prometheus** (Padgham and Winikoff, 2003)
- **Adelfe** (Bernon et al., 2003)
- **Ingenias** (Pavón and Gómez-Sanz, 2003)
- **Tropos** (Bresciani et al., 2004)
- **ASEME** (Spanoudakis and Moraitis, 2009)
- **Malaca, Gormas, ….**
AOSE Methodologies

Why are there so many?
Which one should I use?

OOD had a number of proposals before UML became a standard
MaSE

- A UML-based Methodology
- MaSE defines a system goal oriented MAS development methodology
- The authors define for the first time inter and intra-agent interactions that must be integrated
- O-MaSE (Deloach, 2005) introduced the organization concept, introducing the use of AUML in MaSE
Goal-based role definition

- Goal Hierarchy Diagram
- Role model
- Agent class model
Plan Model

- Get Temperature plan

receive(requestTemp(), corr) [temp <> undefined()] ^ send(send(temp), corr)

- State Initialize
  - t = setTimer(rate)
  - temp = undefined()

- State Wait
  - [timedOut(t)]

- State ReadTemp
  - t = setTimer()
  - n = findNeighbors()
  - temp = readSensor()

- State ComputeGradient
  - g = computeGrad(temp)

- State RaiseAlarm
  - g = raiseAlarm(temp, loc)

receive(monitor(area), Initiator) ^ send(accept(), Initiator)

[g <= threshold]

^ send(send(temp), <n>)

[g > threshold]
The Gaia Methodology

- The Gaia methodology (Wooldridge et al., 2000) is an attempt to define a general methodology that is specifically tailored to the analysis and design of MAS.

- Gaia defines the structure of a MAS in terms of a role model.

- The model identifies the roles that agents have to play within the MAS and the interaction protocols between the different roles.
The Gaia process

- requirements statement
- roles model
- interactions model
- agent model
- services model
- acquaintance model

Analysis

Design

EASSS 2016 - N. Spanoudakis - MoD-AOSE 26 July 2013
The Gaia Analysis phase

- Roles consist of:
  - responsibilities,
  - Liveness
  - Safety
  - permissions,
  - Activities, and
  - protocols.

Roles Schema: CUSTOMER (CUST)

| Description: | Organisation or individual requiring a service quote. |
| Protocols and Activities: | MakeCall, GiveRequirements |
| Permissions: | generates customerDetails // Owner of customer information |
| | customerRequirements // Owner of customer requirements |

Responsibilities
Liveness:

CUSTOMER = ( MakeCall. GiveRequirements )+

Safety:
- true

<table>
<thead>
<tr>
<th>GiveRequirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST</td>
</tr>
<tr>
<td>provide service details</td>
</tr>
</tbody>
</table>

Operator | Interpretation
--- | ---
\( x . y \) | x followed by y
\( x | y \) | x or y occurs
\( x^* \) | x occurs 0 or more times
\( x^+ \) | x occurs 1 or more times
\( x^\omega \) | x occurs infinitely often
\([x]\) | x is optional
\( x \parallel y \) | x and y interleaved
The Gaia design phase

- Agent types aggregate roles

- ... and have acquaintances
The Gaia2JADE process (Moraitis and Spanoudakis, 2006) proposed a method for generating code for the JADE framework.

Another work (Bittencourt et al., 2009) automated this selection using semantic web technologies but stopped at defining each Gaia activity as a generic JADE Behaviour.

ROADMAP proposed an hierarchy of roles for integrating roles.

Zambonelli et al. (2003) proposed
- the environmental model
- organizational structure with institutional rules
Agent UML (AUML)

- Is an extension of UML to capture agent technology concepts such as protocol, role, etc
- MAS development can be a top-down decomposition process starting from the roles and protocols (Odell et al., 2001)
- The Agent Interaction Protocol (AIP) model is used by a number of methodologies and practitioners becoming de-facto standard
Agent Interaction Protocol (AIP)

(a) UML 1.x with extensions

(b) UML 2.0

EASSS 2016 - N. Spanoudakis - MoD-AOSE 26 July 2013
Let’s extend the Sequence diagram metamodel:

- A message can have cardinality
- A message can have alternative paths and names
- We have roles instead of objects
An AIP metamodel
Top-down decomposition process
AUML Use Case Diagram

Customer

Order requested

Order sent

Order canceled

Process Order

Order Processing System

provider

Cancel Order

provider

Order Handler
The main idea of the vowels methodology is that a MAS is consisted of four major component types:

- the Agent
- the Environment
- Interactions
- Organization

It is a methodology that introduced these four different aspects in MAS development for the first time in a modular architecture.
PASSI models and phases
Agents Identification Diagram
Prometheus supports the development of intelligent agents, entities with
- Goals
- Beliefs
- Plans
- Events

Uses the popular JACK Intelligent Agents Platform (Winikoff, 2005) for implementation

Uses AUML AIP diagrams for protocols definition
System Overview Diagram
Agent Overview Diagram

EASSE 2016 - N. Spanoudakis - MoD-AOSE 26 July 2013
- UML and BDI-based agent modeling
- AUML can be used for defining protocols
- Interesting tools are available for a quite complex but also detailed methodology (more than 100 activities)
- A tool allows for model transformation by example for aiding the developers to automate transition between different phases models
Elements of the agent viewpoint

(Pavón, Gómez-Sanz & Fuentes, 2005)
Tropos

- Based on ideas brought by requirements-driven software engineering methods (i*)
- Allows for capturing non-functional requirements
- Allows for mapping Tropos concepts to BDI concepts
- And development using JACK
- Temporal logic-based formal Tropos
- AUML AIP for protocols design
Early requirements capture
Goal Analysis

- Decompose
- Allow for adaptive modeling
- Aid or hinder a soft goal

(Bresciani et al., 2004)
Protocols Specification

A key issue in AOSE is protocols specification
How is a protocol defined?
What are its properties?
the message types of ACLs (or performatives) are understood as speech acts (Austin 1975)

Performatives express the intent of an agent when it sends a message to another agent

FIPA defined an ACL message by the atom:

\[ \text{performative}(\text{sender}, \text{receiver}, \text{content}) \]
Moore’s Conversation Policies

- Moore (2000) makes the assumption that developers that adopt models can understand a formal specification and implement it as they see fit.

- For example, the message
  \textit{request}(sender, receiver, action)
  expresses that:
  - The receiver believes that the sender wants him to do the action
  - The receiver believes that the sender wants the receiver to want to do the action (Moore, 1999)
Definition of a CP

- According to the work of Moore, the conversation policies are implementation independent.

- A conversation policy (CP) defines:
  - how one or more conversation partners respond to messages they receive,
  - what messages a partner expects in response to a message it sends, and,
  - the rules for choosing among competing courses of action.
A conversation

- Modeled as a statechart
- Participants are orthogonal components
CPs are available in the agent’s repository.

While in a CP an unexpected message in the same conversation can cause a search for another CP.
Paurobally et al. (2004) propose that an inter-agent protocol should be:

- correct (having no contradictory states),
- unambiguous (what each agent should do),
- complete (defining all possible outcomes)
- verifiable (its properties can be verified).

They combine the

- Statecharts
- Agent Negotiation Meta-Language (ANML)
  - based on Propositional Dynamic Logic (PDL)
View Both Participants

- auctioneer starts
  - some bidder b bids (price: p);
    (p > cp max p(b))? auctioneer announces(b,p);
    cp:=p; hb:=b
  - auctioneer declares no more bids
    - auctioneer declared no more bids
      - auctioneer declares no sale
  - auctioneer declares no sale

- auction in progress
- english auction - second variant (cp,hb)

- item sold
  - auctioneer sells item (to: hb at price cp)
- auction finished
  - item not sold
Individual Point of View

english auction - second variant - bidder B's perspective(cp, hb)

EASSS 2016 - N. Spanoudakis - MoD-AOSE  26 July 2013
A commitment has (Formara and Colombetti, 2003):

- debtor
- creditor
- content (a proposition)
- a list of propositions that have to be satisfied in order for the commitment to become active;
- state \{unset, cancelled, pending, active, fulfilled, violated\}
- a timeout valid only for unset commitments

Protocols are presented as interaction diagrams

- States: System states
- Transitions: speech-acts / environment events
Colored Petri Nets (CPN)

- Petri nets are well suited for modeling the concurrent behavior of distributed systems
- Formal model allowing for verification
- Mazouzi et al. (2002) showed how to transform an AUML interaction diagram to a CPN
  - Complexity remains tractable
  - Can become very complex
  - Limited reusability and abstraction

EASSS 2016 - N. Spanoudakis - MoD-AOSE  26 July 2013
Exclusive OR part of an AIP to CPN

X: initiator agent
Y: participant agent
P: proposition or content of the message
AIP to CPN

Diagram of a process model involving Initiator and Participant states with transitions for request-when, not-understood, refuse-1, refuse-2, agree, inform-done, inform-ref, and failure. The model includes synchronization and synchronization checks.
Welcome to the Agent Systems Engineering Methodology (ASEME) Site

The Agent Systems Engineering Methodology (ASEME) is a model-driven agent-oriented software engineering methodology designed to support the development of agent-based systems. It provides a framework for modeling and designing agent systems in an integrated and systematic manner.

26 July 2013
ASEME can be considered as a Hybrid methodology using ideas from other methodologies such as Tropos, AUML and Gaia but also from the work of Moore.

We use it to demonstrate how an AOSE Methodology uses:
- The modeling language
- The process
- The tools
AMOLA Model: Requirements Analysis

- The **System Actors-Goals model (SAG)** based on the Tropos actor diagram

- And its metamodel definition using Ecore, the Eclipse Modeling Framework’s (EMF) model of a model

---

**Meeting Manager**
- Request new meeting

**Learn user habits**
- Personal Assistant

**Faculty Personnel**
- Manage meetings

---

EASSS 2016 - N. Spanoudak

26 July 2013
The **System Use Cases model (SAG)** based on the UML use cases

- Request new meeting
- Manage meetings
- Learn user habits
- Meetings Manager
- Personal Assistant

And its metamodel definition

- **UseCase**
  - name
  - specified_by

- **Role**
  - name

- **HumanRole**
  - participates_in
  - participator

- **SystemRole**
  - included_by
  - include

*Source: N. Spanoudakis*
The **Agent Interaction Protocol model (AIP)**

<table>
<thead>
<tr>
<th>Requests new meeting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Participants</strong></td>
</tr>
<tr>
<td><strong>Rules for engaging</strong></td>
</tr>
<tr>
<td><strong>Outcomes</strong></td>
</tr>
<tr>
<td><strong>Process</strong></td>
</tr>
</tbody>
</table>

**And its metamodel definition**
The **System Roles model (SRM)** based on the Gaia roles model

**Role:** Personal Assistant

**Capabilities and Protocols:**
- learn user habits, request new meeting:
  - personal assistant, ...

**Activities:**
- learn user preference, update user preferences, ...

**Liveness:**
- personal assistant = request new meeting ||
  - learn user habits
  - learn user habits = learn user preference.
  - update user preferences
  ...

And its metamodel
A grammar defines the syntax of a language, such is EBNF (Wirth, 1996)

BnfGrammar $\rightarrow \{ \text{Rule} \}$

Rule $\rightarrow$ NonTerminal $\rightarrow$ Expression

Expression $\rightarrow$ Alternative | Composition | PExpression

PExpression $\rightarrow$ Optional | AtLeastOne | Arbitrary | Symbol | ( Expression )

Alternative $\rightarrow$ PExpression | (PExpression | Alternative)

Composition $\rightarrow$ PExpression +

Optional $\rightarrow$ [ Expression ]

AtLeastOne $\rightarrow$ PExpression +

Arbitrary $\rightarrow$ { Pexpression }

Symbol $\rightarrow$ Terminal | NonTerminal

Defines itself
### AMOLA: Liveness

#### EBNF syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>liveness</code></td>
<td>→ { formula }</td>
</tr>
<tr>
<td><code>formula</code></td>
<td>→ leftHandSide = expression</td>
</tr>
<tr>
<td><code>leftHandSide</code></td>
<td>→ string</td>
</tr>
<tr>
<td><code>expression</code></td>
<td>→ term</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>parallelExpression</code></td>
<td>→ term</td>
</tr>
<tr>
<td><code>orExpression</code></td>
<td>→ term</td>
</tr>
<tr>
<td><code>sequentialExpression</code></td>
<td>→ term . term . ... . term</td>
</tr>
<tr>
<td><code>term</code></td>
<td>→ basicTerm</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>basicTerm</code></td>
<td>→ string</td>
</tr>
<tr>
<td><code>number</code></td>
<td>→ digit</td>
</tr>
<tr>
<td><code>digit</code></td>
<td>→ 1</td>
</tr>
<tr>
<td><code>string</code></td>
<td>→ letter</td>
</tr>
<tr>
<td><code>letter</code></td>
<td>→ a</td>
</tr>
</tbody>
</table>

#### Semantics

<table>
<thead>
<tr>
<th>Operator</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x . y</code></td>
<td>x followed by y</td>
</tr>
<tr>
<td>`x</td>
<td>y`</td>
</tr>
<tr>
<td><code>x*</code></td>
<td>x occurs 0 or more times</td>
</tr>
<tr>
<td><code>x+</code></td>
<td>x occurs 1 or more times</td>
</tr>
<tr>
<td><code>x ω</code></td>
<td>x occurs infinitely often</td>
</tr>
<tr>
<td><code>[x]</code></td>
<td>x is optional</td>
</tr>
<tr>
<td>`x</td>
<td></td>
</tr>
<tr>
<td>`</td>
<td>x ω</td>
</tr>
</tbody>
</table>
The Inter (EAC) and Intra-Agent Control (IAC) based on statecharts

And its metamodel definition
AMOLA Design: Statecharts

- Syntax (Harel and Naamad, 1996):
  - Start state
  - End state
  - Basic state
  - And state
  - Or state
  - Condition state
  - History state
  - Transition
  - Transition Expression: $e[c]/a$
Example Statechart for a reservation

Assume that once a reservation becomes accepted, then it is in a waiting state until the client picks up the vehicle. The client then returns the vehicle and completes the lifecycle of the reservation. Before the client picks up the vehicle he can cancel the reservation.
The transformations are fully automated
A model of a previous phase is transformed to an instance of the model of the next phase (initial model)
Editing and refinement of models is done by the engineer and produces the refined model
Using the OMG Software Process Engineering Metamodel (SPEM v2.0)
Create and Edit the SAG Model

Meetings Manager

Request new meeting

Personal Assistant

Learn user habits
The SAG 2 SUC Transformation

Meetings Manager → Request new meeting → Personal Assistant

Learn user habits

95
The SAG 2 SUC Transformation

Meetings Manager

Request new meeting

Personal Assistant

Learn user habits

Meeting Manager

Request new meeting

Personal Assistant

Learn user habits
SUC 2 SRM Transformation

Meetings Manager
- Receive new request
- Send new request
- Receive new results
- Send new results

Request new meeting
- <<include>>
- <<include>>
- <<include>>
- <<include>>

Personal Assistant
- Learn user habits
- Learn user preference
- Update user preferences
- <<include>>
- <<include>>
- <<include>>
SUC 2 SRM Transformation

Role:
Capabilities and Protocols:

Activities:

- Request new meeting
- Send new request
- Receive new results
- Learn user habits
- Learn user preference
- Update user preferences

26 July 2013
Role: Personal Assistant

Capabilities and Protocols:
manage meetings, learn user habits,
negotiate meeting date, request change meeting, request new meeting

Activities:
get user request, read schedule, show results,
learn user preference, update user preferences,
send change request, receive change results, send new request, receive new results,
receive proposed date, decide response, send results, receive outcome, update schedule

Liveness:
Refine SRM

Role: Personal Assistant

Capabilities and Protocols:
manage meetings, learn user habits,
negotiate meeting date, request change meeting, request new meeting

Activities:
get user request, read schedule, show results,
learn user preference, update user preferences,
send change request, receive change results, send new request, receive new results,
receive proposed date, decide response, send results, receive outcome, update schedule

Liveness:
personal assistant = ?
manage meetings = get user request ? read schedule ? request change meeting ? request new meeting? show results
learn user habits = learn user preference ? update user preferences
### Role: Personal Assistant

<table>
<thead>
<tr>
<th></th>
<th>Request new meeting</th>
<th>Negotiate meeting date</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Participants</strong></td>
<td>Personal Assistant</td>
<td>Meetings Manager</td>
</tr>
<tr>
<td><strong>Rules for engaging</strong></td>
<td>He needs to create a meeting</td>
<td>-</td>
</tr>
<tr>
<td><strong>Outcomes</strong></td>
<td>He has received a confirmation of the meeting creation</td>
<td>-</td>
</tr>
<tr>
<td><strong>Process</strong></td>
<td>request new meeting = send new request. receive new results</td>
<td>request new meeting = receive new request. send new results</td>
</tr>
</tbody>
</table>

Learn user habits = Learn user preference ? update user preferences

Request new meeting = *send new request. receive new results*

Request change meeting = *send change request. receive change results*

Negotiate meeting date = *receive proposed date. (decide response. send results. receive outcome)+*

? update schedule

---

Refine SRM

Get information from existing AIP model
Refine SRM

Role: Personal Assistant

Capabilities and Protocols:
manage meetings, learn user habits,
negotiate meeting date, request change meeting, request new meeting

Activities:
get user request, read schedule, show results,
learn user preference, update user preferences,
send change request, receive change results, send new request, receive new results,
receive proposed date, decide response, send results, receive outcome, update schedule

Liveness:
personal assistant = ?
manage meetings = get user request ? read schedule ? request change meeting ? request new meeting? show results
learn user habits = learn user preference ? update user preferences
request new meeting = send new request. receive new results
request change meeting = send change request. receive change results
negotiate meeting date = receive proposed date. (decide response. send results. receive outcome)+
? update schedule
Refine SRM

Role: Personal Assistant

Capabilities and Protocols:
manage meetings, learn user habits,
negotiate meeting date, request change meeting, request new meeting

Activities:
get user request, read schedule, show results,
learn user preference, update user preferences,
send change request, receive change results, send new request, receive new results,
receive proposed date, decide response, send results, receive outcome, update schedule

Liveness:
personal assistant = (manage meetings. learn user habits) \( \omega \) | | (negotiate meeting date) \( \omega \)
manage meetings = get user request. (read schedule | request change meeting | request new meeting). show results
learn user habits = learn user preference. update user preferences
request new meeting = send new request. receive new results
request change meeting = send change request. receive change results
negotiate meeting date = receive proposed date. (decide response. send results. receive outcome)+. update schedule
ASEME Analysis phase: Functionality Graph

Capabilities
- handle dangerous situation
- service user
- learn user habits
- request for services
- communicate

Activities
- determine user condition
- invoke heart rate service
- get user coordinates
- present information to the user
- get user order
- get user preferences
- update user preferences
- learn user preference
- search broker
- send message
- receive message

Functionalities
- algorithm
- invoke OSGi service
- Human-Machine Interface (HMI)
- MIDP record technology
- machine learning
- JADE FIPA DF
- JADE FIPA AMS

Legend: <decomposition> <uses>

EASSS 2016 - N. Spanoudakis - MoD-AOSE 26 July 2013
Design phase

- In the design phase liveness formulas are transformed to statecharts

- Then,
  - The variables (in/out params) of each state activity are defined (corresponding to beliefs)
    - Defined as objects
  - The transition expressions are defined $e[c]/a$
    (corresponding to plan choosing based on beliefs)
    - Define events and conditions for enabling plan execution
    - Update beliefs
Transformation templates

- Are applied recursively in formulas
SRM 2 IAC Transformation

Liveness:

personal assistant = (manage meetings. learn user habits)ω || (negotiate meeting date)ω

learn user habits = learn user preference. update user preferences

26 July 2013
The Inter- and Intra-Agent Control

- The inter-agent control (EAC) is a statechart defining the parallel behaviour of two or more roles.

- The intra-agent control (IAC) coordinates the interactions between the agent’s capabilities (or modules).

- Every role in an EAC can be merged in the IAC model as-is and it can be refined:
  - By turning a state to a superstate with substates.

- IAC allows the parallel execution of multiple protocols.
Integrating an EAC model in an IAC model
Integrating an EAC model in an IAC model
Integrating an EAC model in an IAC model
Integrating an EAC model in an IAC model

personal assistant

manage meetings
- get user request
- request new meeting
- send new request
- receive new results
- read schedule
- request change meeting
- send change request
- receive change results
- show results
- receive results

learn user habits
- learn user preference
- update user preferences

negotiate meeting date
- receive proposed date
- decide response
- send results
- receive outcome
Support for sub-dialogs

- Personal assistant
  - Learn user habits
  - Update user preferences
  - Manage meetings
    - Get user request
    - Show results
    - Request new meeting
      - Send new request
      - Receive new results
    - Request change meeting
      - Send change request
      - Receive change results
  - Negotiate meeting date
    - Receive proposed date
    - Decide response
      - Send results
      - Receive outcome
    - Provide schedule information
      - Receive schedule request
      - Read schedule
      - Send schedule
  - Read schedule
  - Send schedule

Support for embedded dialogs

Request for Services

service requester (sr)

request ≠ ∅

send request message

Request(sr,sp, request) /t1 = 10000

receive response message

Inform(sp,sr,results) ∨ Refuse(sp,sr,results) ∨ Failure(sp,sr,results) ∨ timeout(t1) = True

service provider (sp)

receive request message

Request(sr,sp, request) /t2 = 10000

process request

send response message

results ≠ ∅

Inform(sp,sr,results) ∨ Refuse(sp,sr,results) ∨ Failure(sp,sr,results) ∨ timeout(t2) = True
Support for embedded dialogs

broker

request for services SP

receive request message

send response message

process request

service match

invoke data management

request for services SR

receive message

send message

request for services SP\(\omega\)

... 10 times

EASSS 2016 - N. Spanoudakis - MoD-AOSE

26 July 2013
For each node in S
   If node is root then create Agent class
   Else if $\lambda$(node)="BASIC" then create a SimpleBehaviour
   Else if $\lambda$(node)="AND" then create a ParallelBehaviour ($\mid \mid$ gaia op.)
   Else if sons(node).size() = 2 and there exists transitionExpression $x \mid (node.2, x, node.2)$ belongs to $\delta$ then create a CyclicBehaviour ($\omega$ gaia op.)
   Else if sons(node).size() = 3 and there exists transitionExpression $x \mid (node.2, x, node.2)$ belongs to $\delta$ then create a SimpleBehaviour ($+ gaia$ op.)
   Else if there exists $x$ belongs to sons(node) $\mid \lambda(x)=$CONDITION then
      If sons(node).size() = 4 then create a SimpleBehaviour ($\ast$ gaia op.)
      Else create a SequentialBehaviour ($\mid gaia$ op.)
      End if
   Else create a SequentialBehaviour ($.$ gaia op.)
   End if
End for
We have three types of tools:

- Graphical editors
- Transformations
- Integrated Development Environment
Transformations using ATL

Ecore.ecore

SAG.ecore

ATL

SUC.ecore

SAG2SUC.atl

SAGModel.xmi

M2M

SUCModelInitial.xmi

XMI: XML Metadata Interchange

EASSS 2016 - N. Spanoudakis - MoD-AOSE 26 July 2013
M2M transformation using ATL (Jouault & Kurtev, 2006)
Code generation using EMF Xpand

```
«DEFINE javaClass FOR IAC::Model»
  «LET name AS packageName»
    «EXPAND nodeClass(packageName, this) FOREACH nodes»
    «EXPAND variableHolderClass(packageName, this) FOREACH variables»
  «ENDLET»
«ENDDEFINE»

«DEFINE nodeClass(String packageName, Model model) FOR IAC::Node»
  «IF label.compareTo("0")==0»
    «EXPAND agentClass(packageName, model) FOR this»
  «ELSE»
    «EXPAND behaviourClass(packageName, model) FOR this»
  «ENDIF»
«ENDDEFINE»

«DEFINE agentClass(String packageName, Model model) FOR IAC::Node»
  «FILE classFileName()»
    package «packageName»;
    import jade.core.Agent;
    public class «className()» extends Agent{
      public void setup(){
        //add behavior
        addBehaviour(«getAgentBehaviour(this,model)»);
      }
      protected void takeDown() {
        doDelete();
      }
    }
  «ENDFILE»
«ENDDEFINE»
```
JADE source code generation
Automatic code generation

- Targeting three platforms (Spanoudakis and Moraitis, 2010; Paraschos et al., 2012)

<table>
<thead>
<tr>
<th>Project – Case study</th>
<th>ASK-IT</th>
<th>MARKET-MINER</th>
<th>Robocup player</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation Platform</td>
<td>JADE</td>
<td>Rhapsody</td>
<td>Monas</td>
</tr>
<tr>
<td>Total project code</td>
<td>160,241 (bytes)</td>
<td>179,750 (bytes)</td>
<td>826 (lines)</td>
</tr>
<tr>
<td>Automatically generated (bytes)</td>
<td>41,759 (bytes)</td>
<td>97,631 (bytes)</td>
<td>805 (lines)</td>
</tr>
<tr>
<td>Manually written Java code (bytes)</td>
<td>118,482 (bytes)</td>
<td>60,365 (bytes)</td>
<td>21 (lines)</td>
</tr>
<tr>
<td>Manually written non-java code (bytes)</td>
<td>-</td>
<td>21,754</td>
<td>-</td>
</tr>
<tr>
<td>Percentage of automatically generated code</td>
<td>26.06%</td>
<td>54.31%</td>
<td>97.45%</td>
</tr>
</tbody>
</table>
More aspects of the truth...

<table>
<thead>
<tr>
<th>Table 2: Comparison of development methods.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric / Concept</td>
</tr>
<tr>
<td>Run-Time Performance (load)</td>
</tr>
<tr>
<td>Development Phase</td>
</tr>
<tr>
<td>Total Development Time</td>
</tr>
<tr>
<td>Lines of Source Code</td>
</tr>
<tr>
<td>Auto-Generated Lines</td>
</tr>
<tr>
<td>Number of State Variables</td>
</tr>
<tr>
<td>Debugging Phase</td>
</tr>
<tr>
<td>Total Debugging Time</td>
</tr>
<tr>
<td>Number of Bugs</td>
</tr>
</tbody>
</table>

(Paraschos et al., 2012)
SAG model graphical editor

- Using EMF Graphical Modeling Framework (GMF)

(Argyriou, 2010)
SUC model graphical editor init
AIP model

- AssignPills
- Interface
  - A web service invocation occurred
  - not noting that a user with a specific username has been assigned new pills.
  - Notified the relevant PA.
  - WaitForNewPillPrescriptionService.
  - SelectTargetPAAgent.
  - SendNewPillPrescriptionRequest
- PersonalAssistant
  - Waiting to receive information about the obligations of the monitored user.
  - The new pill prescription has been recorded and possible agent action has been triggered.
  - ReceiveNewPillPrescriptionRequest.
  - UpdateUserSchedule
A graphical view of SRM

- The functionality graph

- Functionality sending a standard FIPA ACL message
- Interfaces with external systems
How to evaluate a methodology?

- It supports new features
- It is evaluated against other successful ones over a set of criteria (danger of bias)
- Through a successful case study for a real world system
- Compare it against others through a benchmark or common application (conference management system)
- De facto - Through acceptance by practitioners
During a lab session 28 undergraduate students used KSE to design an attacker behavior for RoboCup SPL. All of them completed the task and evaluated KSE positively.
ASEME usage

- **Built systems**
  - The MarketMiner Product Pricing Agent (Spanoudakis and Moraitis, 2009)
  - A Hybrid Wind Turbine Monitoring System (Smarsly and Hartmann, 2010)
  - The HERA Ambient Assisted Living System (Marcais et al., 2011)
  - Robotic Behaviors for the RoboCup Competition (Kouretes TDP 2011; 2012; 2013)
  - A Structural Health Monitoring System (Smarsly and Law, 2012)

- **Has been extended for Application Domains**
  - Agent-Oriented Monitoring Systems Engineering (AGEME) (Smarsly and Hartmann, 2010)
  - Kouretes Statechart Editor (KSE) (Topalidou-Kyniazopoulou et al., 2012)
Method Engineering

Where a team customizes the software process
Different methodologies have their strengths and weaknesses

An engineer can need to combine method fragments from different methodologies

Method fragments are reusable methodological parts that can be used by engineers in order to produce a new design process for a specific situation (Cossentino et al., 2007)

This allows a development team to come up with a hybrid methodology that will support the needs of specific programming and modeling competencies
Transform the SRM analysis model to the IAC design model
Transform the SRM analysis model to the IAC design model
Transform the SRM analysis model to the IAC design model
How to combine fragments?

- Define the scenario
- Define input/output models
- Apply transformations

- E.g.
  - Use the **Gaia roles model** outputted by my Gaia analysis phase and then use the ASEME design phase (starting with **the previous fragment**) as I want to develop JADE agents
Mapping Metamodels

- Gaia metamodel

- ASEME SRM
Transformation

- There are procedural or declarative languages to write transformation models Using the Eclipse Modeling Framework

  - **Atlas Transformation Language (ATL)**
    ```java
    rule GaiaRole2SRMRole{
      from gaiaRole : Gaia!Role
      to srmRole : SRM!Role
      (srmRole.name <= gaiaRole.name)
    }
    ```

  - **Java**
    ```java
    SRM.Role newRole = new SRM.Role();
    newRole.name = gaiaRole.name;
    ```
Hahn, Madrigal-Mora and Fischer proposed in 2009 the PIM4Agents metamodel

PIM4Agents abstracts from existing agent-oriented methodologies, programming languages, and platforms and could thus be considered as platform-independent

A method engineer can transform his/her design model to PIM4Agents and then have an automatic implementation in JACK or JADE platforms
The PIM4Agents concept

(Hahn, Madrigal-Mora and Fischer, 2009)
143

IDEs

• ASEME IDE

• KSE
The ASEME IDE Dashboard
The Kouretes Statechart Editor (KSE)

(Topalidou-Kyniazopoulou et al., 2012)
Live and on-demand model's validation

Validation of the model according to

- Harel’s statechart rules
- transition expression's syntax
- variable's type and name syntax
A KSE Agent

- Robocup 2012 goalie

(Topalidou-Kyniazopoulou et al., 2012)
RoboCup 2012 Goalie

EASSS 2016 - N. Spanoudakis - MoD-AOSE 26 July 2013
Evolution of MAS Modeling Languages and Methodologies

(Estefania Argente, 2009)
Research Directions

- Automation in models refinement
  - using knowledge and repositories of models
- Self-Assessment and Healing Agent Capability
  - by monitoring the intra-agent control model execution
- Adaptation automation
  - Using evolutionary techniques and fine tuning
- Prove properties of ASEME
THANK YOU!

Questions????

Find out more about my work and CV at: http://users.isc.tuc.gr/~nispanoudakis

Get in touch for ideas related to the above topics.


Hahn C., Madrigal-Mora C., Fischer K. (2009), A platform-independent metamodel for multiagent systems, Autonomous Agents and Multi-Agent Systems,18(2), pp 239–266


Kleppe A., (2008), Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley Professional


EASSS 2016 - N. Spanoudakis - MoD-AOSE 26 July 2013


