

Agent-Oriented Software Engineering

Multiagent Systems LM
Sistemi Multiagente LM

Ambra Molesini & Andrea Omicini
{ambra.molesini, andrea.omicini}@unibo.it

Ingegneria Due
ALMA MATER STUDIORUM—Università di Bologna a Cesena

Academic Year 2010/2011



Outline of Part I: General Concepts

- 1 Software Engineering
- 2 Software Process
- 3 Methodologies
- 4 Meta-Models
 - SPEM
 - OPF & OPEN
- 5 Method Engineering
 - Method Fragment Representation
 - Method Assembly



Outline of Part II: AOSE

6 AOSE

7 Agent Oriented Methodologies

- MAS Meta-models
- AOSE Methodologies: An Overview
- Methodologies Documentation
- Methodology Challenges

8 Agent Oriented Situational Method Engineering

- Method Fragment Representation
- PRoDe: A Process for the Design of Design Processes
- Method Fragment extraction and Repository creation
- Result Evaluation



Outline of Part III: Research Directions

9 Research Directions & Vision

10 Conclusions



Part I

General Concepts



Outline

- 1 Software Engineering
- 2 Software Process
- 3 Methodologies
- 4 Meta-Models
 - SPEM
 - OPF & OPEN
- 5 Method Engineering
 - Method Fragment Representation
 - Method Assembly



Software

- Software is **abstract** and **intangible** [Sommerville, 2007]:
 - it is not constrained by materials, or governed by physical laws, or by manufacturing process
- On the one hand, this simplifies software engineering as there are no physical limitations on the potential of software
- On the other hand, the lack of natural constraints means that software can easily become extremely complex and hence very difficult to understand
- So, software engineers should
 - adopt a **systematic and organised approach** to their work
 - use **appropriate tools and techniques** depending on the problem to be solved, the development constraints and the resources available



Software Engineering

What is Software Engineering?

Software Engineering is an **engineering discipline** concerned with theories, methods and tools for professional software development [Sommerville, 2007]

What is the aim of Software Engineering?

Software Engineering is concerned with all aspects of **software production** from the early stage of system specification to the system maintenance / incremental development after it has gone into use [Sommerville, 2007]



Software Engineering: Concerns

- There is a need to *model* and *engineer* both
 - the **development process**
 - Controllable, well documented, and reproducible ways of producing software
 - the **software**
 - ensuring a given level of quality—e.g., % of errors and performances)
 - enabling reuse, maintenance, and incremental development
- This requires suitable
 - abstractions
 - tools



Software Engineering Abstractions

- Mostly, software deals with *abstract entities*, having a real-world counterpart
 - not necessarily a concrete one
 - such as numbers, dates, names, persons, documents. . .
- In what terms should we model them in software?
 - data, functions, objects, agents. . .
 - i.e., what are the **abstractions** that we could / should use to model software?
- Abstractions might depend on the available technologies
 - we may adopt OO abstractions for OO programming environments
 - but this is not mandatory: we may use OO abstractions just because they are better, even for COBOL programming environments



Tools

- *Notation tools* represent the outcomes of the software development
 - diagrams, equations, figures. . .
- *Formal models* prove properties of software prior to the development
 - lambda-calculus, pi-calculus, Petri nets. . .
- *CASE tools* are based on notations and models to facilitate activities
 - simulators



Software Engineering & Computer Science

- Computer science is concerned with theory and fundamentals—*modelling* computational systems
- Software engineering is concerned with the practicalities of developing and delivering useful software—*building* computational systems
- Deep knowledge of computer science is essential for software engineering in the same way that deep knowledge of physics is essential for electric engineers
- Ideally, all of software engineering should be underpinned by theories of computer science. . . but this is not the case, in practice
- Software engineers must often use *ad hoc* approaches to developing software systems
- Elegant theories of computer science cannot always be applied to real, complex problems that require a software solution [Sommerville, 2007]



Software Engineering & System Engineering

- System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering
- System engineers are involved in system specification, architectural design, integration and deployment—they are less concerned with the engineering of the system components
- Software engineering is part of this process concerned with developing the software infrastructure, control, applications and databases in the system [Sommerville, 2007]



Outline

- 1 Software Engineering
- 2 Software Process**
- 3 Methodologies
- 4 Meta-Models
 - SPEM
 - OPF & OPEN
- 5 Method Engineering
 - Method Fragment Representation
 - Method Assembly



Development Process

Development Process [Cernuzzi et al., 2005]

- The **development process** is an ordered set of steps that involve all the activities, constraints and resources required to produce a specific desired output satisfying a set of input requirements
- Typically, a process is composed by different stages/phases put in relation with each other
- Each stage/phase of a process identify a portion of work definition to be done in the context of the process, the resources to be exploited to that purpose and the constraints to be obeyed in the execution of the phase
- Case by case, the work in a phase can be very small or more demanding
- Phases are usually composed by a set of activities that may, in turn, be conceived in terms of smaller atomic units of work (steps)

Software Process

Software Process [Fuggetta, 2000]

The **software development process** is the coherent set of policies, organisational structures, technologies, procedures and deliverables that are needed to conceive, develop, deploy and maintain a *software* product



Software Process: Concepts

The software process exploits a number of contributions and concepts [Fuggetta, 2000]

Software development technology — Technological support used in the process. Certainly, to accomplish software development activities we need tools, infrastructures, and environments

Software development methods and techniques — Guidelines on how to use technology and accomplish software development activities. The methodological support is essential to exploit technology effectively

Organisational behavior — The science of organisations and people.

Marketing and economy — Software development is not a self-contained endeavor. As any other product, software must address real customers' needs in specific market settings.

Software Process: Activities

Generic activities in all software processes are [Sommerville, 2007]:

Specification — What the system should do and its development constraints

Development — Production of the software system

Validation — Checking that the software is what the customer wants

Evolution — Changing the software in response to changing demands



The Ideal Software Process

The Ideal Software Process?

There is no an ideal process

[Sommerville, 2007]



Many Sorts of Software Processes

- Different types of systems require different development processes [Sommerville, 2007]
 - real time software in aircraft has to be completely specified before development begins
 - in e-commerce systems, the specification and the program are usually developed together
- Consequently, the generic activities, specified above, may be organised in different ways, and described at different levels of details for different types of software
- The use of an inappropriate software process may reduce the quality or the usefulness of the software product to be developed and/or increased



Software Process Model

- A **Software Process Model** is a simplified representation of a software process, presented from a specific perspective [Sommerville, 2007]
- A process model prescribes which **phases** a process should be organised around, in which order such phases should be executed, and when interactions and coordination between the work of the different phases should be occur
- In other words, a process model defines a *skeleton*, a *template*, around which to organise and detail an actual process



Software Process Model: Examples

- Examples of process models are

Workflow model — this shows sequence of activities along with their inputs, outputs and dependencies

Activity model — this represents the process as a set of activities, each of which carries out some data transformation

Role/action model — this depicts the roles of the people involved in the software process and the activities for which they are responsible



Generic Software Process Models

- Generic process models

Waterfall — separate and distinct phases of specification and development

Iterative development — specification, development and validation are interleaved

Component-based software engineering — the system is assembled from existing components



Outline

- 1 Software Engineering
- 2 Software Process
- 3 Methodologies**
- 4 Meta-Models
 - SPEM
 - OPF & OPEN
- 5 Method Engineering
 - Method Fragment Representation
 - Method Assembly



Methodologies vs. Methods: General Issue

- Disagreement exists regarding the relationship between the terms *method* and *methodology*
- In common use, methodology is frequently substituted for method; seldom does the opposite occur
- Some argue this occurs because methodology sounds more scholarly or important than method
- A footnote to [methodology](#) in the 2006 American Heritage Dictionary notes that
 - *the misuse of methodology obscures an important conceptual distinction between the tools of scientific investigation (properly methods) and the principles that determine how such tools are deployed and interpreted (properly methodologies)*



Methodologies vs. Methods in Software Engineering

- In Software Engineering the discussion continues. . .
 - Some authors argue that a software engineering method is a recipe, a series of steps, to build software, while a methodology is a codified set of recommended practices. In this way, a software engineering method could be part of a methodology
 - Some authors believe that in a methodology there is an overall philosophical approach to the problem. Using these definitions, Software Engineering is rich in methods, but has fewer methodologies



Method

Method [Cernuzzi et al., 2005]

- A method prescribes a way of performing some kind of activity within a process, in order to properly produce a specific output (i.e., an artefact or a document) starting from a specific input (again, an artefact or a document).
- Any phases of a process, to be successfully applicable, should be complemented by some methodological guidelines (including the identification of the techniques and tools to be used, and the definition of how artifacts have be produced) that could help the involved stakeholders in accomplishing their work according to some defined best practices



Methodology

Methodology [Ghezzi et al., 2002]

- A methodology is a collection of methods covering and connecting different stages in a process
- The purpose of a methodology is to prescribe a certain coherent approach to solving a problem in the context of a software process by preselecting and putting in relation a number of methods
- A methodology has two important components
 - one that describe the process elements of the approach
 - one that focuses on the work products and their documentation



Methodologies vs. Software Process

- Based on the above definitions, and comparing software processes and methodologies, we can find some common elements in their scope [Cernuzzi et al., 2005]
 - both are focusing on what we have to do in the different activities needed to construct a software system
 - however, while the software development process is more centered on the global process including all the stages, their order and time scheduling, the methodology focuses more directly on the specific techniques to be used and artifacts to be produced
- In this sense, we could say that methodologies focus more explicitly on *how* to perform the activity or tasks in some specific stages of the process, while processes may also cover more general management aspects, e.g., basic questions about *who* and *when*, and *how much*



Outline

- 1 Software Engineering
- 2 Software Process
- 3 Methodologies
- 4 Meta-Models**
 - SPEM
 - OPF & OPEN
- 5 Method Engineering
 - Method Fragment Representation
 - Method Assembly



Meta-models

Definition

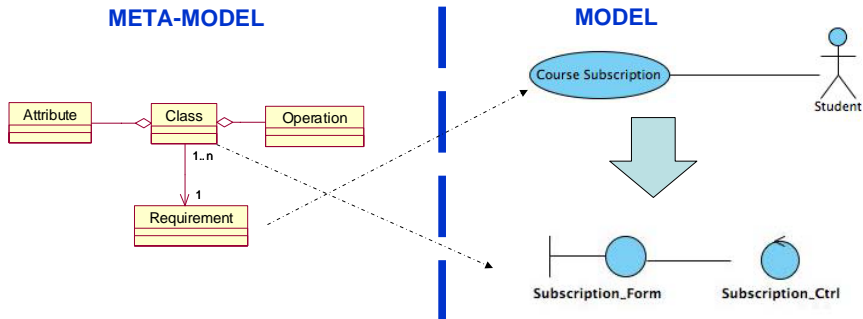
Meta-modelling is the analysis, construction and development of the frames, rules, constraints, models and theories applicable and useful for the modelling in a predefined class of problems

- A meta-model enables checking and verifying the completeness and expressiveness of a methodology by understanding its deep semantics, as well as the relationships among concepts in different languages or methods
- the process of designing a system consists of instantiating the system meta-model that the designers have in their mind in order to fulfill the specific problem requirements [Bernon et al., 2004]



Software Design: The Role of System Meta-model

- Designing a software means instantiating its meta-model



Using Meta-models

- Meta-models are useful for specifying the concepts, rules and relationships used to define a family of related methodologies
- Although it is possible to describe a methodology without an explicit meta-model, formalising the underpinning ideas of the methodology in question is valuable when checking its consistency or when planning extensions or modifications
- A good meta-model must address all of the different aspects of methodologies, i.e. the process to follow and the work products to be generated
- In turn, specifying the work products that must be developed implies defining the basic modelling building blocks from which they are built
- Meta-models are often used by methodologists to construct or modify methodologies



Meta-models & Methodologies

- Methodologies are used by software development teams to construct software products in the context of software projects
- Meta-model, methodology and project constitute, in this approach, three different areas of expertise that, at the same time, correspond to three different levels of abstraction and three different sets of fundamental concepts
- As the work performed by the development team at the project level is constrained and directed by the methodology in use, the work performed by the methodologist at the methodology level is constrained and directed by the chosen meta-model
- Traditionally, these relationships between *modelling layers* are seen as instance-of relationships, in which elements in one layer are instances of some element in the layer above



Meta-model & Processes

- The use of meta-models to underpin object-oriented processes was pioneered in the mid-1990s by the OPEN Consortium [OPEN Working Group, 1997] leading to the current version of the OPEN Process Framework (OPF) and to the recent standard “Software Engineering Metamodel for Development Methodologies” ISO/IEC 24744 ¹
- The Object Management Group (OMG) then issued a request for proposals for what turned into the SPEM (Software Processing Engineering Metamodel) [Object Management Group, 2008]

¹See http://www.iso.org/iso/catalogue_detail.htm?csnumber=38854



Outline

- 1 Software Engineering
- 2 Software Process
- 3 Methodologies
- 4 Meta-Models**
 - SPEM
 - OPF & OPEN
- 5 Method Engineering
 - Method Fragment Representation
 - Method Assembly

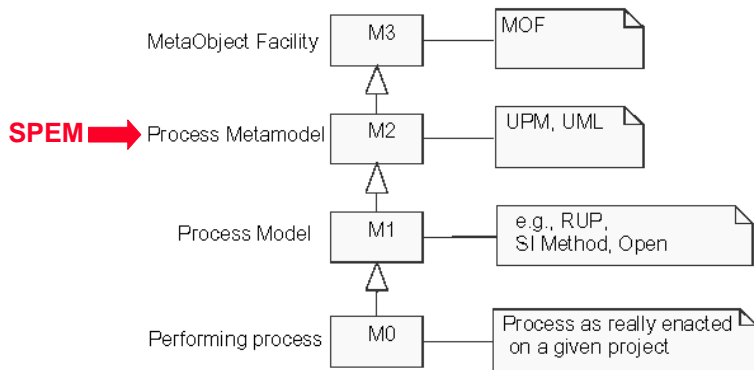


Software Process Engineering Meta-model (SPEM)

- SPEM (Software Process Engineering Meta-model) [Object Management Group, 2008] is an OMG standard object-oriented meta-model defined as an UML profile and used to describe a concrete software development process or a family of related software development processes
- SPEM is based on the idea that a software development process is a collaboration between active abstract entities called **roles** which perform operations called **activities** on concrete and real entities called **work products**
- Each role interacts or collaborates by exchanging work products and triggering the execution of activities
- The overall goal of a process is to bring a set of work products to a well-defined state



SPEM: Level of Abstraction

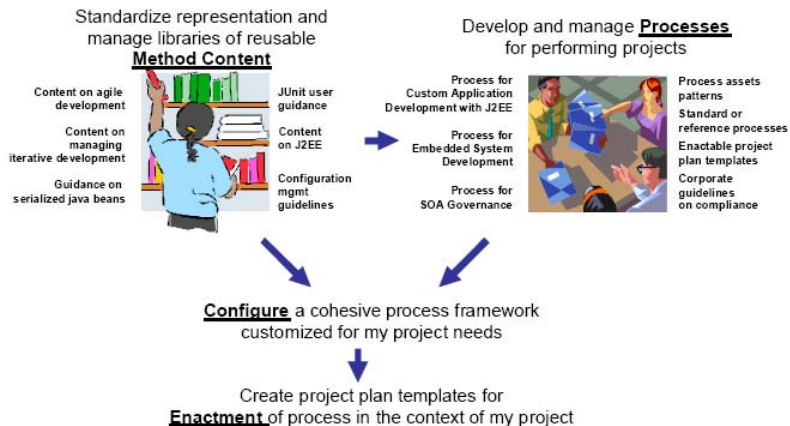


SPEM: Goals

- The goals of SPEM are to
 - support the representation of one specific development process
 - support the maintenance of several unrelated processes
 - provide process engineers with mechanisms to consistently and effectively manage whole families of related processes promoting process reusability



SPEM I

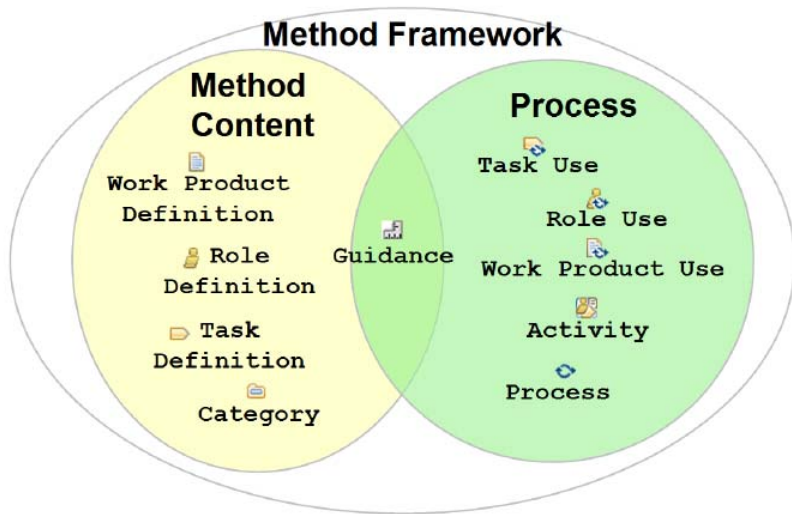


SPEM II

- Clear separation between
 - Method Contents** — introduce the concepts to document and manage development processes through natural language description
 - Processes** — defines a process model as a breakdown or decomposition of nested *Activities*, with the related *Roles* and input / output *Work Products*
 - Capability patterns** — reusable best practices for quickly creating new development processes

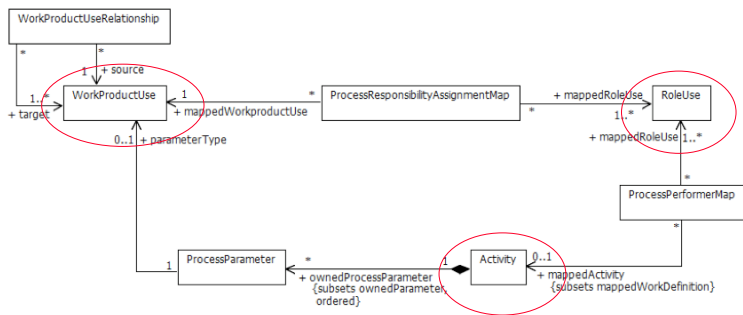


SPEM: Method Content and Process



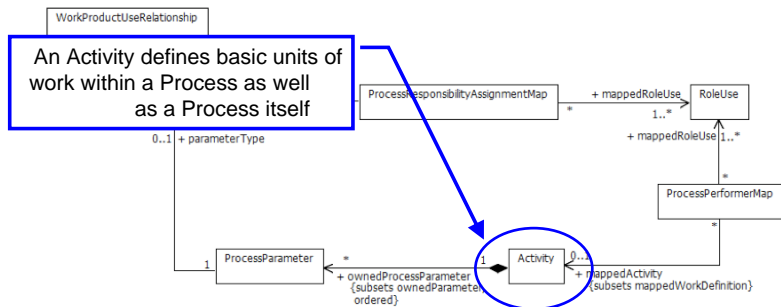
Roles, Activities & Work Products

- A software development process is seen as a collaboration between abstract active entities called **process roles** that perform operations called **activities** on concrete, tangible entities called **work products**



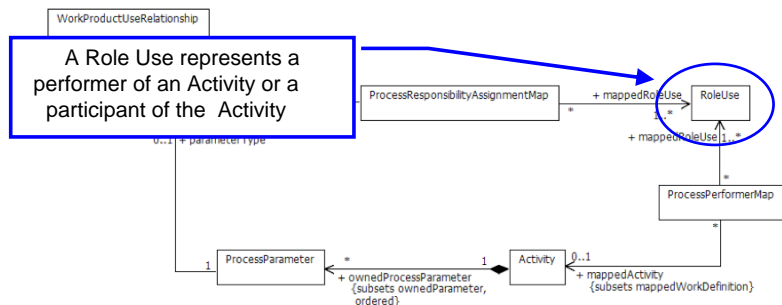
Roles, Activities & Work Products

- A software development process is seen as a collaboration between abstract active entities called **process roles** that perform operations called **activities** on concrete, tangible entities called **work products**



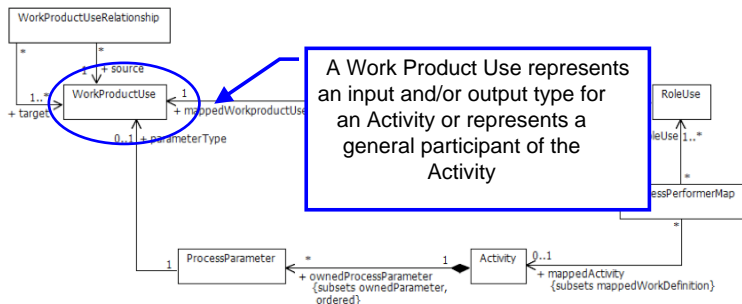
Roles, Activities & Work Products

- A software development process is seen as a collaboration between abstract active entities called **process roles** that perform operations called **activities** on concrete, tangible entities called **work products**















Roles, Activities & Work Products

- A software development process is seen as a collaboration between abstract active entities called **process roles** that perform operations called **activities** on concrete, tangible entities called **work products**

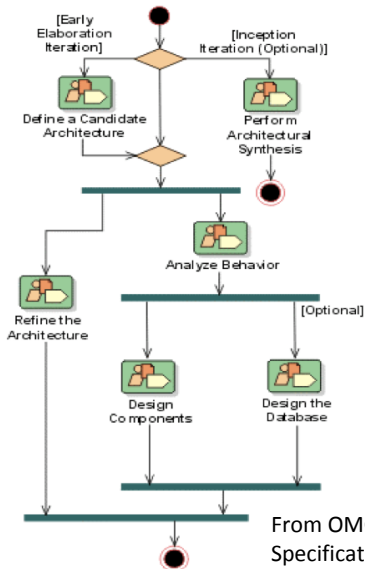


SPEM Notation

Stereotype	Symbol
Activity	
Category	
Composite role and Team	
Guidance	
Milestone	
Process	
Process Component	
Process Pattern	
Role Definition and Use	
Task Definition and Use	
Tool Definition	
WorkProduct Definition and Use	



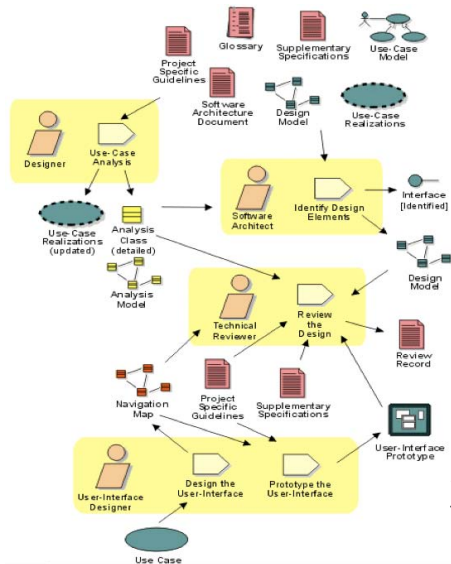
SPEM: WorkFlow Diagram



From OMG SPEM 2.0
Specifications



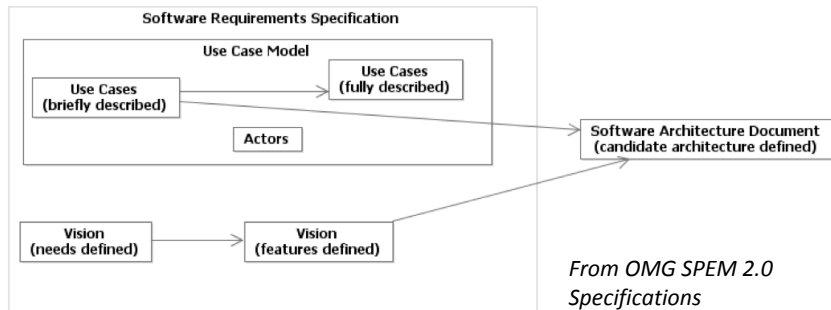
SPEM: Activity Details Diagram



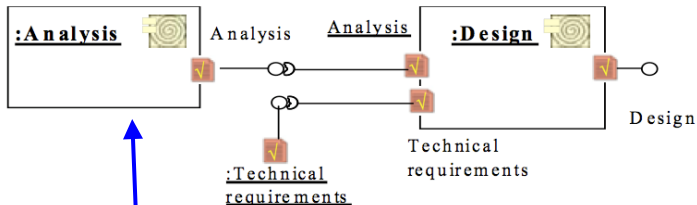
*From OMG SPEM 2.0
Specifications*



SPEM: Work Product Dependency Diagram



SPEM: Process Component Diagram

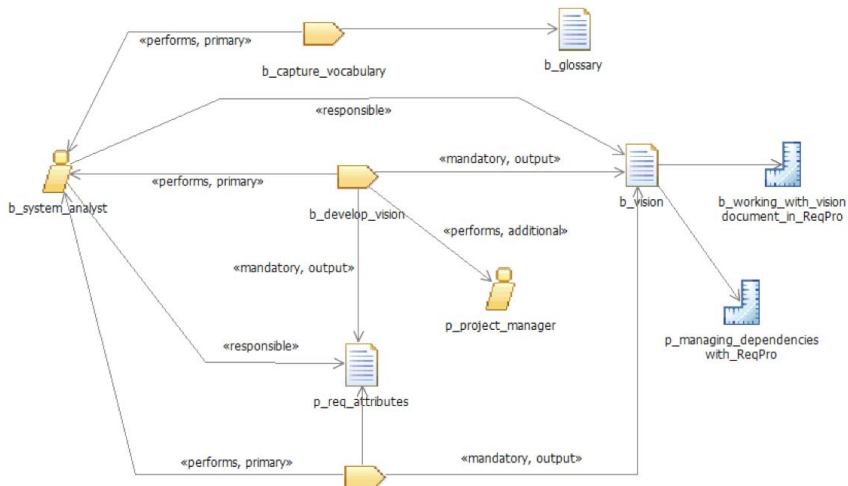


A Process Component contains exactly one Process represented by an Activity, and defines a set of Work Product Ports that define the inputs and outputs for a Process Component.

From OMG SPEM 2.0 Specifications



SPEM: Class Diagram



From OMG SPEM 2.0 Specifications



Outline

- 1 Software Engineering
- 2 Software Process
- 3 Methodologies
- 4 Meta-Models**
 - SPEM
 - **OPF & OPEN**
- 5 Method Engineering
 - Method Fragment Representation
 - Method Assembly

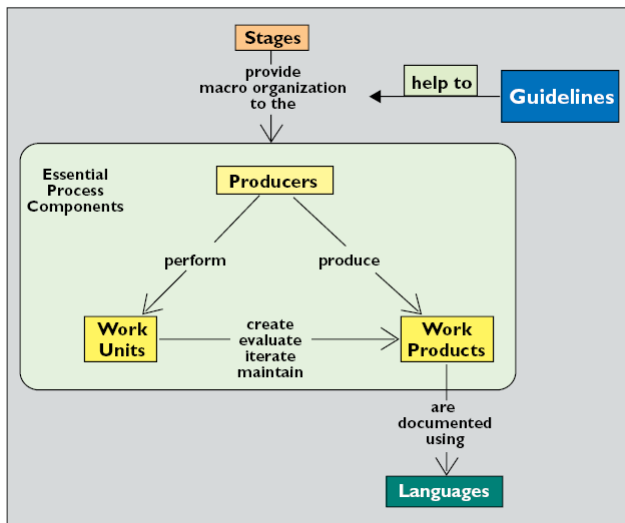


OPEN

- Object-oriented Process, Environment, and Notation (OPEN) [OPEN Working Group, 1997] is a full lifecycle, process-focussed, methodological approach that was designed for the development of software intensive applications
- OPEN is defined as a process framework, known as the OPF (OPEN Process Framework)
- This is a process meta-model from which can be generated an organisationally-specific process (instance)
- Each of these process instances is created by choosing specific Activities, Tasks and Techniques (three of the major metalevel classes) and specific configurations
- The definition of process include not only descriptions of phases, activities, tasks, and techniques but issues associated with human resources, technology, and the life-cycle model to be used



Metalevel Classes [Henderson-Sellers, 2003]



Work Product & Language & Producer

- A *work product* is any significant thing of value (e.g., document, diagram, model, class, application) that is developed during a project
- A *language* is the medium used to document a work product. Use case and object models are written using a modelling language such as the Unified Modeling Language (UML) or the OPEN Modelling Language (OML)
- A *producer* is anything that produces (i.e., creates, evaluates, iterates, or maintains), either directly or indirectly, versions of one or more work products. The OPF distinguishes between those direct producers (persons as well as roles played by the people and tools that they use) and indirect producers (teams of people, organisations and endeavours)



Work Unit

- A *work unit* is a functionally cohesive operation that is performed by a producer during an endeavour and that is reified as an object to provide flexibility during instantiation and tailoring of a process
- The OPF provides the following predefined classes of work units:
 - Task** — functionally cohesive operation that is performed by a direct producer. A task results in the creation, modification, or evaluation of a version of one or more work products
 - Technique** — describes in full detail how a task are to be done
 - Activity** — cohesive collection of workflows that produce a related set of work products. Activities in OPEN are coarse granular descriptions of what needs to be done



Stage

- A *stage* is a formally identified and managed duration or a point in time, and it provides a macro organisation to the work units
- The OPF contains the following predefined classes of stage:
 - Cycle** — there are several types of cycle e.g. lifecycle
 - Phase** — consisting of a sequence of one or more related builds, releases and deployments
 - Workflow** — a sequence of contiguous task performances whereby producers collaborate to produce a work product
 - Build** — a stage describing a chunk of time during which tasks are undertaken
 - Release** — a stage which occurs less frequently than a build. In it, the contents of a build are released by the development organization to another organisation
 - Deployment** — occurs when the user not only receives the product but also, probably experimentally, puts it into service for on-site evaluation
 - Milestone** — is a kind of Stage with no duration. It marks an event occurring



Outline

- 1 Software Engineering
- 2 Software Process
- 3 Methodologies
- 4 Meta-Models
 - SPEM
 - OPF & OPEN
- 5 Method Engineering**
 - Method Fragment Representation
 - Method Assembly



Methodologies

- As for software development, individual methodologies are often created with specific purposes in mind [Henderson-Sellers, 2005]
 - particular domains
 - particular segments of the lifecycle
- Users often make the assumption that a methodology is not in fact constrained but, rather, is universally applicable
- This can easily lead to *methodology failure*, and to the total rejection of methodological thinking by software development organisation
- The creation of a single universally applicable methodology is an unattainable goal
- We should ask ourselves how could we create a methodological environment in which the various demands of different software developers might be satisfied altogether



Method Engineering

Method Engineering [Brinkkemper, 1996]

Method engineering is the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems

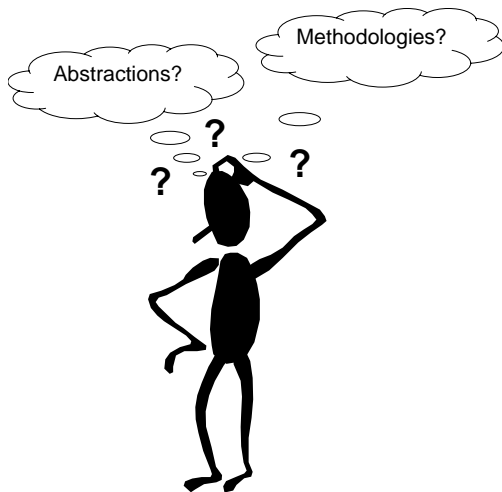


Different Definitions [Brinkkemper, 1996]

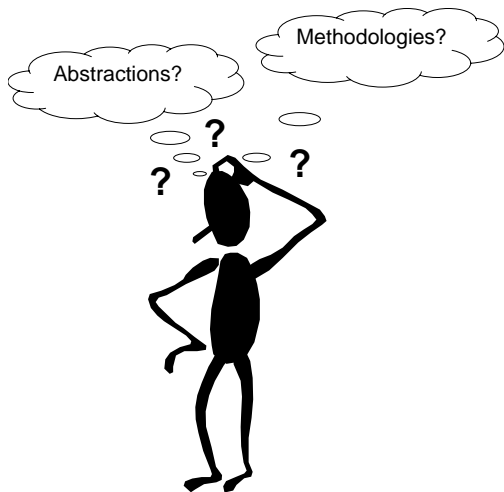
- Method as an approach to perform a systems development project, based on a specific way of thinking, consisting of directions and rules, structured in a systematic way in development activities with corresponding development products
- Methodology as the systematic description, explanation and evaluation of all aspects of methodical information systems development



Method & Methodology



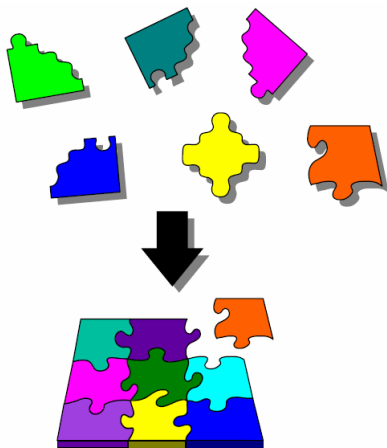
Method & Methodology



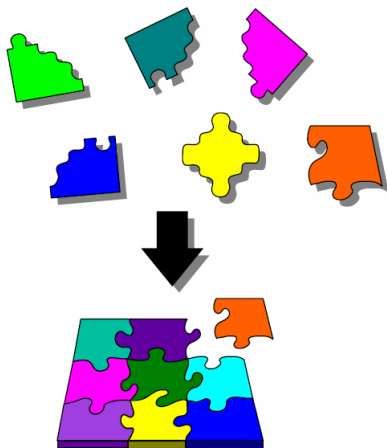
- All the concepts and ideas used in the Method Engineering are also applicable in our definitions of methodology and method
- Method Engineering tries to model methodological processes and products by isolating conceptual *method fragments*
- This fragments act as methodological “building blocks”



Method Engineering: Motivations



Method Engineering: Motivations



- Adaptability – to specific projects, companies, needs & new development settings
- Reuse – of best practices, theories & tools



Method Engineering: Concerns

- Similarly as software engineering is concerned with all aspects of software production, so is method engineering dealing with all engineering activities related to methods, techniques and tools
- The term method engineering is not new but it was already introduced in mechanical engineering to describe the construction of working methods in factories
- Even if the work of Brinkkemper is dated, most of the open research issues presented was not well addressed yet
 - Meta-modelling techniques
 - Tool interoperability
 - Situational method(ology)
 - Comparative review of method(ologie)s and tools



Meta-Modelling Techniques

- The design and evaluation of methods and tools require special purpose specification techniques, called meta-modelling techniques, for describing their procedural and representational capabilities.
- Issues are:
 - what are the proper constructs for meta-modelling?
 - what perspectives of meta-models should be distinguished?
 - is there a most optimal technique for meta-modelling, or is the adequacy of the technique related to the purpose of the investigation?



Tool Interoperability

- A lots of tools that only cover part of the development life-cycle exist
- So the system development practice is confronted with the proper integration of the tools at hand, called interoperability of tools.
- Open problems are related to the overall architecture of the integrated tools
- Should this be based on the storage structure (i.e. the repository) in a data-integration architecture, or on a communication structure between the functional components in a control-integration architecture?



Situational Methods & Comparative Review

- As all projects are different, they cannot be properly supported by a standard method(ology) in a textbook or manual
- How can proper methodical guidance and corresponding tool support be provided to system developers?
- Construction principles for methods and techniques need further investigation
- How can the quality of a method or of a tool be expressed in order to compare them in a sound, scientifically verifiable way?
- Quality of methods comprises aspects as completeness, expressiveness, understandability, effectiveness of resources, and efficiency

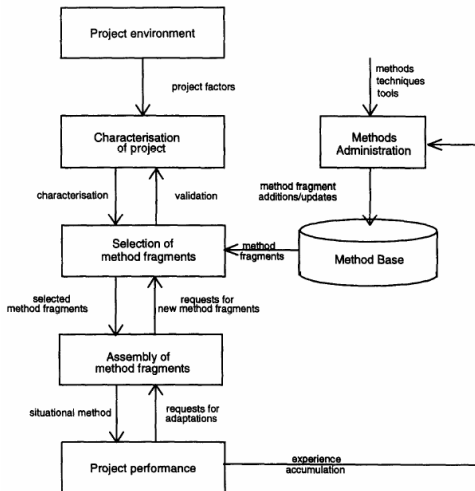


Situational Methodologies

- A situational method is an information systems development method tuned to the situation of the project at hand
- Engineering a situational method requires standardised building blocks and guidelines, so-called meta-methods, to assemble these building blocks
- Critical to the support of engineering situational methods is the provision of *standardised method building blocks* that are stored and retrievable from a so-called method base
- Furthermore, a **configuration process** should be set up that guides the assembly of these building blocks into a situational method
- The building blocks, called **method fragments**, are defined *as coherent pieces of information system development methods*



Configuration Process [Brinkkemper, 1996]



Situational Method Engineering I

- Every project is different, so it is essential in the method configuration process to characterize the project according to a list of contingency factors
- This project characterization is input to the selection process, where method fragments from the method base are retrieved
- Experienced method engineers may also work the other way round, i.e. start with the selection of method fragments and validate this choice against the project characterization
- The unrelated method fragments are then assembled into a situational method
- As the consistency and completeness of the method may require additional method fragments, the selection and validation processes could be repeated



Situational Method Engineering II

- Finally, the situational method is forwarded to the systems developers in the project
- As the project may not be definitely clear at the start, a further elaboration of the situational method can be performed during the course of the project
- Similarly drastic changes in the project require to change the situational method by the removal of inappropriate fragments followed by the insertion of suitable ones



Method Fragments

- [Brinkkemper et al., 1999] classify method fragments according to three different dimensions
 - Perspective — product and process
 - Abstraction level — conceptual and technical
 - Layer of granularity — five different level



Perspective

- Perspective distinguishes product fragments and process fragments

Product fragments — model the structures of the products (deliverables, diagrams, tables, models) of a systems development method

Process fragments — are models of the development process. Process fragments can be either high-level project strategies, called method outlines, or more detailed procedures to support the application of specification techniques



Abstraction Level

- *Abstraction level* distinguishes conceptual level and technical level
 - Method fragments on the conceptual level are descriptions of information systems development methods or part thereof
 - Technical method fragments are implementable specifications of the operational parts of a method, i.e. the tools
- Some conceptual fragments are to be supported by tools, and must therefore be accompanied by corresponding technical fragments
- One conceptual method fragment can be related to several external and technical method fragments

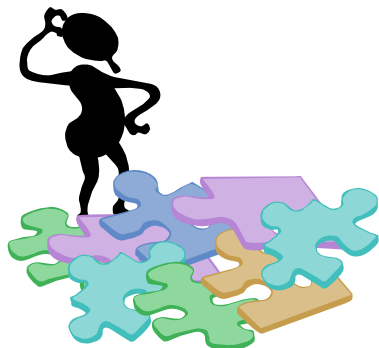


Layer of Granularity

- A method fragment can reside on one of five possible granularity layers
 - Method** — addressing the complete method for developing the information system
 - Stage** — addressing a segment of the life-cycle of the information system
 - Model** — addressing a perspective of the information system
 - Diagram** — addressing the representation of a view of a Model layer method fragment
 - Concept** — addressing the concepts and associations of the method fragments on the Diagram layer, as well as the manipulations defined on them



And Now?



- Two important questions
 - How to represent method fragments?
 - How to assemble method fragments?
- In order to assemble method fragments into a meaningful method, we need a procedure and representation to model method fragments and impose some constraints or rules on method assembly processes



Outline

- 1 Software Engineering
- 2 Software Process
- 3 Methodologies
- 4 Meta-Models
 - SPEM
 - OPF & OPEN
- 5 Method Engineering**
 - Method Fragment Representation**
 - Method Assembly

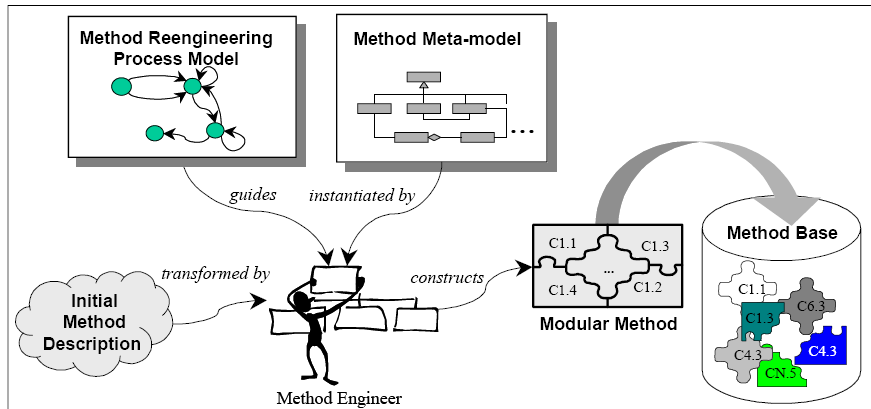


Method Fragment Representation

- In the last decade a lots of work is done in the context of Method Engineering
- However this technique is not entered in the mainstream of the Software Engineering
- There are no consensus in academia and no industry efforts are done
- Each research group has created its method fragment representation
- Here we briefly present the work of Ralyté and Rolland, that has inspired the work of the FIPA group in the context of AOSE
- The OPEN by Brian Henderson-Sellers has already presented in one of the previous Section



Method Reengineering [Ralyté and Rolland, 2001a]

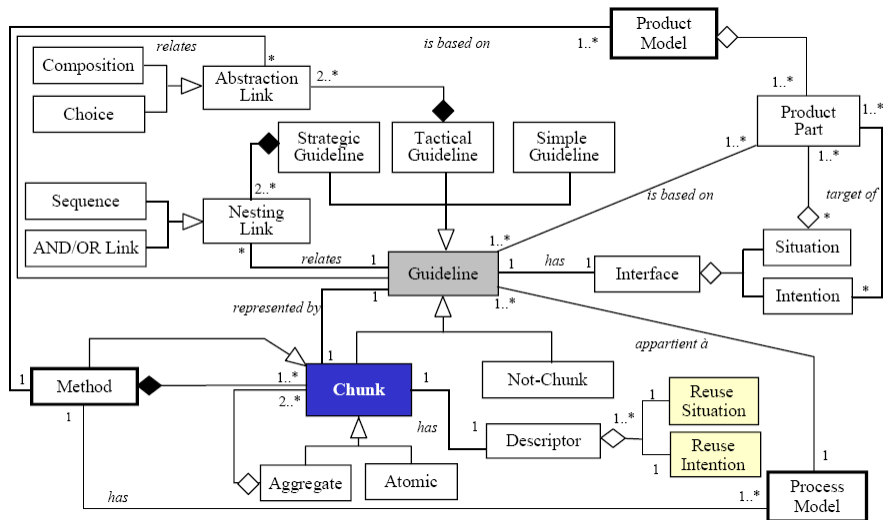


Method Reengineering

- In this approach Ralyté and Rolland adopt the notion of *method chunk* [Ralyté and Rolland, 2001a]
- A method chunk ensures a tight coupling of some process part and its related product part. It is a coherent module and any method is viewed as a set of loosely coupled method chunks expressed at different levels of granularity
- The authors present the method meta-model. . .



The Method Meta-model [Ralyté and Rolland, 2001a]



Method Meta-model

- According to this meta-model a method is also viewed as a method chunk of the highest level of granularity
- The definition of the method chunk is *process-driven* in the sense that a chunk is based on the decomposition of the method process model into reusable guidelines
- Thus, the core of a method chunk is its guideline to which are attached the associated product parts needed to perform the process encapsulated in this guideline
- A guideline embodies method knowledge to guide the application engineer in achieving an intention in a given situation
- Therefore, the guideline has an interface, which describes the conditions of its applicability (the situation) and a body providing guidance to achieve the intention, i.e. to proceed in the construction of the target product



Guidelines

- The body of the guideline details how to apply the chunk to achieve the intention
- The interface of the guideline is also the interface of the corresponding method chunk
- Guidelines in different methods have different contents, formality, granularity, etc.
- In order to capture this variety, the authors identify three types of guidelines: simple, tactical and strategic



Guidelines Types

- A *simple guideline* may have an informal content advising on how to proceed to handle the situation in a narrative form. It can be more structured comprising an executable plan of actions leading to some transformation of the product under construction
- A *tactical guideline* is a complex guideline, which uses a tree structure to relate its sub-guidelines one with the others
- A *strategic guideline* is a complex guideline called a map which uses a graph structure to relate its sub-guidelines. Each sub-guideline belongs to one of the three types of guidelines. A strategic guideline provides a strategic view of the development process telling which intention can be achieved following which strategy



Outline

- 1 Software Engineering
- 2 Software Process
- 3 Methodologies
- 4 Meta-Models
 - SPEM
 - OPF & OPEN
- 5 Method Engineering**
 - Method Fragment Representation
 - Method Assembly**



Method Assembly

- In the last decade a lots of work is done in the context of Method Assembly
- This leads to a proliferation of different techniques for Method Assembly, and each of them adopts a peculiar representation and phases
- Here we briefly show some rules from Brinkkemper, the Method Assembly techniques by Ralyté and Rolland and the OPEN by Brian Henderson-Sellers



Brinkkemper's Rules I

[Brinkkemper et al., 1999] introduce several general rules for the method assembly

- Rule 1** — At least one concept, association or property should be newly introduced to each method fragment to be assembled, i.e. a method fragment to be assembled should not be a subset of another
- Rule 2** — We should have at least one concept and/or association that connects between two method fragments to be assembled
- Rule 3** — If we add new concepts, they should be connectors to both of the assembled method fragments
- Rule 4** — If we add new associations, the two method fragments to be assembled should participate in them



Brinkkemper's Rules II

- Rule 5 — There are no isolated parts in the resulting method fragments
- Rule 6 — There are no concepts which have the same name and which have the different occurrences in a method description
- Rule 7 — The activity of identifying the added concepts and associations that are newly introduced for method assembly should be performed after their associated concepts are identified
- Rule 8 — Let A and B be the two method fragments to be assembled, and C the new method fragment. In C, we should have at least one product which is the output of A and which is the input of B, or the other way round
- Rule 9 — Each product fragment should be produced by a “corresponding” process fragment



Brinkkemper's Rules III

- Rule 10 — Suppose a product fragment has been assembled. The process fragment that produces this product fragment consists of the process fragments that produce the components of the product fragment
- Rule 11 — A technical method fragment should supports a conceptual method fragment
- Rule 12 — If an association exists between two product fragments, there should exist at least one association between their respective components
- Rule 13 — There are no “meaningless” associations in product fragments, i.e. every association is “meaningful” in the sense that it can semantically consistently connect to specific concepts



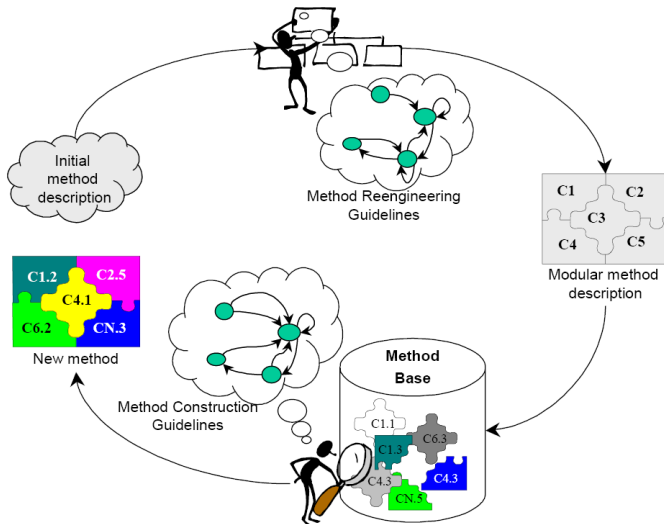
A Different Approach

- Jolita Ralyté and Colette Rolland have proposed a different approach for assembling method chunks
- In particular they have individuated two different assembly strategies:
 - association** — The assembly process by association consists in connecting chunks such that the first one produces a product which is the source of the second chunk
 - integration** — The assembly process by integration consists in identifying the common elements in the chunks product and process models and merging them

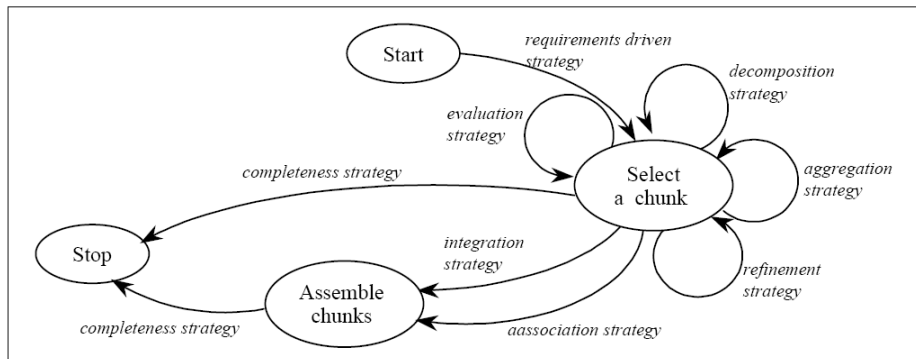


Assembly Based Method Engineering

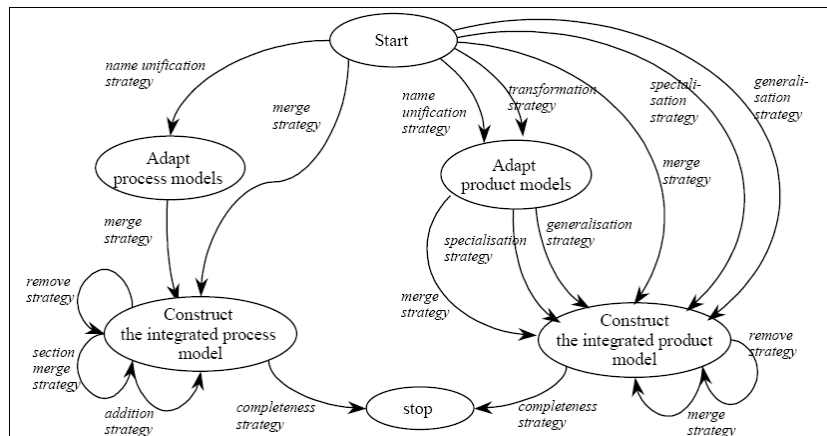
[Ralyté and Rolland, 2001a]



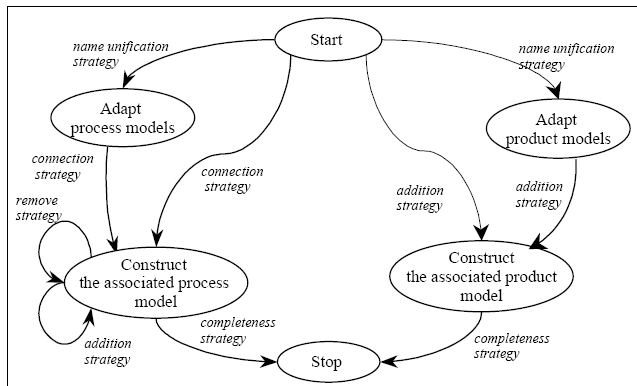
Assembly Map [Ralyté and Rolland, 2001b]



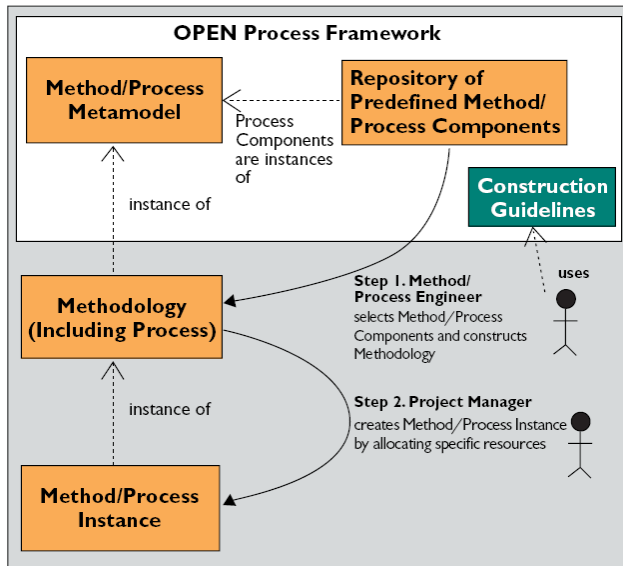
Integration Map [Ralyté and Rolland, 2001b]



Association Map [Ralyté and Rolland, 2001b]



OPEN Process Framework [Henderson-Sellers, 2003]

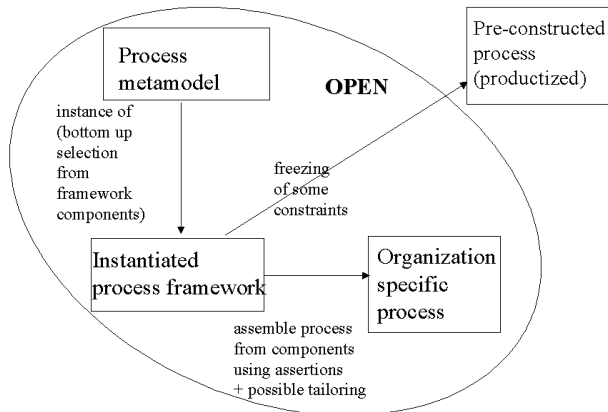


Usage Guidelines

- The core of the Method Assembly in OPF are *usage guidelines* covering:
 - Instantiating the class library to produce actual process components
 - Choosing the best process components
 - Tailoring the fine detail inside the chosen process components
 - Extending the existing class library of predefined process components



OPEN Process Framework [OPEN Working Group, 1997]



It is even possible to take sets of constraints to create "pre-tailored" styles of OPEN process



Process Construction Guidelines

- A process construction guideline is a usage guideline intended to help process engineers instantiate the development process framework and then select the best component instances in order to create the process itself
- Specifically, it will provide guidance concerning how to:
 - Select the work products to develop
 - Select the producers (e.g., roles, teams, and tools) to develop these work products
 - Select the work units to perform
 - How to allocate tasks and associated techniques to the producers
 - How to group the tasks into workflows, activities
 - Select stages of development that will provide an overall organization to these work units



Matrix

- OPEN recommends construction of a number of matrices linking, for example, Activities with Tasks and Tasks with Techniques
- The possibility values in these matrices indicate the likelihood of the effectiveness of each individual pair
- These values should be tailored to a specific organization or a specific project
- Typically a matrix should have five levels of evaluation: mandatory, recommended, optional, discouraged, forbidden
- However a two levels evaluation matrix (use/do not use) gives good results



Matrix Example [Henderson-Sellers, 2003]

Task	Activity					
	1	2	3	4	5	6
Code		x				
Construct the object model		x				x
Develop and implement resource allocation plan						
develop iteration plan	x					
develop timebox plan	x					
set up metrics collection program	x					
specify quality goals	x					
Evaluate quality			x	x		x
Identify CIRTs (Class, Instance, Role, or Type)		x				
Map roles onto classes		x(OOP)				
Test			x	x	x	
Write manuals and other documentation			x	x	x	x
key 1. Project planning 2. Modeling and Implementation: OO analysis, design, programming 3. Verification and validation 4. User review 5. Consolidation 6. Evaluation						



Tailoring Guidelines

- Once the process framework has been instantiated and placed into effect, one typically finds that one needs to perform some fine-tuning by tailoring the instantiated process components as lessons are learned during development
- Tailoring guidelines are usage guidelines intended to help process engineers tailor the instantiated process components



Extension Guidelines

- No class library can ever be totally complete
- Because of the large differences between development projects, new classes of process components will eventually be needed
- Also, software engineering is an evolving discipline, and new process components will need to be added as the field advance
- A process framework should therefore come with extension guidelines, whereby an extension guideline is a usage guideline intended to help the process engineer extend the existing development process framework by adding new classes of process components



Part II

Agent Oriented Software Engineering



Outline

6 AOSE

7 Agent Oriented Methodologies

- MAS Meta-models
- AOSE Methodologies: An Overview
- Methodologies Documentation
- Methodology Challenges

8 Agent Oriented Situational Method Engineering

- Method Fragment Representation
- PRoDe: A Process for the Design of Design Processes
- Method Fragment extraction and Repository creation
- Result Evaluation



Why Agent-Oriented Software Engineering?

- Software engineering is necessary to discipline
 - Software systems and software processes
 - Any approach relies on a set of abstractions and on related methodologies and tools
- Agent-based computing introduces novel abstractions and asks for
 - Making the set of abstractions required clear
 - Adapting methodologies and producing new tools
- Novel, specific agent-oriented software engineering approaches are needed



Agents: Weak Viewpoint

- An *agent* is a software component with internal (either reactive or proactive) threads of execution, and that can be engaged in complex and stateful interactions protocols
- A *multi-agent system* is a software systems made up of multiple independent and encapsulated loci of control (i.e., the agents) interacting with each other in the context of a specific application viewpoint. . .



SE Viewpoint on Agent-Oriented Computing

We commit to weak viewpoint because

- It focuses on the characteristics of agents that have impact on software development
 - Concurrency, interaction, multiple loci of control
 - Intelligence can be seen as a peculiar form of control independence; conversations as a peculiar form of interaction
- It is much more general
 - Does not exclude the strong AI viewpoint
 - Several software systems, even if never conceived as agent-based one, can be indeed characterised in terms of weak multi-agent systems
- Also,
 - it is consistent with the A&A viewpoint



SE Implications of Agent Features I

- **Autonomy**
 - Control encapsulation as a dimension of modularity
 - Conceptually simpler to tackle than a single (or multiple inter-dependent) locus of control
- **Situatedness**
 - Clear separation of concerns between
 - the active computational parts of the system (the agents)
 - the resources of the environment
- **Sociality**
 - Not a single characterising protocol of interaction
 - Interaction as an additional SE dimension
- **Openness**
 - Controlling self-interested agents, malicious behaviours, and badly programmed agents
 - Dynamic re-organisation of software architecture

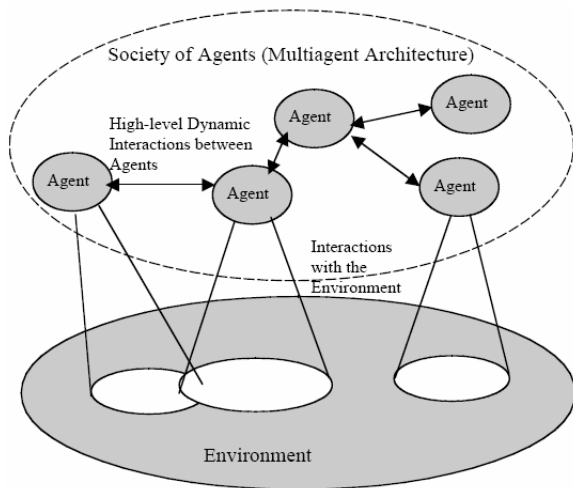


SE Implications of Agent Features II

- Mobility and Locality
 - Additional dimension of autonomous behaviour
 - Improve locality in interactions



MAS Characterisation



Agent-Oriented Abstractions

- The development of a multi-agent system should fruitfully exploit *abstractions* coherent with the above characterisation
 - Agents** — autonomous entities, independent loci of control, situated in an environment, interacting with each other
 - Environment** — the world of resources agents perceive
 - Interaction protocols** — as the acts of interactions among agents and between agents and resources of environment
- In addition, there may be the need of abstracting from
 - the *local context* where an agent lives (e.g., a sub-organisation of agents) so as to handle mobility & openness



Outline

6 AOSE

7 Agent Oriented Methodologies

- MAS Meta-models
- AOSE Methodologies: An Overview
- Methodologies Documentation
- Methodology Challenges

8 Agent Oriented Situational Method Engineering

- Method Fragment Representation
- PProDe: A Process for the Design of Design Processes
- Method Fragment extraction and Repository creation
- Result Evaluation



What is an AO methodology?

- AOSE methodologies mainly try to suggest a clean and disciplined approach to analyse, design and develop multi-agent systems, using specific methods and techniques
- AOSE methodologies, typically start from a *meta-model*, identifying the basic abstractions onto be exploited in development
- On this base, they exploit and organise these abstractions so as to define guidelines on how to proceed in the analysis, design, and development, and on what output to produce at each stage



Outline

6 AOSE

7 Agent Oriented Methodologies

- MAS Meta-models
- AOSE Methodologies: An Overview
- Methodologies Documentation
- Methodology Challenges

8 Agent Oriented Situational Method Engineering

- Method Fragment Representation
- PRoDe: A Process for the Design of Design Processes
- Method Fragment extraction and Repository creation
- Result Evaluation



MAS Meta-model

- MAS meta-models usually include concepts like role, goal, task, plan, communication
- In the agent world the meta-model becomes a **critical element** when trying to create a new methodology because in the agent oriented context, to date, there are not common denominator
 - each methodology has its own concepts and system structure



The process description

- Three are the main elements of a design process
 - Activity
 - Process Role
 - Work Product
- AOSE processes are also affected by
 - MAS Meta-model (MMM) Element
- SPEM does not support the MMM Elements



Extending SPEM Specifications [Seidita et al., 2009]

- MMM is the starting point for the construction of a new design process
 - each part (one or more elements) of this meta-model can be instantiated in one (or more) fragment(s)
- Each fragment **refers** to one (or more) MMM element(s)
 - refers = instantiates/relates/quotes/refines
- The MMM element is the constituent part of a Work Product
- The MMM is not part of the SPEM meta-model
 - it is the element which leads us in modifying and extending SPEM diagram

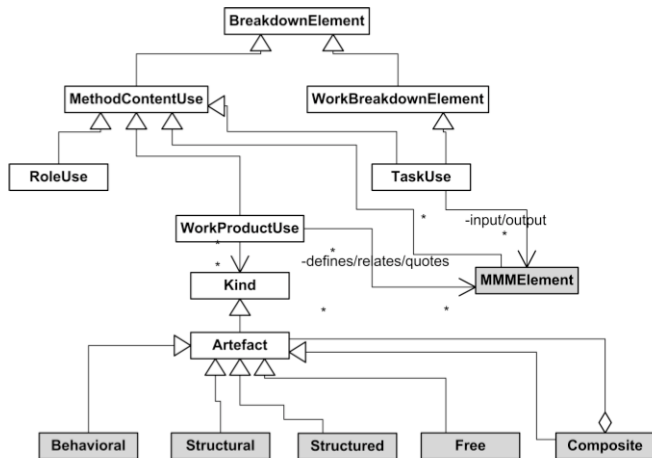


Extending SPEM Specifications [Seidita et al., 2009]

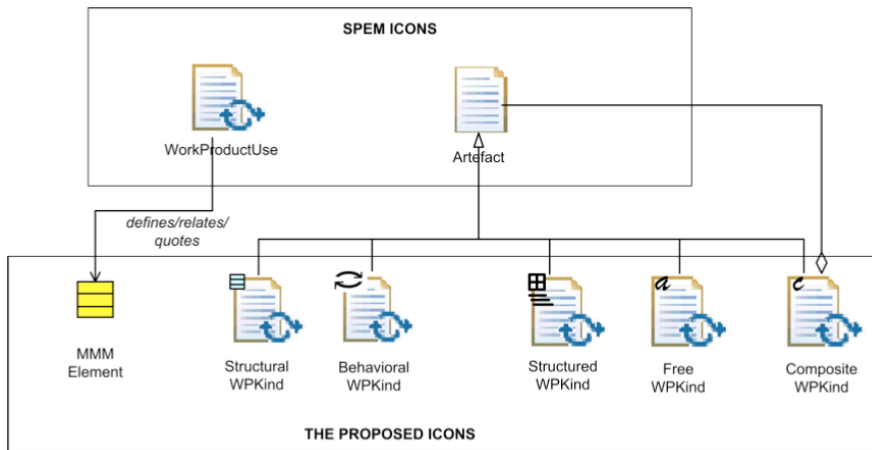
- The need for establishing which is the real action a process role performs on a MMM element when he is carrying out a specific activity
- The set of actions:
 - **define** – it is performed when a MMM element is introduced for the first time and its features are defined in a portion of process (hence in a fragment)
 - **relate** – when a relationship is created (defined) among two or more MMM elements previously defined in another portion of process
 - **quote** – a MMM element or a relationship is quoted in a specific work product
 - **refine** – a MMM element attribute is defined or a value is identified for it
- We also find useful to specify the work product kind by referring to an explicit set of WP kinds



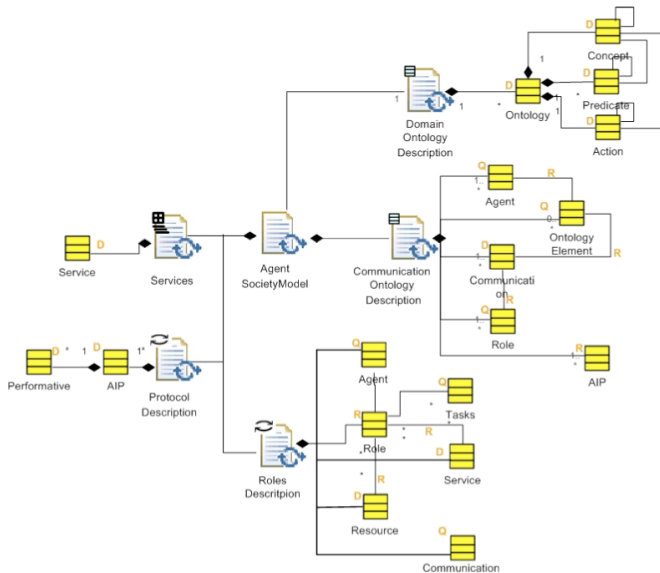
Extending SPEM Specifications [Seidita et al., 2009]



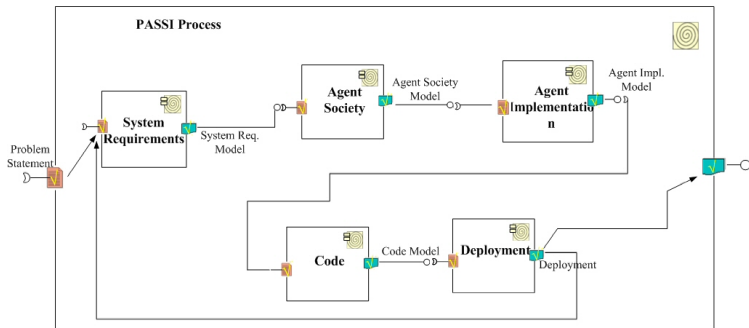
Proposed Icons



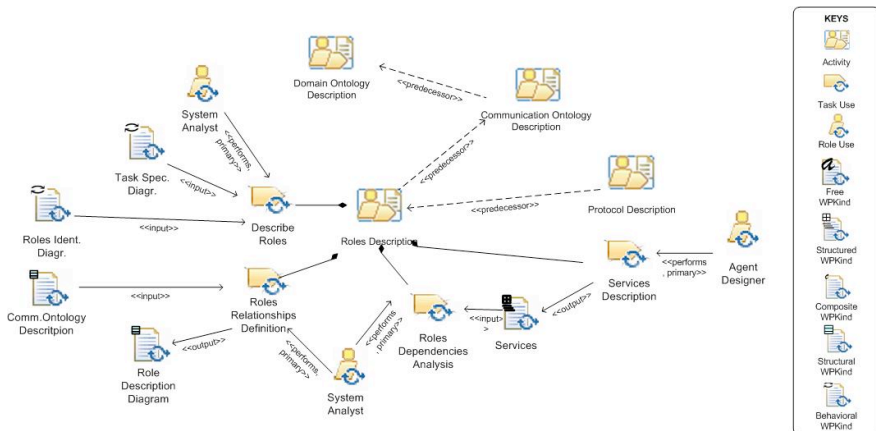
The Dependency Diagram



Example: PASSI Component Diagram



Example: PASSI Process Activity Diagram



Outline

6 AOSE

7 Agent Oriented Methodologies

- MAS Meta-models
- **AOSE Methodologies: An Overview**
- Methodologies Documentation
- Methodology Challenges

8 Agent Oriented Situational Method Engineering

- Method Fragment Representation
- PRoDe: A Process for the Design of Design Processes
- Method Fragment extraction and Repository creation
- Result Evaluation



AOSE Methodologies

- Here we discuss
 - ADELFE [Bernon et al., 2005], [Capera et al., 2004], [Bernon and Capera, 2008]
 - Gaia [Wooldridge et al., 2000] , [Zambonelli et al., 2003], [Cernuzzi et al., 2010]
 - PASSI [Cossentino, 2005], [Cossentino et al., 2008], [Cossentino et al., 2007b]
 - Tropos [Susi et al., 2005], [Bresciani et al., 2004], [Hadar et al., 2010]
 - Prometheus [Padgham and Winikof, 2003], [Padgham and Winikoff, 2005], [DeLoach et al., 2009]
 - SODA [Molesini et al., 2010], [Molesini et al., 2008], [Molesini et al., 2009]
 - INGENIAS [Grasia Group, 2009], [Pavòn et al., 2005], [García-Magariño et al., 2009]



Characteristics of ADELFE

- ADELFE is dedicated to the design of systems that are complex, open and not well-specified (Adaptive Multi-Agent Systems)
- The primary objective of ADELFE method is to cover all the phases of a classical software design
- RUP has been tailored to take into account specificities coming from the design of AMAS
- ADELFE follows the vocabulary of RUP
- Only the requirement, analysis and design *work definitions* require modifications in order to be adapted to AMAS, other appearing in the RUP remaining the same
- ADELFE is supported by several Tools



Adaptive Multi-Agent Systems Theory

- The openness and dynamics are source of *unexpected* events and an open systems plugged into a dynamic environments has to be able to adapt to these changes, to *self-organise*
- Self-organisation is a means to make the system adapt but also to overcome complexity
- If a system is complex and its algorithm unknown, it is impossible to code its *global function*
- This function has then to emerge at the macro level (system level) from the interaction at the micro level (component level)
- It is sufficient to build a system whose components have cooperative attitude to make it realise an expected function
- Cooperation is the local criterion that enables component to find the right place within the organisation

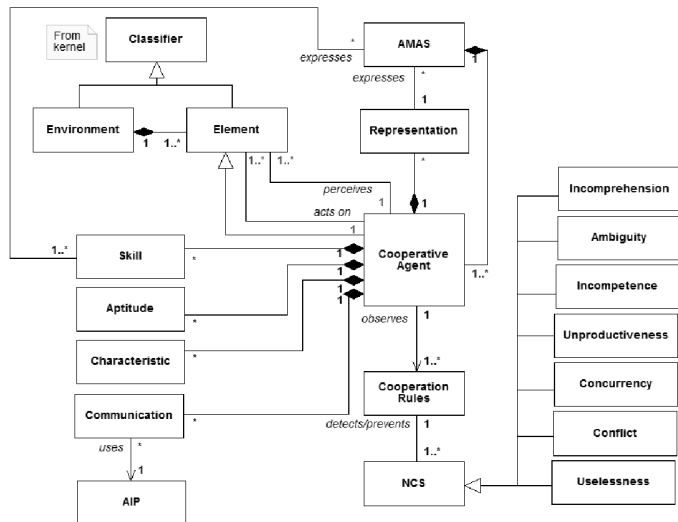


Adaptive Multi-Agent Systems

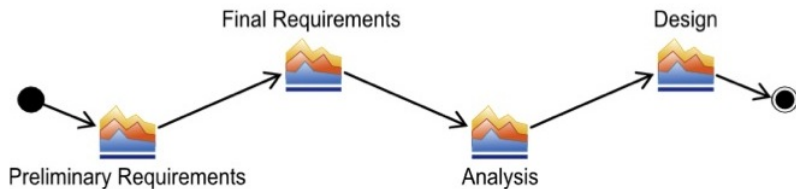
- Adaptive Multi-Agent Systems are composed of agents that permanently try to maintain cooperative interactions with other.
- Any *cooperative agent* in AMAS follow a specific lifecycle that consists in:
 - the agent gets perceptions from its environment
 - it autonomously uses them to decide what to do in order to reach its own goal
 - it acts to realise the action on which it has previously decided



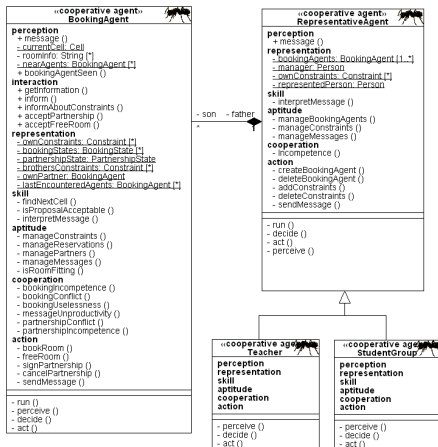
The ADELFE Meta-model



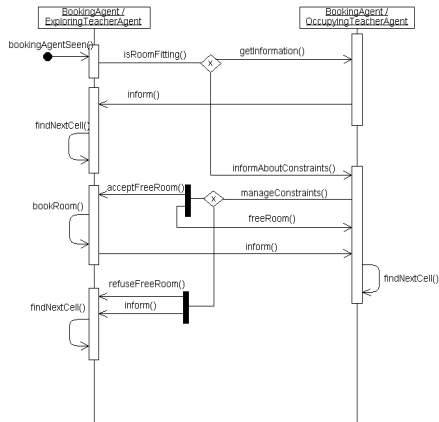
The ADELFE Process



ADELFE: Example



ADELFE: Example



The Gaia Methodology

- It is the most known AOSE methodology
 - firstly proposed by Jennings and Wooldridge in 1999
 - extended and modified by Zambonelli in 2000
 - final Stable Version in 2003 by Zambonelli, Jennings, Wooldridge
 - many other researchers are working towards further extensions. . .



The Gaia Methodology

- Starting from the requirements (what one wants a software system to do)
- Guide developers to a well-defined design for the multi-agent system
- Model and dealing with the characteristics of complex and open multi-agent systems
- Easy to implement

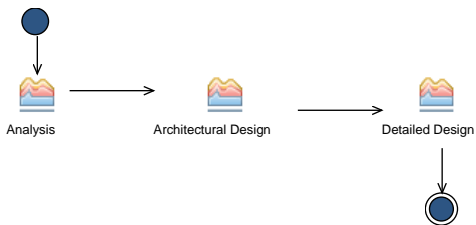


The Gaia Methodology

- Exploits organisational abstractions
 - conceive a multi-agent systems as an organisation of individual, each of which playing specific roles in that organisation
 - and interacting accordingly to its role
- Introduces a clear set of abstractions
 - roles, organisational rules, organisational structures
 - useful to understand and model complex and open multi-agent systems
- Abstract from implementation issues



The Gaia Process



Gaia: Example

Role Name: ReviewCatcher

Description:

This role is in charge of selecting reviewers and distributing papers among them.

Protocol and Activities:

GetPaper, CheckPaperTopic, CheckRefereeExpertise,

CheckRefereeConstraints, AssignPaperReferee,

ReceiveRefereeRefuse, UpdateDBSubmission, UpdateDBReferee

Permissions:

Reads	<i>paper_submitted</i>	in order to check the topic and authors
	<i>referee-data</i>	in order to check the expertise and constraint (i.e. the referee is one of the authors, or belong to the same organization)
Changes	<i>DB Submission</i>	assigning a referee to the paper
	<i>DB Referee</i>	assigning the paper to the referee incrementing the number of assigned papers

Responsibilities:

Liveness: ReviewCatcher = (GetPaper.CheckPaperTopic.CheckRefereeExpertise.CheckRefereeConstraints.AssignPaperReferee.ReceiveRefereeRefuse | UpdateDBSubmission.UpdateDBReferee)ⁿ

Safety: $\forall \text{ paper: number_of_referees} \geq n$
 Referee \neq Author
 Referee_organization \neq Author_organization



Gaia: Example

Action	Environment Abstraction	Description
Reads	Paper Submitted	the Web site receives a paper
	Review Submitted	the Web site receives a review
Changes	DB Submission	insert in the data base the paper or the review received; one per each track
	DB Reviewer	insert in the data base the personal data of the reviewer, the topic of expertise and the maximum number of papers the referee accepted to review

Protocol Name: ReceivePaperAssignmentment

Initiator: ReviewCatcher

Partner: ReviewPartitioner

Input: *paper_submitted*

Description: The **ReviewPartitioner**, having checked the area of the paper, assigns the paper to the corresponding **ReviewCatcher** (the Vice-Chair in charge of that area).

Output: *The paper is assigned to a specific area and the DB Submission is updated*



The PASSI Methodology

- PASSI (Process for Agent Societies Specification and Implementation) is a step-by-step requirement-to-code methodology
- The methodology integrates design models and concepts from both Object Oriented Software Engineering and MAS using UML notation
- PASSI refers to the most diffuse standards: UML, FIPA, JAVA, Rational Rose
- PASSI is conceived to be supported by PTK (PASSI Tool Kit) an agent-oriented CASE tool

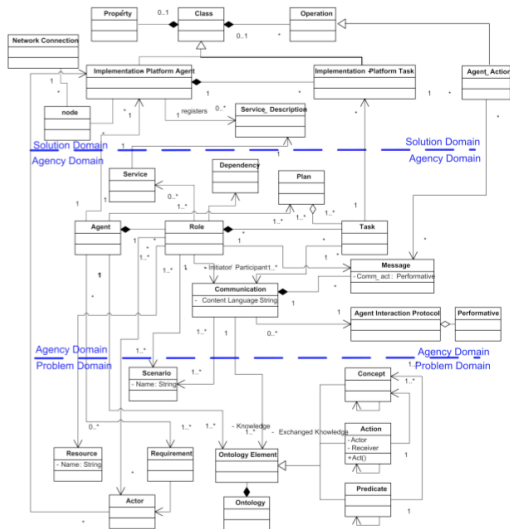


The PASSI Methodology

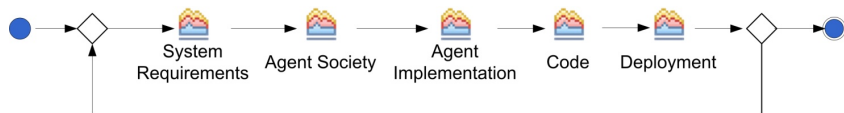
- PASSI process supports:
 - modelling of requirements is based on use-cases
 - ontology that as a central role in the social model
 - multiple perspectives: agents are modelled from the social and internal point of view, both structurally and dynamically
 - reuse of existing portions of design code; this is performed through a pattern-based approach
 - design of real-time systems
 - the design process is incremental and iterative
- Extends UML with the MAS concepts



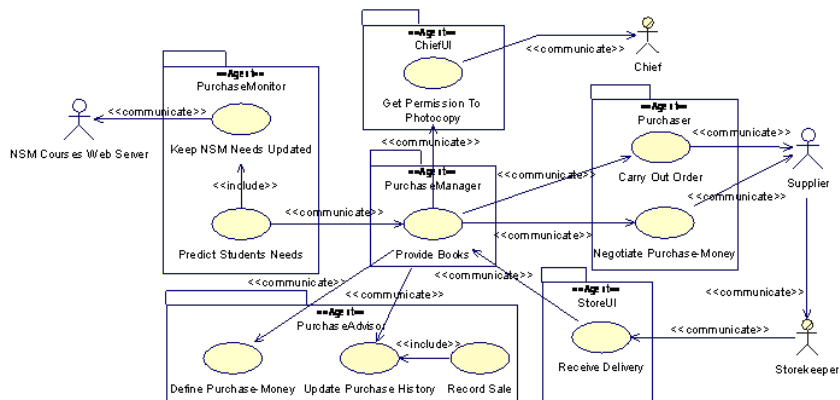
The PASSI Meta-model



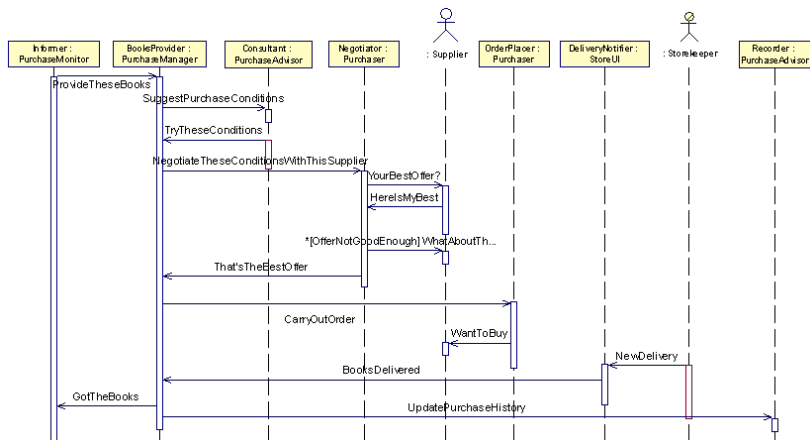
The PASSI Process



PASSI: Example



PASSI: Example



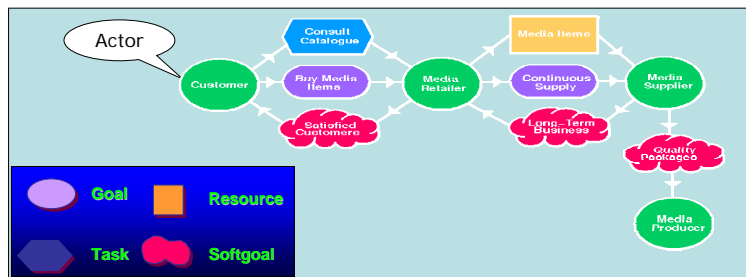
The Tropos Methodology

- Tropos is an agent-oriented software development methodology founded on two key features
 - (i) the notions of agent, goal, plan and various other knowledge level concepts are fundamental primitives used uniformly throughout the software development process
 - (ii) a crucial role is assigned to requirements analysis and specification when the system-to-be analysed with respect to its intended environment
- Then the developers can capture and analyse the goals of stakeholders
- These goals play a crucial role in defining the requirements for the new system: prescriptive requirements capture the what and the how for the system-to-be

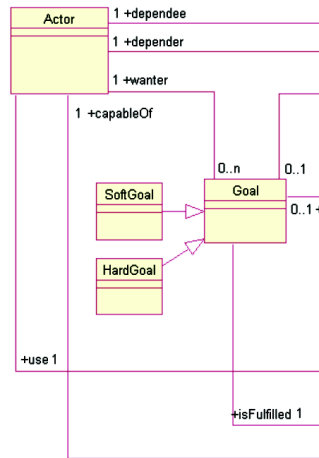
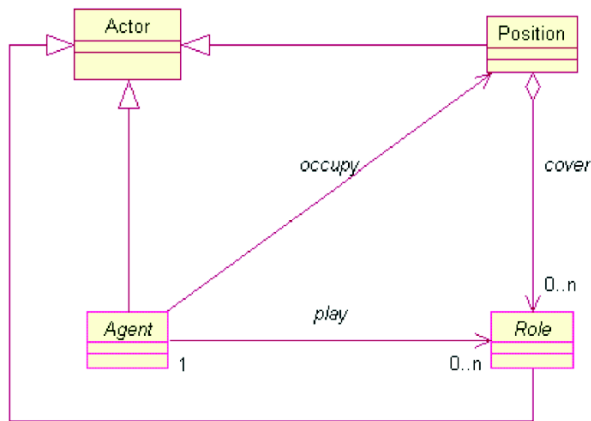


The Tropos Methodology

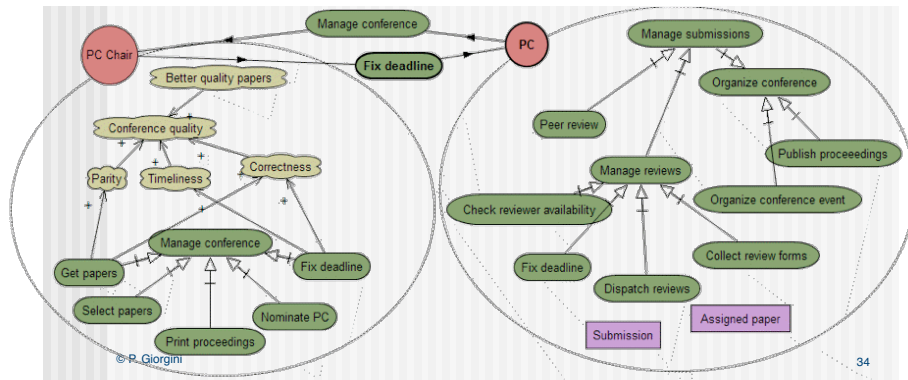
- Tropos adopts Eric Yu's i^* model which offers actors (agents, roles, or positions), goals, and actor dependencies as primitive concepts for modelling an application during early requirements analysis



The Tropos Meta-model



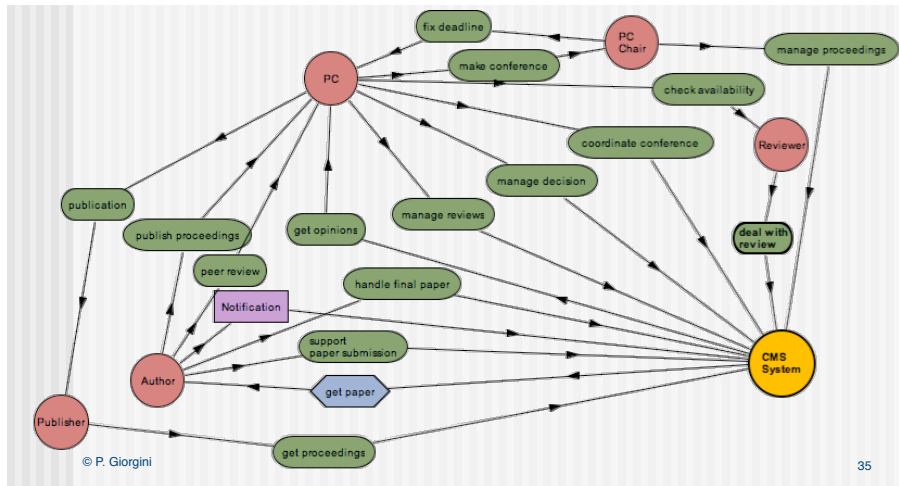
Tropos: Example



34



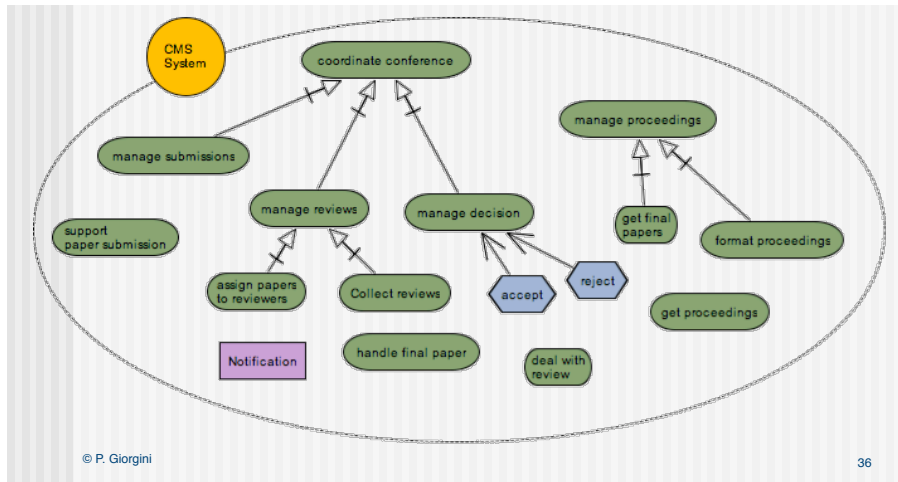
Tropos: Example



35



Tropos: Example



36



The Prometheus Methodology

- Prometheus is a detailed process for specifying, designing, and implementing intelligent agent systems
- The goal in developing Prometheus is to have a process with defined deliverables which can be taught to industry practitioners and undergraduate students who do not have a background in agents and which they can use to develop intelligent agent systems
- Prometheus distinguishes itself from other methodologies by supporting the development of intelligent agents:
 - providing *start-to-end* support,
 - having evolved out of practical industrial and pedagogical experience,
 - having been used in both industry and academia, and, above all, in being detailed and complete

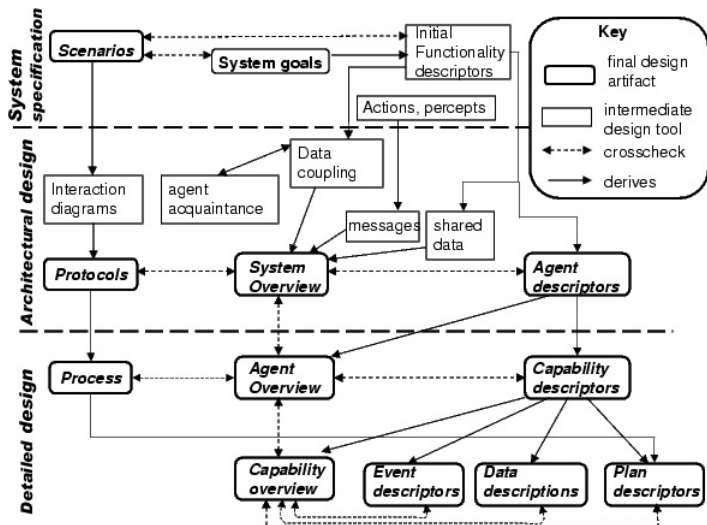


The Prometheus Methodology

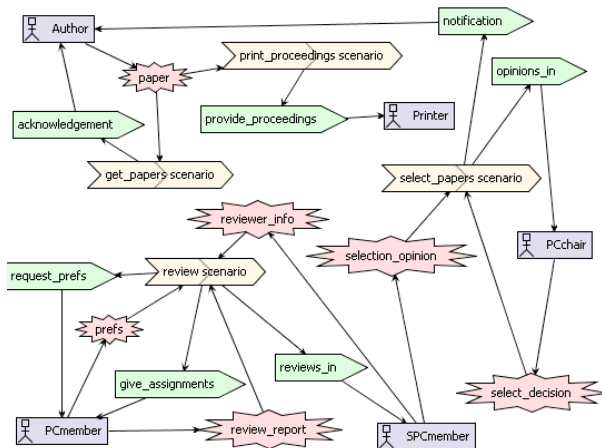
- Prometheus is also amenable to tool support and provides scope for cross checking between designs
- The methodology consists of three phases: system specification, architectural design, and detailed design
- Although the phases are described in a sequential fashion it is acknowledged that like most Software Engineering methodologies, practice involves revisiting earlier phases as one works out the details



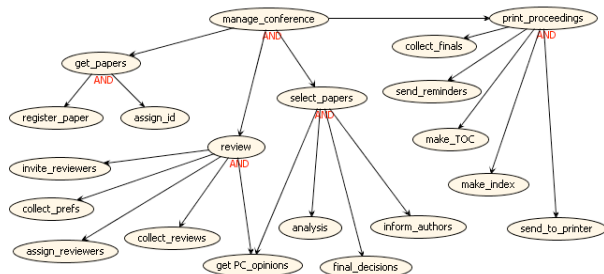
The Prometheus Overview



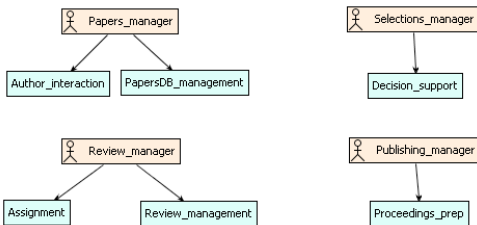
Prometheus: Example



Prometheus: Example



Prometheus: Example



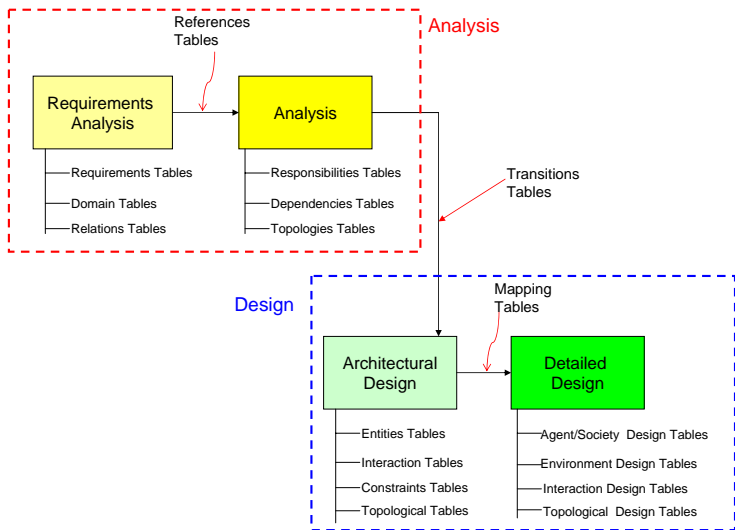
SODA: Societies in Open and Distributed Agent spaces

SODA ...

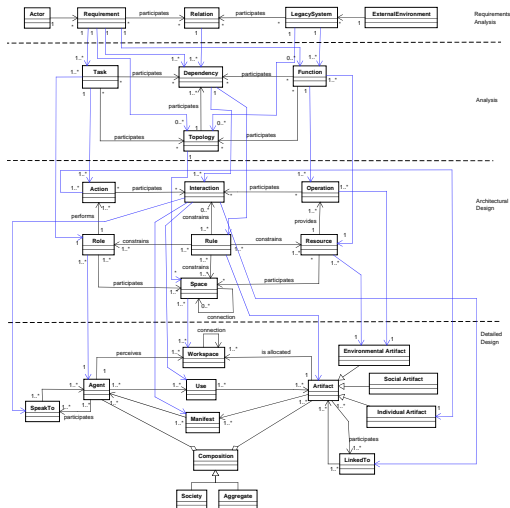
- ... is an agent-oriented methodology for the analysis and design of agent-based systems
- ... focuses on **inter-agent** issues, like the engineering of societies and environment for MAS
- ... adopts **agents** and **artifacts** – after the A&A meta-model [Omicini et al., 2006] – as the main building blocks for MAS development
- ... introduces a simple *layering* principle in order to cope with the complexity of system description
- ... adopts a tabular representation



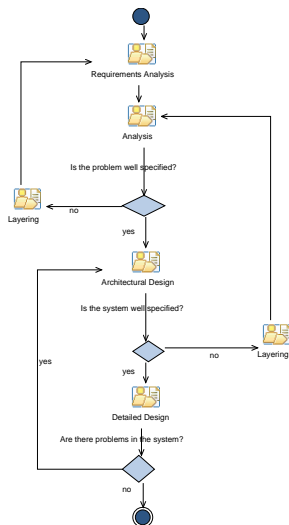
SODA: Overview



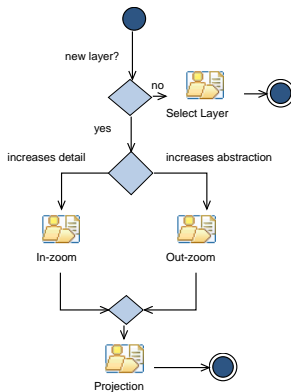
The SODA Meta-model



The SODA Process



The SODA Layering



SODA: Example

Requirement	Description
ManageStartUp	creating call for papers and defining the rules of the organisation
ManageSubmission	managing user registration and paper submissions
ManagePartitioning	partitioning papers based on the conference structure
ManageAssignment	managing the assignment process according to the organisation rules
ManageReview	managing the review process and sending reviews to authors



SODA: Example

Function	Description
management user	managing user information
management review	managing review information
management paper	managing paper information
management assignment	managing assignment information
management partitioning	managing partitioning information
management process	managing start-up information
webSite	web interface of the conference



SODA: Example

Rule	Description
Deadline-Rule	paper can be sent only if current time precedes the deadline
User-Rule	get user is possible if the requested user is the requester or the requester is the PC-chair
Author-Rule	author can access and modify only his public paper information
Match-Rule	papers can be partitioned according key words
AutRev-Rule	the PC-member cannot be one of the paper authors
Review-Rule	the PC-member cannot access private information about his papers



The INGENIAS Methodology

- The INGENIAS methodology covers the analysis and design of MAS and it is intended for general use
- The methodology is supported by the INGENIAS Development Kit (IDK), which contains a graphical editor for MAS specifications
- Besides, the INGENIAS Agent Framework (IAF), integrated in the IDK, has been proposed for enabling a full model-driven development and transforming automatically specifications into code in the Java Agent Development Framework
- The software development process proposed by the methodology is based on RUP [Kruchten, 2003]
- The methodology distributes the tasks of analysis and design in three consecutive phases: Inception, Elaboration and Construction
- Each phase may have several iterations (where iteration means a complete cycle of development)

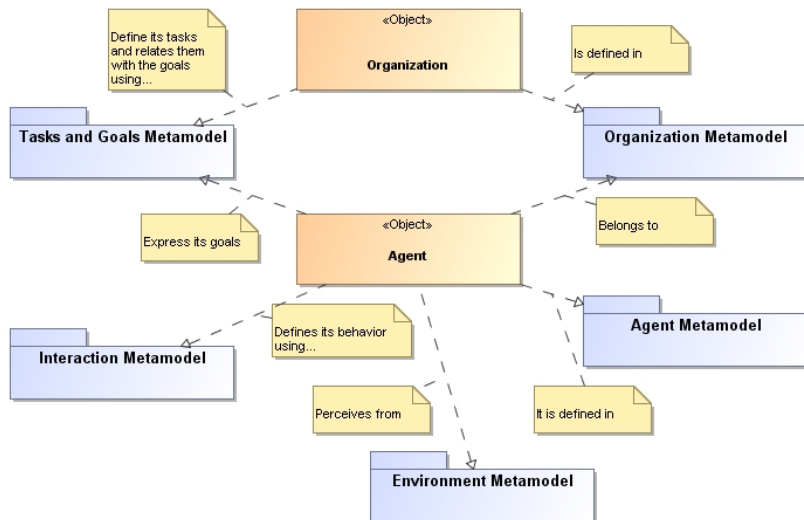


The INGENIAS Methodology

- INGENIAS follows the Model Driven Development(MDD), so it is based on the definition of a set of meta-models that describe the elements that form a MAS from several viewpoints
- The specification of a MAS is structured in five viewpoints:
 - ① the definition, control and management of each agent mental state
 - ② the agent interactions
 - ③ the MAS organisation
 - ④ the environment
 - ⑤ the tasks and goals assigned to each agent



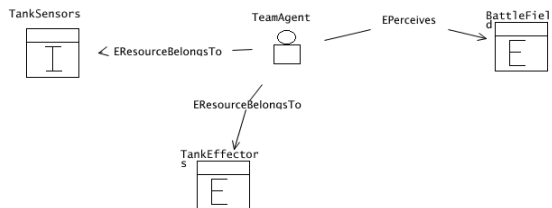
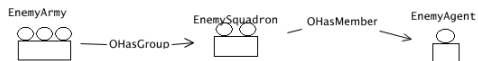
The INGENIAS Meta-model



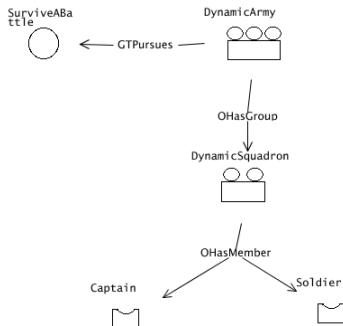
The INGENIAS Process



INGENIAS: Example



INGENIAS: Example



Outline

6 AOSE

7 Agent Oriented Methodologies

- MAS Meta-models
- AOSE Methodologies: An Overview
- **Methodologies Documentation**
- Methodology Challenges

8 Agent Oriented Situational Method Engineering

- Method Fragment Representation
- PRoDe: A Process for the Design of Design Processes
- Method Fragment extraction and Repository creation
- Result Evaluation



AOSE & Processes

- In the software engineering field, there is a common agreement about the fact that there is not a unique methodology or process, which fits all the application domains
- This means that the methodology or process must be adapted to the particular characteristics of the domain for which the new software is developed
- There are two major ways for adapting methodologies:
 - tailoring: particularization or customization of a pre-existing processes
 - Situational Method Engineering (SME): process is assembled from pre-existent components, called fragments, according to user's needs (see next section)
- The research on SME has become crucial in AOSE since a variety of special-purpose agent-oriented methodologies have been defined in the past years to discipline and support the MASs development



AOSE & Processes

- Each of the AO methodologies proposed until now presents specific **meta-model**, **notation**, and **process**
- These characteristics
 - are fundamental for a correct comprehension of a methodology
 - should be documented in a proper way for supporting the creation of new ad-hoc AOSE methodologies
- SME is strictly related to the documentation of the existing methodologies
 - the successfully construction of a new process is based on the correct integration of different fragments that should be well formalised
- The methodologies' documentation should be done in a standard way in order to facilitate
 - the user's comprehension
 - the adoption of automatic tools able to interpret the fragment documentation



Methodologies Documentation

- The IEEE FIPA Design Process Documentation and Fragmentation (DPDF) working group
[IEEE FIPA Design Process Documentation and Fragmentation Working Group] has recently proposed a **template** for documenting AO methodologies
- This template
 - has been conceived without considering any particular process or methodology → all processes can be documented using it
 - is neutral regarding the MAS meta-model and/or the modelling notation adopted in describing the process
 - has a simple structure resembling a tree, so documentation is made in a natural and progressive way:
 - addressing in first place the general description and meta-model definition which constitute the root elements of the process
 - detailing in a second step the branches which are the phases
 - is easy to use for a software engineer as it relies on few previous assumptions
 - suggests as notation the use of the OMG's standard SPEM [Object Management Group, 2008] with few extensions



Template structure

1. Introduction
 - 1.1. The (process name) Process lifecycle
 - 1.2. The (process name) Meta-model
 - 1.2.1. Definition of MAS meta-model elements
 - 1.3. Guidelines and Techniques
2. Phases of the (process name) Process
 - 2.1. (First) Phase
 - 2.1.1. Process roles
 - 2.1.2. Activity Details
 - 2.1.3. Work Products
 - 2.2. (Second) Phase
 - 2.2.1. Process roles
 - 2.2.2. Activity Details
 - 2.2.3. Work Products
 - ... (further phases) ...
3. Work Product Dependencies



Methodologies Documentation: Benefits

- The template helps
 - in easily catching/understanding/studying the methodology: it seems evident the facility of studying another methodology when the new one uses an approach we already know
 - in reusing parts
 - in identifying similarities and differences in the methodologies



Methodologies Documentation: Examples

Examples

- `http://www.pa.icar.cnr.it/cossentino/fipa-dpdf-wg/docs.htm`
- `http://www.alice.unibo.it/xwiki/bin/view/SODA/Documents`



Outline

6 AOSE

7 Agent Oriented Methodologies

- MAS Meta-models
- AOSE Methodologies: An Overview
- Methodologies Documentation
- Methodology Challenges

8 Agent Oriented Situational Method Engineering

- Method Fragment Representation
- PRoDe: A Process for the Design of Design Processes
- Method Fragment extraction and Repository creation
- Result Evaluation

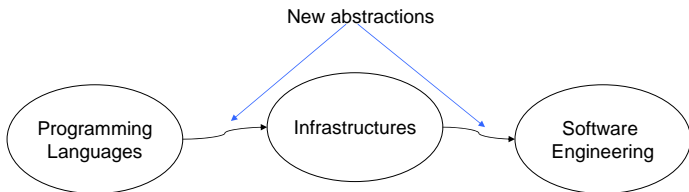


Methodologies & Technologies

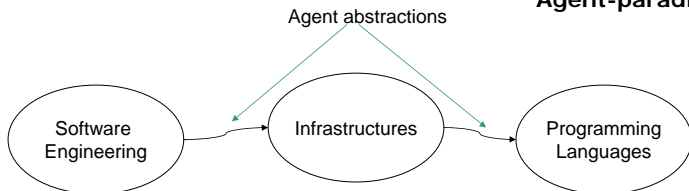
- Today engineers often work with technologies that do not support the abstractions used in the design of the systems
- This is why the research on methodologies becomes the basic point in the scientific activity
- There is a deep **gap** between the AOSE approaches and the available technologies
 - the proposed AOSE methodologies mostly follow a **top-down approach**, where the agent paradigm and the metaphors of the human organisation have been used to analyse, model and design a system
 - multi-agent languages and tools mostly follow a **bottom-up approach**, evolving out of necessity from existing programming languages and development environments



Informatics Technology Evolution



**Traditional
Agent-paradigm**



The Gap

- The gap between methodologies and infrastructures and languages can lead to *dangerous inconsistencies* between the design and the actual implementation of the system
- These are the consequences of the use of concepts and abstractions in the analysis and design stages which are different from those used to deploy and implement the system
- On one side the agent-based abstractions available in the design phase suggest high level of expressivity
- On the other side the development tools, that are still in the stage of academic prototypes, do not support these abstractions



Challenges

- Two important challenges that represent the principal objective of the researchers in the next years [MEnSA Project, 2008]:
 - identification of the effective abstractions to model complex systems as multi-agent systems
 - integration of these abstractions in methodologies that support the whole software life cycle and fill the conceptual gap between agent-oriented methodologies and the infrastructures used to implement agent-based systems
- This leads to the fragmentation of the existing AO methodologies in order to construct new and ad hoc methodologies. . .



Outline

6 AOSE

7 Agent Oriented Methodologies

- MAS Meta-models
- AOSE Methodologies: An Overview
- Methodologies Documentation
- Methodology Challenges

8 Agent Oriented Situational Method Engineering

- Method Fragment Representation
- PRoDe: A Process for the Design of Design Processes
- Method Fragment extraction and Repository creation
- Result Evaluation



Method Engineering I

- The development of complex software systems using the agent-oriented approach requires suitable methodologies which provide explicit support for the key abstractions of the agent paradigm [Cossentino et al., 2007a]
- To date, several methodologies supporting the analysis, design and implementation of MAS have been proposed in the context of AOSE
- Although such methodologies have different advantages when applied to specific problems, it is a fact that a unique methodology cannot be general enough to be useful for everyone without some level of customisation.
- In fact, agent designers, in solving specific problems in a specific application context, often prefer to define their own methodology, specifically tailored to their needs, instead of reusing an existing one

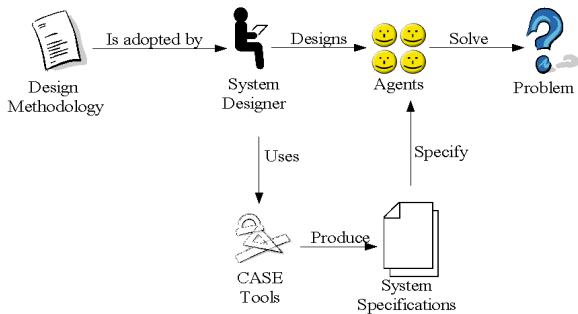


Method Engineering II

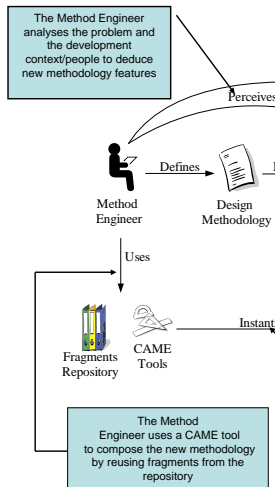
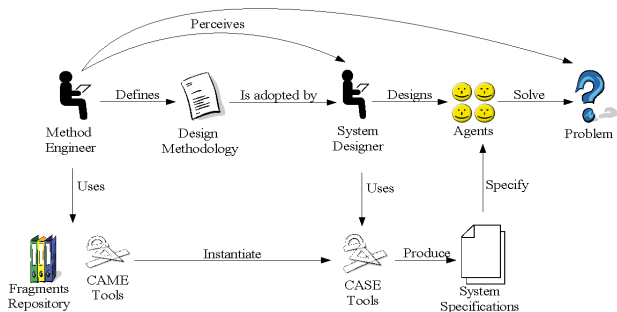
- Thus an approach that combines the designer's need to define his/her own methodology with the advantages and the experiences coming from the existing and documented methodologies is highly required
- A possible solution to this problem is to adopt the *method engineering paradigm*, thus enabling designers of MAS to (re)use parts coming from different methodologies in order to build up a customised approach to their own problems.
- According to this approach, the “development methodology” is constructed by assembling pieces of other methodologies (*method fragments*) from a repository of methods (*method base*).
- The method base is composed of contributions coming from existing methodologies and other novel and specifically conceived fragment



The "Normal" Agent Development Process



Situational Method Engineering



Agent Oriented Situational Method Engineering

- The development methodology is built by the developer by assembling pieces of the process (**method fragments**) from a **method base**
- The method base is composed of contributions coming from existing methodologies and other novel and specifically conceived fragments
- This is the approach used within both the FIPA Technical Committee Methodology (2003-2005) and the IEEE FIPA Design Process Documentation and Fragmentation (DPDF) (2009-X)



Adopting Situational Method Engineering

- What do I need?
 - a collection of method fragments
 - some guidelines about how to assemble fragments
 - a CAME (Computer Aided Method Engineering) tool
 - a CAPE (Computer Aided Process Engineering) tool
 - an evaluation framework (is my new methodology really good?)



CAME Tool

- This tool is based on the method meta-model and it is responsible for method fragment specification, i.e. their product and process parts definition.
- Method fragment specification can be done “from scratch”, by assembly or by modification.
- In the first case product and process models of the fragments are defined by instantiating the method meta-model used by the tool.
- In the second case fragments are assembled in order to satisfy some specific situation.
- In the third case fragments are obtained by modification of other fragments in order to better satisfy the method goal.
- Depending to the method meta-model, the CAME tool should offer graphical modelling facilities and special features.

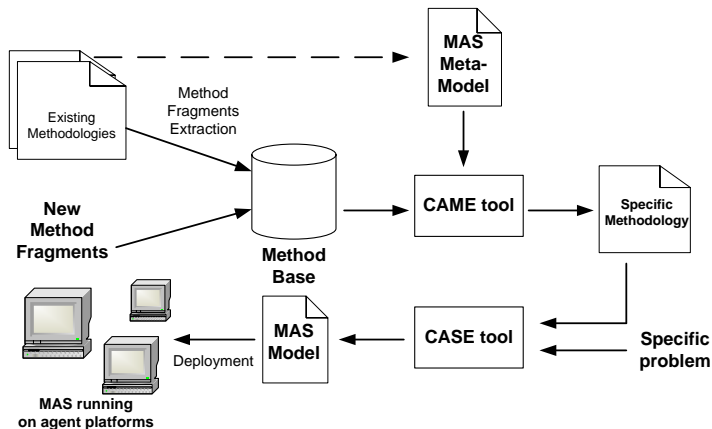


CAPE Tool

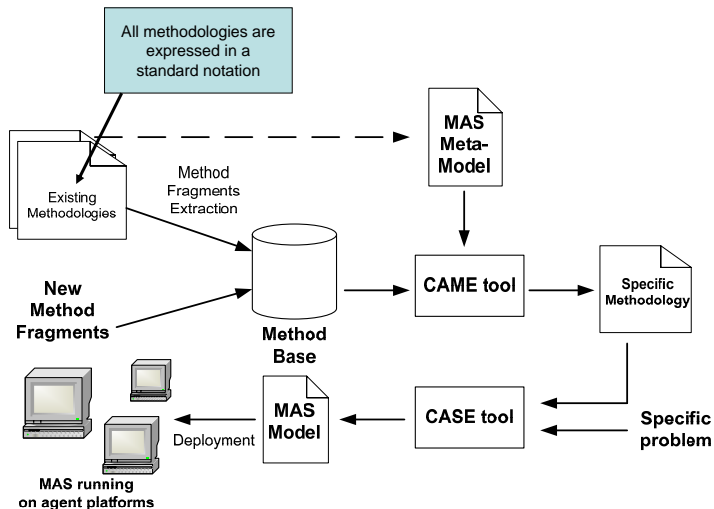
- CAPE tools that could enable the construction of the new design process; these tools should be able to support the definition of the process life-cycle as well as the reuse of fragments from the method base.
- They should enable the adoption of a specific process life-cycle (waterfall, iterative/incremental, spiral, etc.) and the placing of different fragments in it.
- The CAPE tool should “instantiate” a proper CASE tool (see below) that is specifically customised to support the designer in working with the composed methodology.



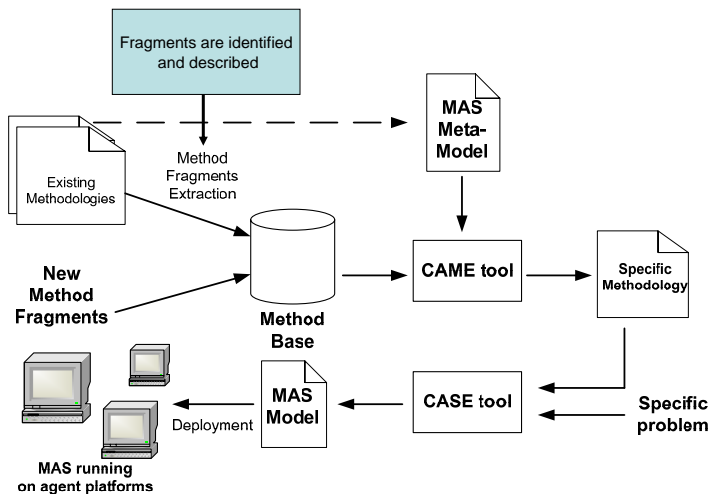
The New Process Production



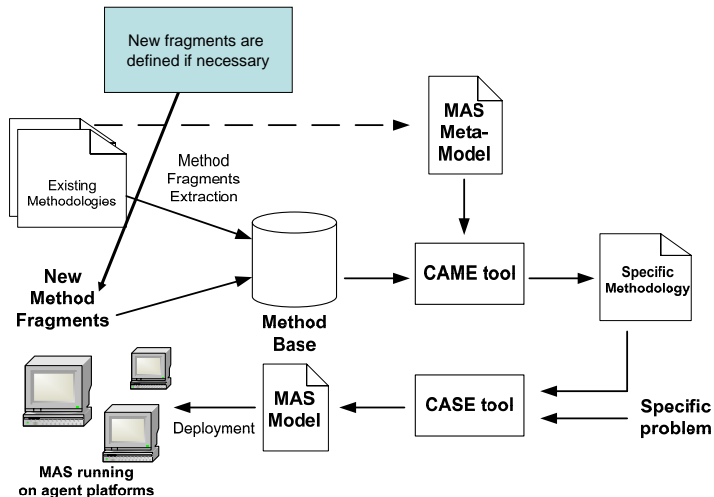
The New Process Production



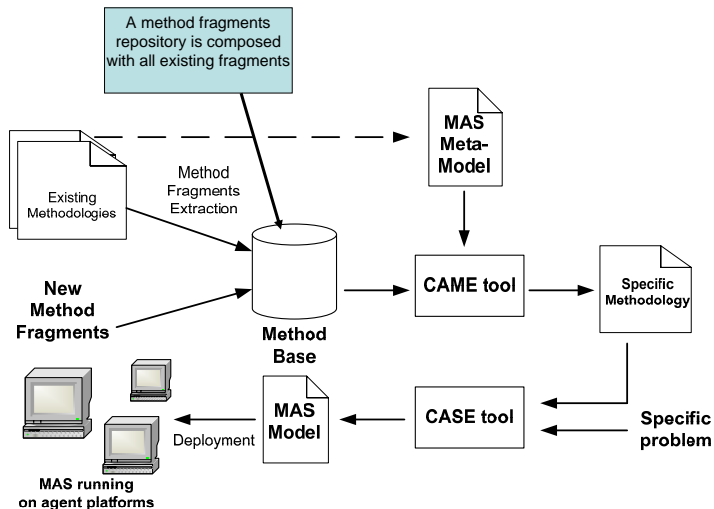
The New Process Production



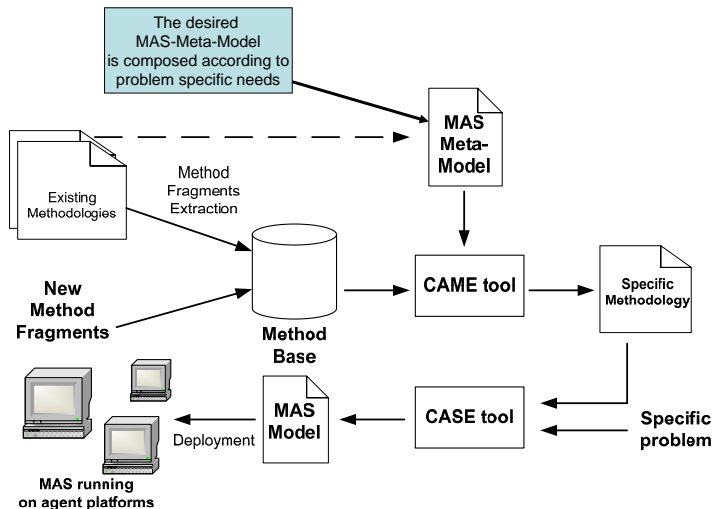
The New Process Production



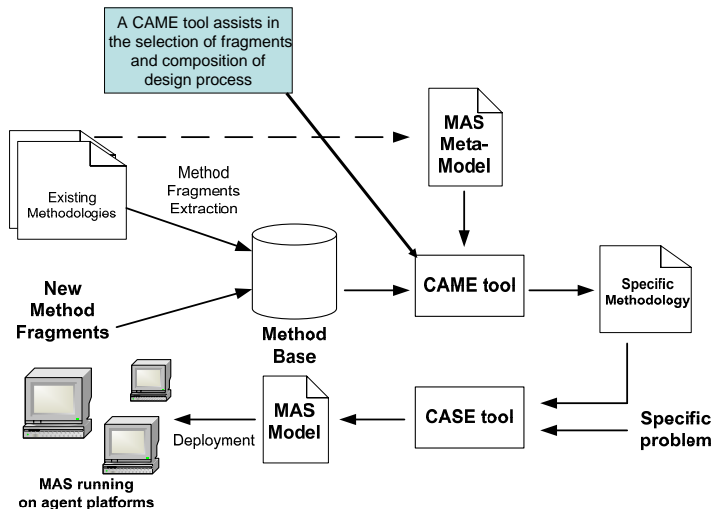
The New Process Production



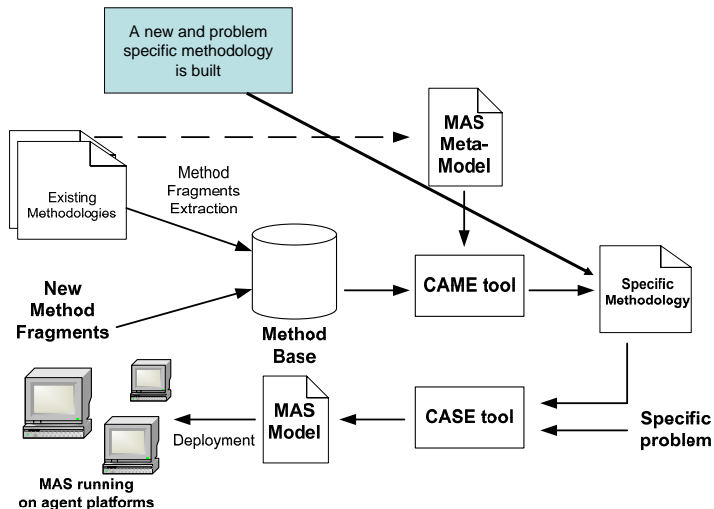
The New Process Production



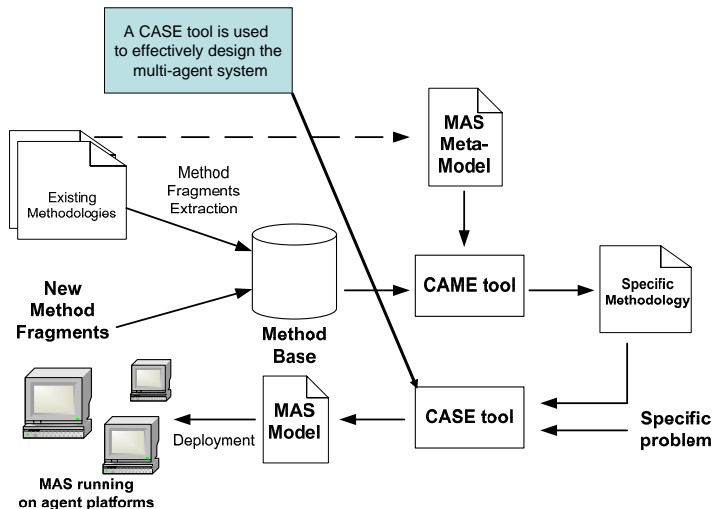
The New Process Production



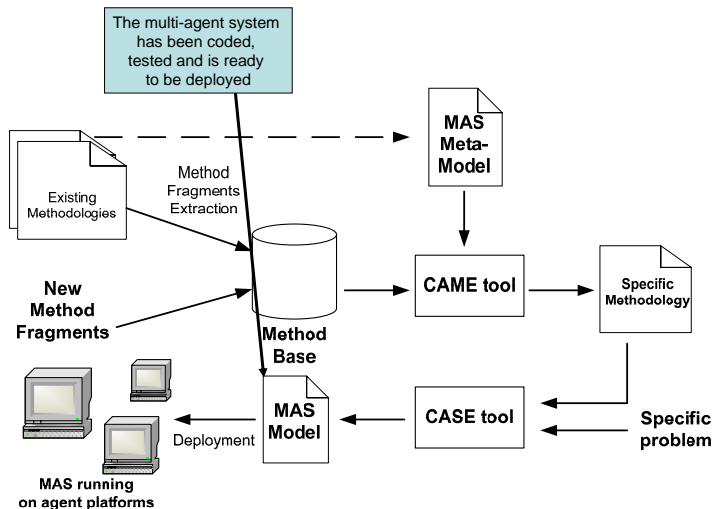
The New Process Production



The New Process Production



The New Process Production



So We Need...

- A specific description of an AOSE fragment
- A way for assembly AOSE fragments



Outline

- 6 AOSE
- 7 Agent Oriented Methodologies
 - MAS Meta-models
 - AOSE Methodologies: An Overview
 - Methodologies Documentation
 - Methodology Challenges
- 8 Agent Oriented Situational Method Engineering**
 - Method Fragment Representation**
 - PProDe: A Process for the Design of Design Processes
 - Method Fragment extraction and Repository creation
 - Result Evaluation



What is a Method Fragment

A **fragment** is a portion of the development process, composed as follows:

- A **portion of process** (what is to be done, in what order), defined with a **SPEM** diagram
- One or more **deliverables** (like (A)UML/UML diagrams, text documents and so on)
- Some **preconditions** (they are a kind of constraint because it is not possible to start the process specified in the fragment without the required input data or without verifying the required guard condition)
- A **list of concepts** (related to the MAS meta-model) to be defined (designed) or refined during the specified process fragment
- **Guideline(s)** that illustrates how to apply the fragment and best practices related to that
- A **glossary of terms** used in the fragment (in order to avoid misunderstandings if the fragment is reused in a context that is different from the original one)
- Other information (composition guidelines, platform to be used, application area and dependency relationships useful to assemble fragments) complete this definition.



Outline

6 AOSE

7 Agent Oriented Methodologies

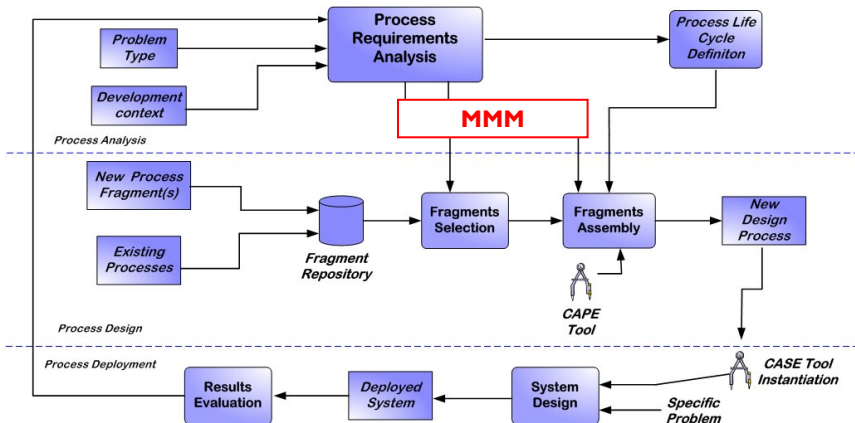
- MAS Meta-models
- AOSE Methodologies: An Overview
- Methodologies Documentation
- Methodology Challenges

8 Agent Oriented Situational Method Engineering

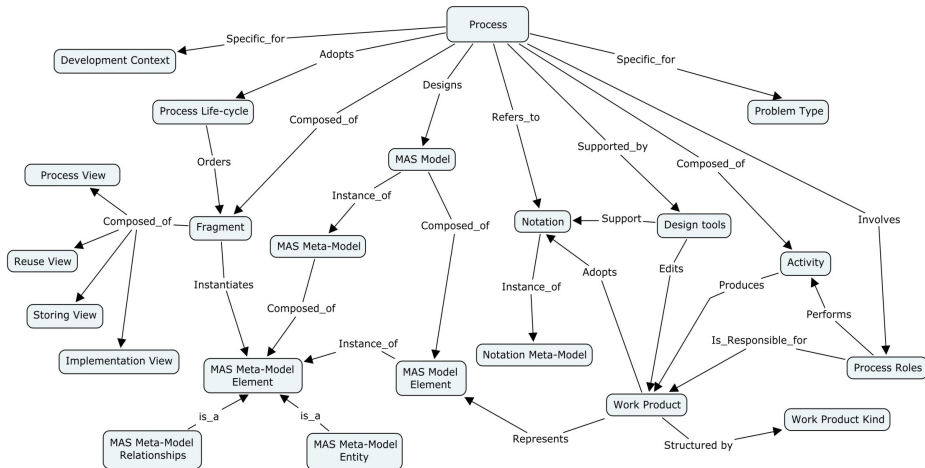
- Method Fragment Representation
- PRoDe: A Process for the Design of Design Processes
- Method Fragment extraction and Repository creation
- Result Evaluation



PRoDe: An Approach for Agent-Oriented Method Engineering [Seidita et al., 2010]



PRoDe: Process Representation

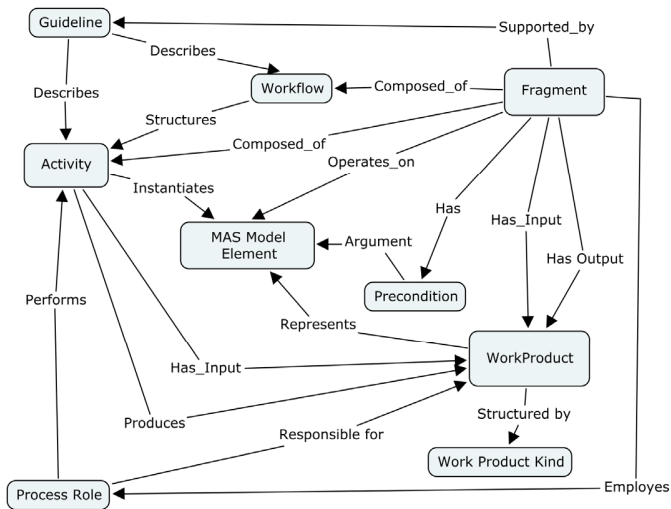


Applying the Proposed Method Fragment Definition

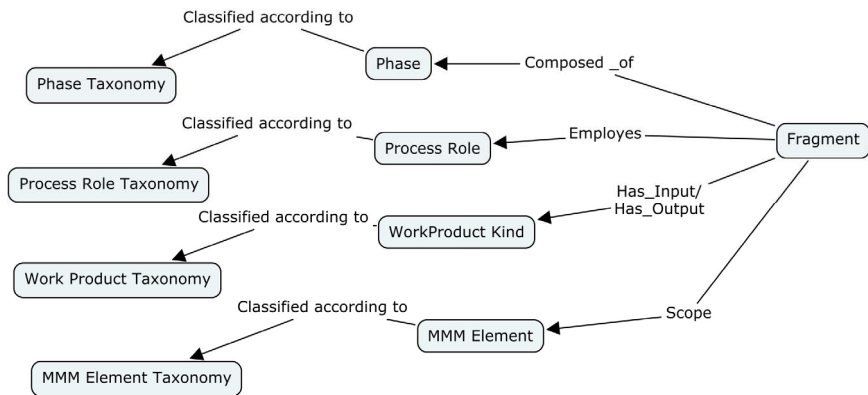
- A method Fragment can be explored from four points of view [Cossentino et al., 2007a]:
 - Process
 - the process related aspect of the fragment: workflow, activity and work product
 - Storing
 - it concerns with the storage of the fragment in the method base and its retrieval
 - Reuse
 - it concerns with the reuse feature of the fragment and lists the elements helpful in reusing the fragment during the composition of a new design process
 - Implementation
 - the implementation of the main elements of the process view
- Method fragment construction is Work Product oriented, a method fragment must deliver a product.



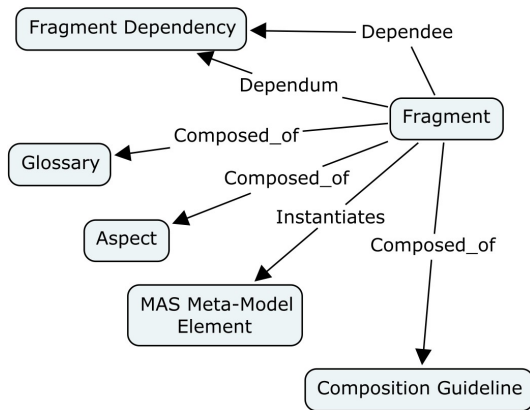
The Process View



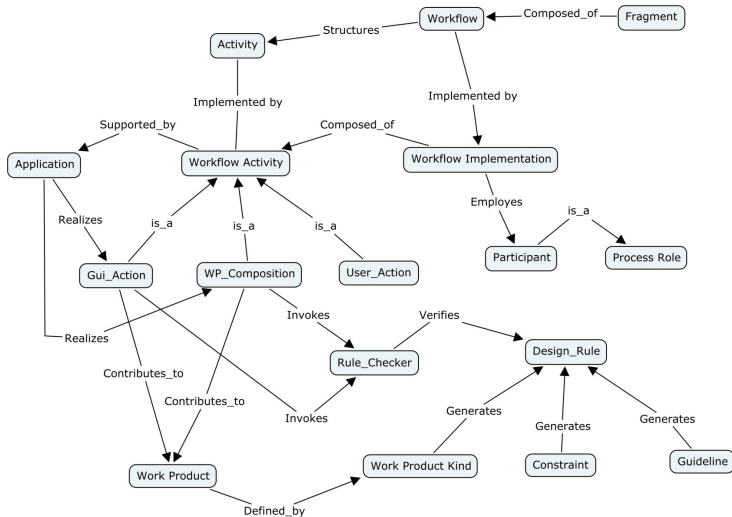
The Storing View



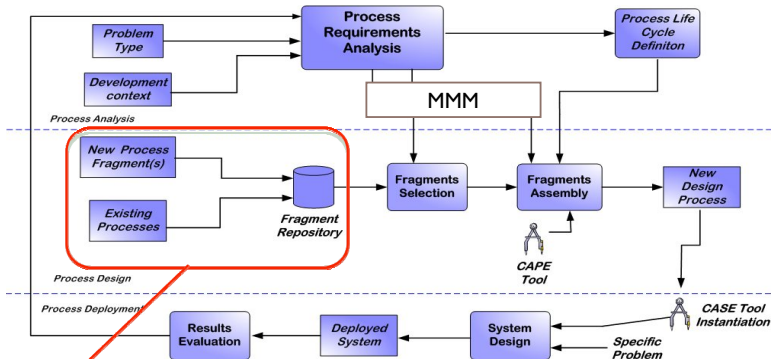
The reuse view



The implementation view

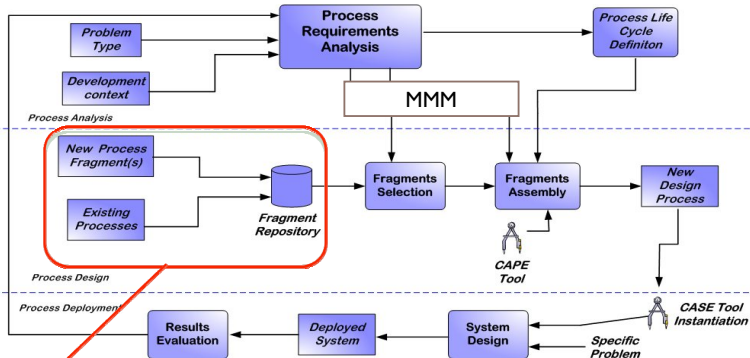


PRoDe divided in three main areas of research



1) A collection of process fragments

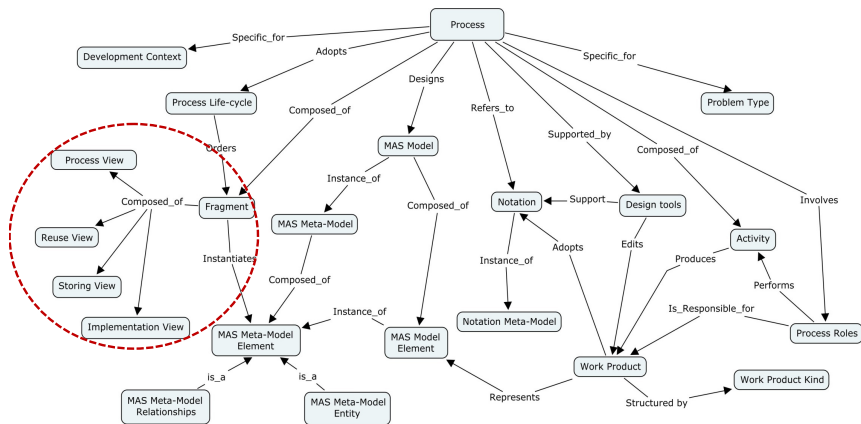
The fragment collection in PRoDe



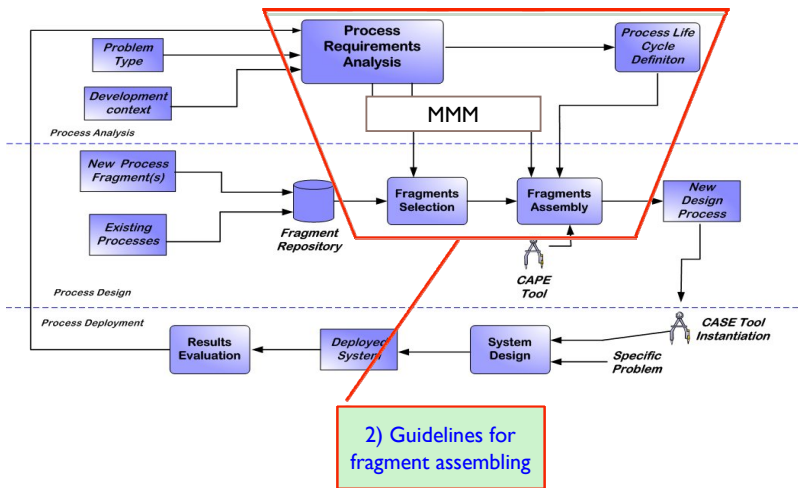
1) A collection of process fragments



The PRoDE Process Representation



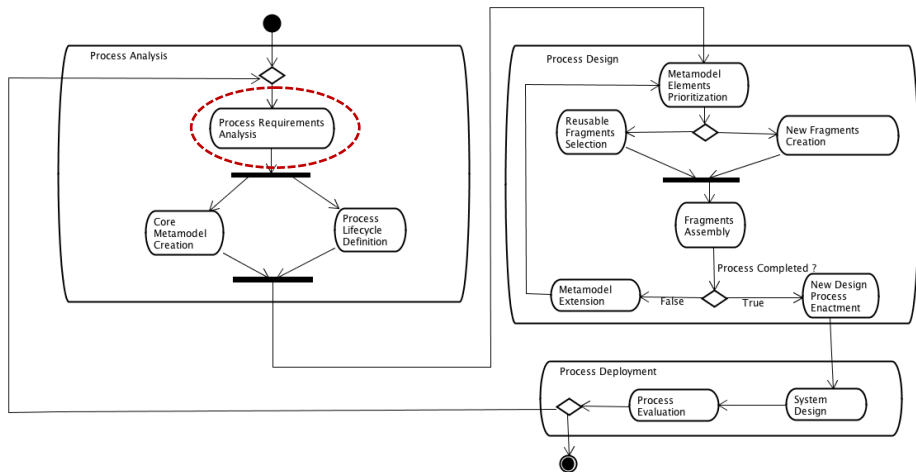
Guidelines for fragments assembling



2) Guidelines for
fragment assembling



Process Analysis and Design in PRoDe

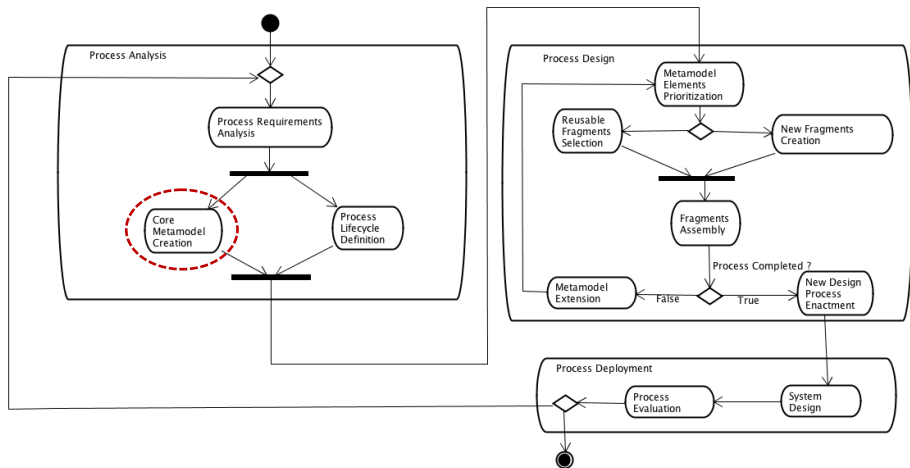


Example: PRoDe Analysis

ASPECS Process Requirement	Strategy	Consequence
Development of very large MASs for hierarchically decomposable problems	Adoption of holonic decomposition of problems	
Reuse of experiences done with PASSI	Support for functional requirements	
	Early identification of agents on the basis of requirements	
	Transformational approach	
	An ontology should be used to model agent's knowledge	
	FIPA-compliance at least at the communication level	
	Input of the process: text scenarios	



Process Analysis and Design in PRoDe

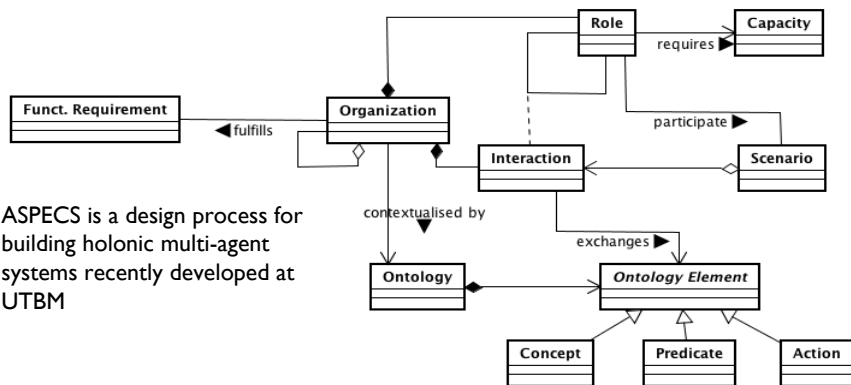


Example: Core meta-model creation

ASPECS Process Requirement	Strategy	Consequence		
		MMME from PASSI	MMME from CRIO	Other
Development of very large MASS for hierarchically decomposable problems	Adoption of holonic decomposition of problems		Capacity, Organization, Role, Interaction, Holon	Organizations, not agents should be the center of the process
Reuse of experiences done with PASSI	Support for functional requirements	Scenario, (Functional) Requirement		
	Early identification of agents on the basis of requirements	Link agent-requirement		Agents should be replaced by organizations
	Transformational approach			3 domains in the MMM
	An ontology should be used to model agent's knowledge	Ontology (including Concepts, Actions, Predicates)		
	FIPA-compliance at least at the communication level	Communication, Message, Interaction Protocol, Ontology, Role		
	Input of the process: text scenarios			Text Scenario is an input of the process



Example: ASPECS core meta-model

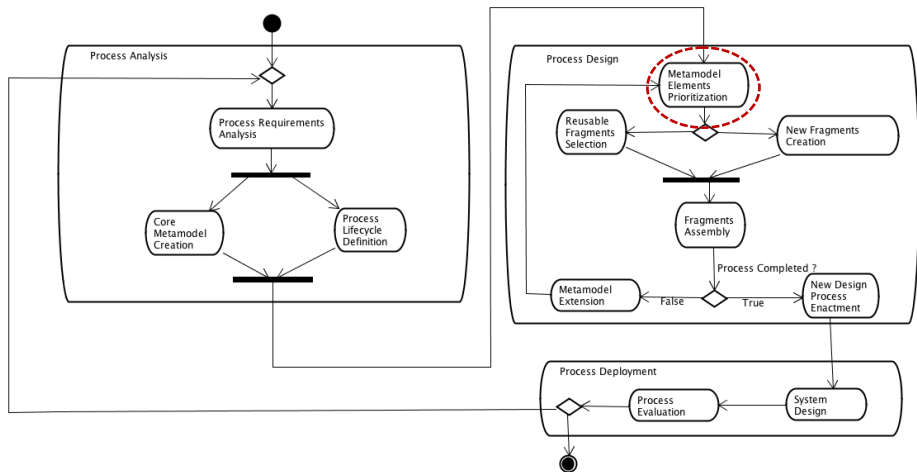


ASPECS is a design process for building holonic multi-agent systems recently developed at UTBM

- A detailed description of ASPECS in [Cosentino et al., 2010]

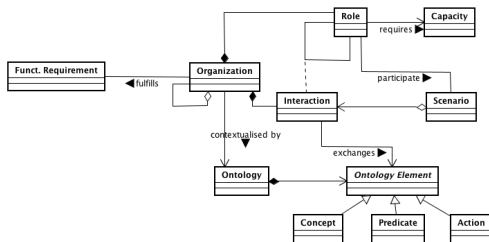


Process Analysis and Design in PRoDe



What is prioritization?

- The problem we face is:
 - What are the first fragments we should introduce in the new process?



??



The algorithm

- Main issues:
 - we assume each process fragment instantiates, relates, refines or quotes MAS Meta-Model Elements (MMMEs)
 - we created an algorithm for assigning a priority to the realisation of some MMMEs:
 - elements that are “leaves” of the meta-model graph are realised at first
 - other elements follow according to the number of their relationships
 - The output is a priority list of fragments



The Prioritization Algorithm (1 of 3) [Seidita et al., 2010]

1. Select a metamodel domain (consider the resulting metamodel as a graph with nodes (MMMEs) and edges (relationships))
2. Define List elements1 as a list of MMMEs that can be defined by reusing fragments from the repository, and the associated priority p : List elements1 (MMME, p), $p=1$;
3. Define List elements2 as a list of MMMEs that cannot be defined by reusing fragments from the repository;
4. Define List elements3 as a list of elements that are not in the core MMM;
5. While the core MMM is not empty
 - a) Select the leaves L_i ($i=1, \dots, n$) that: (i) can be instantiated by fragments of the repository and (ii) have less relationships with other elements
 1. Insert L_i ($i=1, \dots, n$) in List elements1;
 2. Remove elements L_i ($i=1, \dots, n$) from the core MMM;
 3. $p = p+1$;
6. While the core MMM is not empty
 - a) Select the leaves L_i ($i=1, \dots, m$) that can not be instantiated by fragments of the repository;
 1. Insert L_i ($i=1, \dots, m$) in List elements2;
 2. Remove L_i ($i=1, \dots, m$) from the core MMM;



The Prioritization Algorithm (2 of 3)

7. For each element $E1_i$ of List_elements1 select an instantiating fragment from the repository (*verify the correspondence among fragment rationale and the process requirements/strategies*)
 - a) If one fragment corresponds to process requirements and strategies then:
 - I. insert the fragment in the new process composition diagram
 - II. analyze inputs I_i ($i=0, \dots, n$) and outputs O_j ($j=0, \dots, m$) of the fragment
 - A. If some I_i or O_j does not belong to the core MMM then add it to List_elements3; mark the fragment as "To be modified"
 - B. remove $E1_i$ from List_elements1;
 - III. For each element $E2_i$ in List_elements2 analyze if there is a similarity with the elements defined in this fragment
 - A. if yes delete $E2_i$ from List_elements2 and I_i/O_i from List_elements3
 - b) else (if no fragment correspond to requirements and strategies) then
 - I. remove $E1_i$ from List_elements1 and insert it in List_elements2

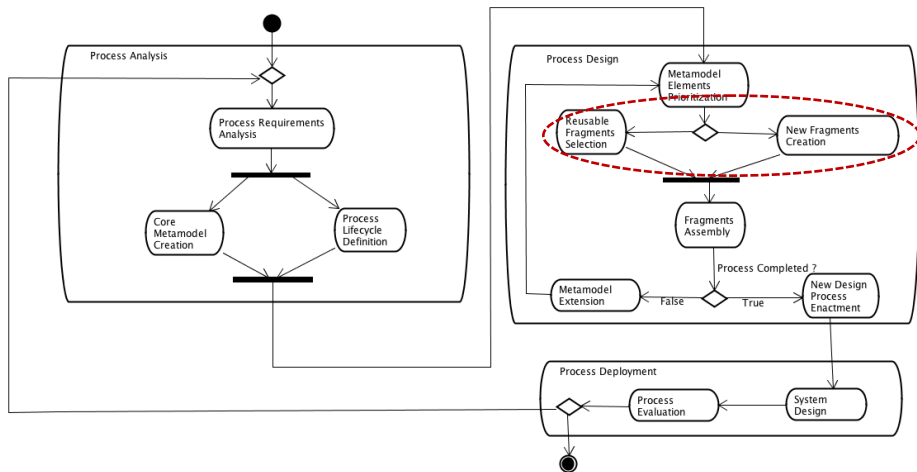


The Prioritization Algorithm (3 of 3)

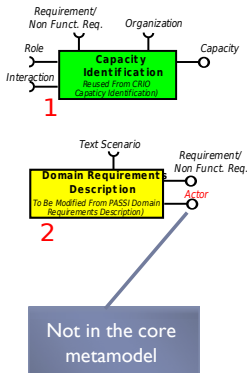
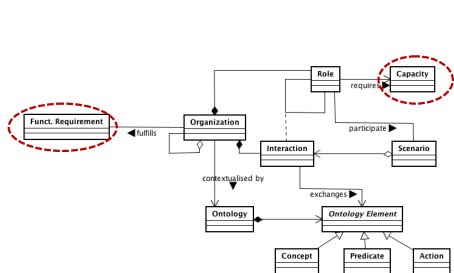
8. For each $E2_i$ ($i=0..m$) in `List_elements2`
 - a) Define a new fragment for instantiating $E2_i$
 - b) Insert the fragment in the new process composition diagram
 - c) Remove $E2_i$ from `List_elements2`
9. For each $E3_i$ ($i=0..m$) in `List_elements3`
 - a) Introduce elements $E3_i$ ($i=0..q$) from `List_elements3` in the core MMM
 - b) Repeat from 2. (consider only the new elements)
10. If the process is not completed (*i.e. not all design activities from requirements elicitation to coding, testing and deployment have been defined*)
 - a) Repeat from 1.



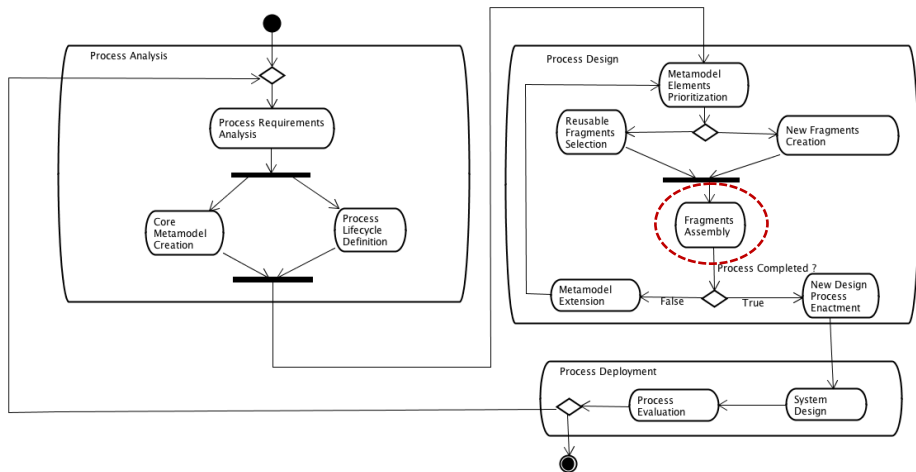
Process Analysis and Design in PRoDe



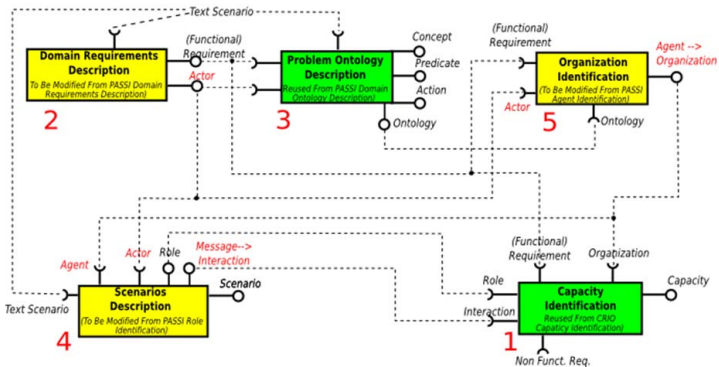
Example: the first two fragments in Building the ASPECS Process



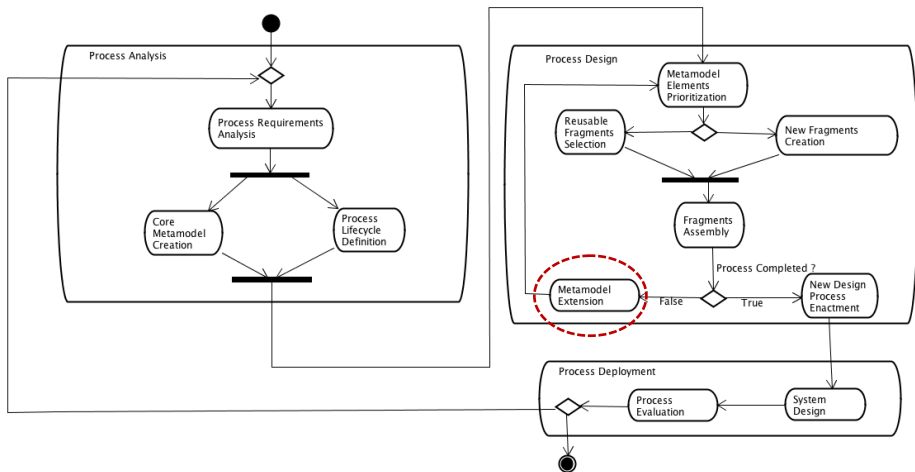
Process Analysis and Design in PRoDe



Example: Aspects process component diagram



Process Analysis and Design in PRoDe

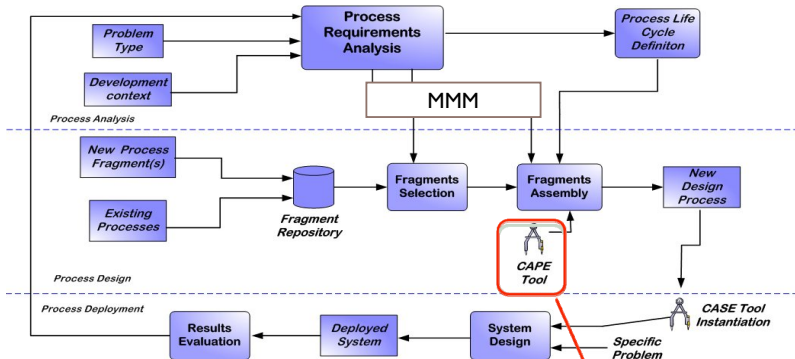


Meta-model Extension

- The Core MAS Meta-model is the starting point for selecting the right fragments from the repository and for assembling them in the new process
- MAS Meta-model extensions come from:
 - the need of incorporating MAS Meta-model Elements (MMMEs) referred in selected fragments
 - new process requirements
 - not all design activities from requirements elicitation to coding, testing and deployment have been defined
- Three different situations may arise:
 - different MAS meta-models contribute to the new one with parts that are totally disjointed
 - different MAS meta-models contribute to the new one with parts that overlap and...
 - ... overlapping elements have the same definitions bounded to elements with different names or on the contrary
 - ... overlapping elements have the same name but different definitions



Supporting Tool in PRoDE



3) A CAPE (Computer Aided Process Engineering) tool

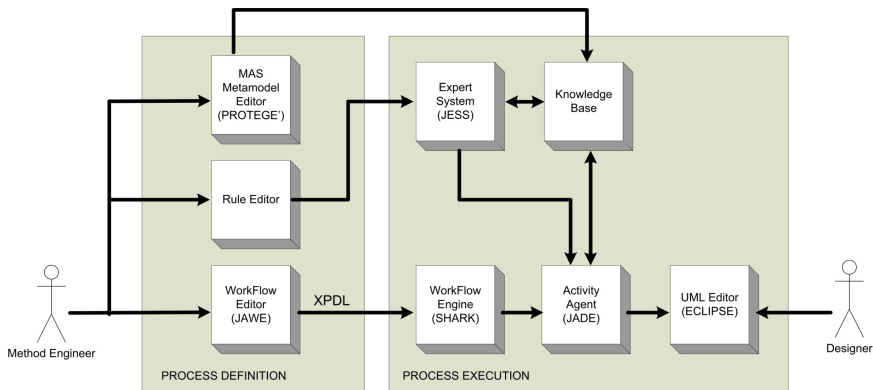


Metameth

- **Metameth** is an (open-source) agent-oriented tool we built to support our experiments in methodologies composition and their application in real projects.
- Metameth is:
 - a CAPE tool: since it supports the definition of the design process life-cycle and the positioning of the different method fragments in the intended place
 - a CAME tool: since it allows the definition of different method fragments
 - a CASE tool: since it supports a distributed design process, it offers several (by now UML) graphical editors and an expert system for verifying the resulting system



Metameth tool architecture



Supporting design activities

- The operations that can be supported by a tool during the design process:
 - GUI Action – the tool interacts with the user (using a GUI) in order to support him in some operations
 - WP Composition – the tool creates/updates a work product on the basis of the already introduced design information
 - Rule Check – semantic and syntactic check of the work product (warning, alerting and suggestions)

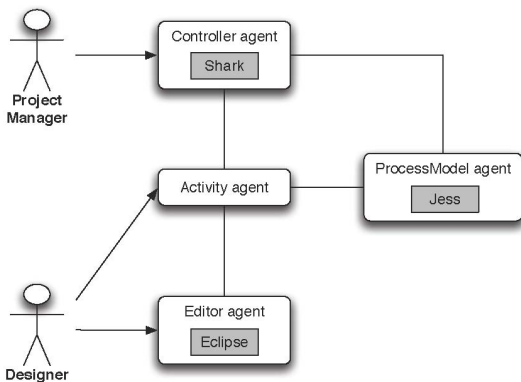


The expert system

- Metameth is composed of a society of agents interacting with users:
 - a controller agent – responsible for the execution of process
 - a community of Activity agents – interacting with designer
 - a ProcessModel agent – is responsible of managing the design information
 - an editor agent – manages the diagram editor



The expert system



The rules

- The Process Model agent is responsible of the activation of Jess rules
- Classification according to five categories:

– Validation

– Semantic interpretation

– Auto-composition

– Update

– Import

– Valid
– Sem

– Auto
– Upda
– Impo

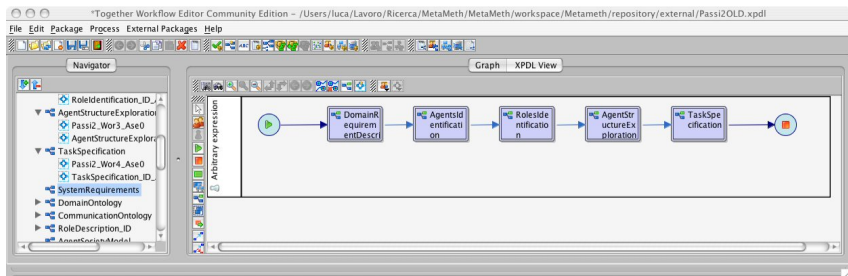


The expert system

- The Metameth expert system is based on JESS
- Rules are expressed in first order logic
- Ontology is designed using Protegè
- Services offered by the expert system:
 - syntax checks: it verifies the abidance to modelling language rules
 - semantic checks: it verifies the abidance to the MAS meta-model (e.g. a role cannot aggregate another one)
 - semantic understanding of diagrams: elements of notations are mapped to their corresponding MAS meta-model element (a use-case is mapped to a requirement)
 - automatic composition of diagrams: some diagrams can be partially composed by accessing information of previous design phases



The Metameth GUI



Outline

6 AOSE

7 Agent Oriented Methodologies

- MAS Meta-models
- AOSE Methodologies: An Overview
- Methodologies Documentation
- Methodology Challenges

8 Agent Oriented Situational Method Engineering

- Method Fragment Representation
- PRoDe: A Process for the Design of Design Processes
- Method Fragment extraction and Repository creation
- Result Evaluation



Method Fragment Extraction

- The **repository** is a data base where method fragments are stored in terms of (usually text) documents
- Fragments extraction is Work Product- and MMM Element-oriented
- A fragment is identified as a portion of process that produces a significant work product (a diagram or other kind of WP)
 - fragments can also be composed: Phase fragment, Composed fragment, Atomic fragment



The Categorisation [Seidita et al., 2006]

- The aim is to unify different elements (from different approaches) under a unique definition
 - a set of common phases of software engineering design processes
 - the principal process role performing these phases
 - a set of work product kind
- The repository allows the classification of fragments according to a set of categories based on the most important meta-model elements
 - Phase
 - Process Role
 - Work Product
 - MMM Element



The need for a taxonomy

- All the processes we studied were created by different research groups and deal with different design philosophies
- Differences in names and definitions of the design process elements
 - sixteen different process roles
 - seventeen phases
 - several work products and MAS Meta-model elements



Phases

- Any kind of design process can be decomposed in phases
 - High level of abstraction for phases resulting from the studied processes
 - Some of them are specific for agent based design process
- Requirements
 - Analysis
 - Design
 - Implementation
 - Testing
 - Deployment
 - Coding

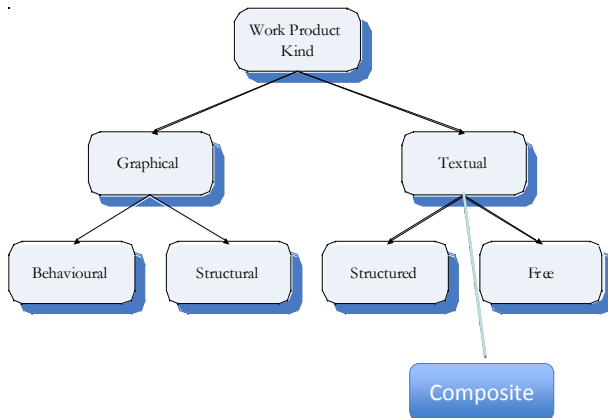


Process Roles

- Identification of an high level process role for each phase
 - Detailing process roles basing on studied processes
- System Analyst
 - Domain Analyst
 - User
 - Agent Analyst
 - Agent Designer
 - User Interface Designer
 - Programmer
 - Test Designer
 - Test Developer



Taxonomy: Work product



The need for a taxonomy

- Three kinds of MAS Meta-model elements
 - Problem domain → all aspects of users problem description including environment representation
 - Agency Domain → agent based concepts useful to define a solution
 - Solution Domain → the structure of the code solution



Repository content

Repository	
Phase	
Requirements	8
Analysis	16
Design	11
Implementation	4
Testing	1
Deployment	1
Coding	3
Process Role	
System Analyst	14
Domain Analyst	7
User	1
Agent Analyst	6
Agent Designer	11
User Interface Designer	1
Programmer	2
Test Designer	1
Test Developers	1
WorkProduct Kind	
Behavioural	9
Structural	24
Structured	6
Free	14
MMM Element	
Problem	12
Social	45
Solution	2



Method fragment retrieval

Applet HTML Page - Microsoft Internet Explorer

File Modifica Visualizza Preferiti Strumenti ?

Indietro - - - - - Cerca Preferiti - - - - -

Ingrasso <http://mozart.csai.unipa.it:8080/appletfragment/AppletMain.html> Vai Collegamenti

Google - - - - - Cerca - - - - - PageRank - - - - - Popup OK - - - - - ABC Ortografia - - - - - Opzioni - - - - - SnagIt

Phases/Activities	Process Roles	Work Product Kind	MMM Element
<input type="checkbox"/> Requirements	<input type="checkbox"/> System Analyst	<input type="checkbox"/> Behavioural	<input type="checkbox"/> Problem
<input type="checkbox"/> Analysis	<input type="checkbox"/> Domain Analyst	<input type="checkbox"/> Structural	<input type="checkbox"/> Social
<input type="checkbox"/> Design	<input type="checkbox"/> User	<input type="checkbox"/> Structured	<input type="checkbox"/> Solution
<input type="checkbox"/> Implementation	<input type="checkbox"/> Agent Analyst	<input type="checkbox"/> Free	
<input type="checkbox"/> Testing	<input type="checkbox"/> Agent Designer		
<input type="checkbox"/> Deployment	<input type="checkbox"/> User Interface Designer		
<input type="checkbox"/> Coding	<input type="checkbox"/> Programmer		
	<input type="checkbox"/> Test Designer		
	<input type="checkbox"/> Test Developer		

Search by Name

Applet appletfragment/AppletMain started

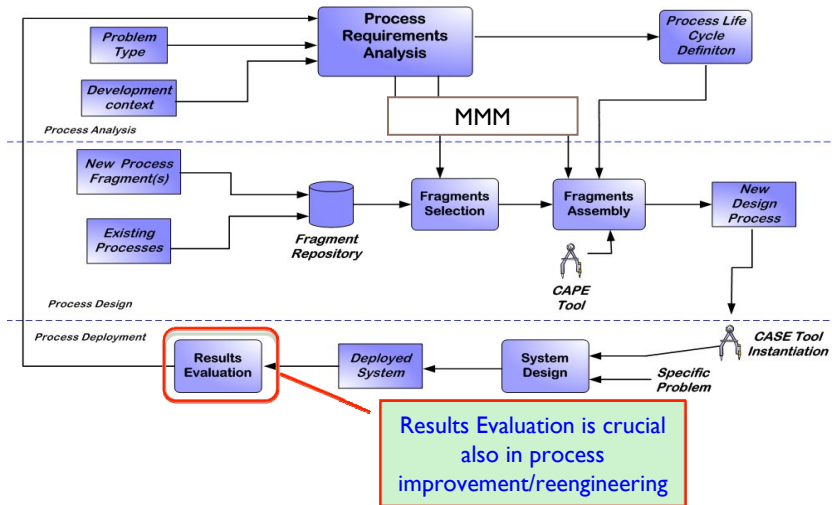
Internet

Outline

- 6 AOSE
- 7 Agent Oriented Methodologies
 - MAS Meta-models
 - AOSE Methodologies: An Overview
 - Methodologies Documentation
 - Methodology Challenges
- 8 Agent Oriented Situational Method Engineering**
 - Method Fragment Representation
 - PRoDe: A Process for the Design of Design Processes
 - Method Fragment extraction and Repository creation
 - **Result Evaluation**



Results Evaluation: an open problem?



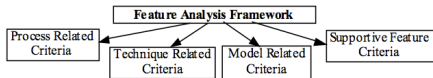
AO Design Process Evaluation

- Q.N. Tran, G. C. Low (2005). Comparison of Ten Agent-Oriented Methodologies. In Agent-Oriented Methodologies, chapter XII, pp. 341-367. Idea Group.
- L. Cernuzzi, G. Rossi (2002). On the evaluation of agent oriented methodologies. In: Proc. of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies, pp. 21-30.
- Arnon Sturm, Dov Dori, Onn Shehory (2004). A Comparative Evaluation of Agent-Oriented Methodologies, in Methodologies and Software Engineering for Agent Systems, Federico Bergenti, Marie-Pierre Gleizes, Franco Zambonelli (eds.)
- Khanh Hoa Dam, Michael Winikoff (2003). Comparing Agent-Oriented Methodologies. In proc. of the Agent-Oriented Information Systems Workshop at AAMAS03. Melbourne (AUS).
- P. Cuesta, A. Gomez, J. C. Gonzalez, and F. J. Rodriguez (2003). A Framework for Evaluation of Agent Oriented Methodologies. CAEPIA'2003
- L. Cernuzzi, M. Cossentino, F. Zambonelli (2005). Process Models for Agent-Based Development. International Journal on Engineering Applications of Artificial Intelligence (EAAI). Elsevier.



Details on AO processes evaluation

[Numi Tran and Low, 2005]



Structure of the evaluation framework

	GAIA	TROPOS	MAS-COMMONAKADS	PROMETHEUS	PASSI
Development lifecycle	Iterative within each phase but sequential between phases	Iterative and incremental	Cyclic risk-driven process	Iterative across all phases	Iterative across and within all phases (except for coding and deployment)
Coverage of the lifecycle	Analysis and Design	Analysis and Design	Analysis and Design	Analysis and Design	Analysis, Design and Implementation
Development perspective	Top-down	Top-down	Hybrid	Bottom-up	Bottom-up
Application domain	Independent (business process management, GIS, traffic simulation)	Independent (e-business systems, knowledge management, health IS)	Independent (Flight reservation, automatic control)	Independent (holonic manufacturing, online bookstore)	Independent (distributed robotics applications, online bookstore)
Size of MAS	<= 100 agent classes	Not specified	Not specified, but possibly any size	Any size	Not specified
Agent nature	Heterogeneous	BDI-like agents	Heterogeneous	BDI-like agents	Heterogeneous
Support for verification and validation	No	Yes	Mentioned but no explicit steps/guidelines provided	Yes	Yes



Details on AO Processes Evaluation

[Sturm and Shehory, 2004]

- Evaluation is based on
 - concepts and properties (autonomy, proactiveness, ...)
 - notations and modeling techniques (accessibility, expressiveness)
 - process (development context, Lifecycle coverage)
 - pragmatics (required expertise, scalability, ...)



Details on AO processes evaluation

- From:
 - Khanh Hoa Dam, Michael Winikoff (2003). Comparing Agent-Oriented Methodologies. In proc. of the Agent-Oriented Information Systems Workshop at AAMAS03. Melbourne (AUS).

- Based on a questionnaire
- Reused and extended in AL3-AOSE TFG3 [AgentLink III, 2006]

Concept/Property	Adelfe	Gaia	Ingenias	OPF	PASSI	Prome-theus	TROPOS
Autonomy	H	H	H	H	H/H/M	H	L
Mental attitudes	L	N	H	H	L/L/M	M	M
Proactiveness	M	L	H	H	H/M/H	H	N
Reactiveness	H	L	H	H	H/H/H	H	N
Concurrency	H	M	H	L	H/H/M	H	L
Teamwork and roles	L	H	H	H	M/H/H	L	M
Cooperation model	AMAS th.	Teamwork	ALL	ALL	Task del./ Teamwork	none	Negotiation/ Task del.
Protocols support	H	H	H	H	H/M/H	H	N
Communication modes	ALL	Async mess.	ALL	ALL	Direct	N	
Communication language	ALL	ACL like	ALL	ALL	Speech acts	messages	
Situatedness	H	H	H	H	H/M/M	H	H
Environment type	All episodic	Dynamic Continuous	All discrete	ALL	ALL	ALL	Inacc., Non episodic, Dynam.



Details on AO processes evaluation

- The Capability Maturity Model Integration (CMMI) [Software Engineering Institute (SEI), 2010]
 - *The overall goal of CMMI is to provide a framework that can share consistent process improvement best practices and approaches, but can be flexible enough to address the rapidly changing needs of the community*
 - SCAMPI (Standard CMMI Assessment Method for Process Improvement)[Software Engineering Institute (SEI), 2006] it is a schema for process evaluation in five steps: activation, diagnosis, definition, action, learning.



Details on AO processes evaluation: CMMI discrete levels

- Levels are used in CMMI to describe an evolutionary path recommended for an organisation that wants to improve the processes
- The maturity level of an organisation provides a way to predict an organisation's performance in a given discipline or set of disciplines
- A **maturity level** is a defined evolutionary plateau for organisational process improvement



Details on AO processes evaluation: CMMI discrete levels

Maturity Level	Description
1-Initial	processes are usually ad hoc and chaotic
2-Managed	processes are planned and executed in accordance with policy
3-Defined	processes are well characterized and understood, and are described in standards, procedures, tools, and methods
4-Quantitatively managed	the organization and projects establish quantitative objectives for quality and process performance and use them as criteria in managing processes
5-Optimizing	an organization continually improves its processes based on a quantitative understanding of the common causes of variation inherent in processes

AOSE processes are (at most) at level 3!!
(only a few of them)



Open issues

- SME is perceived to be a difficult discipline
 - this is only partially true. All new design processes creator performed (usually in a disordered way) the steps proposed and studied by SME
 - agreater diffusion of AO-SME can have positive effects on the development of new AO design processes (specifically in new areas like self-org)
- Major problems with AO-SME
 - AO processes deals with MAS meta-models and they are an open issue in the agent community
 - lack of standards (ISO specification vs FIPA proposal)
 - lack of standard repository of fragments
 - lack of stable (commercial quality) CAPE/CAME tools
 - design process evaluation is still an open issue in both AO and OO software engineering



Part III

Research Directions and Conclusion



Outline

9 Research Directions & Vision

10 Conclusions



Mainstream AOSE Researches

- Methodology
 - dozens of methodologies proposed so far
 - mostly “pencil and papers” exercises with no confrontation with real world problems. . .
- Meta-methodologies
 - interesting and worth to be explored, but. . .
 - these would require much more research coordination and more feedback from real-world experiences
- Models & Notations
 - of great help to clarify agent-oriented abstractions
 - no specific standard still exists
- Infrastructures
 - Very interesting models but. . .
 - (the lack of) a pure agent-oriented language slows down the implementation phase



Is This Enough?

- Let's ask ourselves a simple basic question:
 - what does it mean engineering a MAS?
 - what is the actual subject of the engineering work?
- What is a MAS in a world of:
 - world-wide social and computational networks
 - pervasive computing environments
 - sensor networks and embedded computing
- There is not a single answer:
 - it depends on the observation level
- In the physical world and in micro-electronics [Zambonelli and Omicini, 2004]
 - micro level of observation: dominated by quantum phenomena (and and to be studied/engineered accordingly)
 - macro level of observation: dominated by classical physics
 - meso level of observation: quantum and classical phenomena both appears (and have to be taken into account)



AOSE Observation Levels

- Micro scale
 - small- medium-size MASs
 - control over each component (limited complexity single stakeholder)
 - this is the (only) focus of mainstream AOSE
- Macro scale
 - very large scale distributed MASs
 - no control over single components (decentralization, multiple stakeholders)
 - the kingdom of “self-organisation” people
- Meso scale
 - micro scale components deployed in a macro scale scenario
 - my own system influence and is influenced by the whole
- Very rarely a fully fledged study can be limited to a single level of observation
 - most MASs (even small scale) are open
 - deployed in some sort of macro scale system
 - dynamically evolving together with the system



Micro-Level Challenges (1)

- Assessing AOSE Advantages
 - AO has clear advantages. What about AOSE?
 - methodologies, methodologies, methodologies... ;-(
 - qualitative work
 - we need to show that AOSE
 - helps saving money and human resource
 - leads to higher quality software products
 - quantitative comparison of AOSE vs. non-AOSE complex software development
- Pay Attention to the Software Process
 - most methodologies assume a “waterfall” model
 - either implicitly or explicitly
 - with no counterpart in industrial software development
 - Need for:
 - agile processes
 - agent-specific flexible processes
 - Cf. Knublauch 2002 “Extreme programming for MAS”, Cossentino 2006: “Agile PASSI”
 - Can meta-models be of help in that direction?



Micro-Level Challenges (2)

- Agent-specific notations
 - AUML is ok to spread acceptance but...
 - is it really suited for MASs?
 - and for complex systems in general?
 - even the mainstream SE community doubts about that...
 - do more suitable notations exist?
 - agent-specific ones to be invented
 - other non-UML approaches
 - Cf. Sturm et al. 2003: OPM/MAS
 - AML by Whitestein [Cervenka et al., 2005]
 - A proposal of unified notation by L. Padgham, M. Winikoff, S. De Loach, M. Cossentino [Padgham et al., 2009]
- Intelligence engineering
 - selling AI has always been difficult
 - lack of engineering flavor...
 - agents can help with that
 - embodied, modular, intelligence
 - observable rationality
 - our role should be that of:
 - exploiting scientific results from the AI-oriented MAS community
 - turn them into usable engineered products



Macro-Level Challenges (1)

- The macro level deals with complex collective behavior in large scale MASs
 - some say this is not AOSE...
 - scientific activity
 - observing and reproducing biology
 - but it must become an engineering activity
 - challenging indeed
- Universality in MASs
 - can general laws underlying the behavior of complex MASs be identified?
 - as they are starting being identified in the “complex systems” research community
 - phase transitions, edges of chaos, etc.
 - letting us study and engineer complex MAS
 - abstracting from the specific characteristics of agents (from ants to rational BDI agents)
 - abstracting from the specific content of their interactions
 - Cf. Van Parunak 2004: “Universality in MAS”



Macro-Level Challenges (2)

- Measuring Complex MASs
 - how can we characterize the behavior of large-scale MASs?
 - when we cannot characterise the behavior of single components
 - macro-level measures must be identified
 - to concisely express properties of a system
 - Cf. Entropy, Macro-properties of complex network, etc
 - and tools must be provided to actually measure systems
 - but measuring must be finalised
- Controlling Complex MASs
 - given a measurable property of a MASs
 - software engineers must be able to direct the evolution of a system, i.e., to tune the value of the measurable property
 - in a fully decentralised way
 - and with the possibility of enforcing control over a limited portion of the MAS
 - software engineering will become strictly related to control systems engineering
- Emergent behaviors, physics, biology, etc
 - Cf. The activity of the “SELF ORGANIZATION” Agentlink group



Meso-Level Challenges (1)

- It is a problem of deployment
 - engineering issues related to deployment of a MAS (typically engineered at a micro level of observation). . .
 - . . . into a large scale system (to be studied at a macro-level of observation)
- Impact Analysis
 - how will my system behave when it will deployed in an existing open possibly large scale networked system?
 - how I will influence the existing system?
 - micro-scale aspects:
 - tolerance to unpredictable environmental dynamics on my system
 - internal handlings
 - macro-scale aspect:
 - can my “small” MAS change the overall behavior of the global system?
 - “butterfly effect”?



Meso-Level Challenges (2)

- Identifying the Boundaries
 - how can I clearly identify what is part of my system and what is not?
 - I should identify
 - potential inter-agent and environmental interactions
 - shape the environment (i.e., via agentification)
 - engineer the interactions across the environment
 - in sum: engineering the boundaries of the system
- Trust
 - I can (provably) trust a “small” system of rational agents
 - I can (probabilistically) trust a very large-scale MASs
 - what I can actually say about the small system deployed in the large-scale one
 - how can I measure the “degree of trust”?
- Infrastructures for Open Systems
 - are configurable context-dependent coordination infrastructure the correct answer?
 - are normative approaches the correct ones?
 - We know what we gain but we do not know what we lose
 - Cf. Incentives in social and P2P networks



Research directions and visions: conclusions

- There is not a single AOSE
 - depends on the scale of observation. . .
- The micro scale
 - overwhelmed by research
 - often neglecting very basic questions. . .
- The macro scale
 - some would say this is not AOSE
 - but it must become indeed. . .
- The meso scale
 - fascinating. . .
 - very difficult to be tackled with engineering approaches. . .
- What else?
 - there is so much to engineer around. . .
 - emotional agents, mixed human-agent organisations, interactions with the physical world. . .



Outline

9 Research Directions & Vision

10 Conclusions



Reflections

- In this lesson we have spoken about the Software Engineering and the Agent Oriented Software Engineering
- Some reflections are necessary:
 - What are the aspects related to Engineering?
 - What are the aspects related to Software Engineering?
 - What are the aspects related to the paradigms adopted?
- Before proceeding it is necessary to clarify what is the Engineering in general



What is Engineering?

- In general Engineering is the applied science of acquiring and applying knowledge to *design, analysis, and/or construction of works for practical purposes*
- The American Engineers' Council for Professional Development defines:

Engineering

The creative application of scientific principles to design or develop structures, machines, apparatus, or manufacturing processes, or works utilizing them singly or in combination; or to construct or operate the same with full cognizance of their design; or to forecast their behavior under specific operating conditions; all as respects an intended function, economics of operation and safety to life and property



Engineers

- Engineers borrow from physics and mathematics to find suitable solutions to the problem at hand
- They apply the scientific method in deriving their solutions: if multiple options exist, engineers weigh different design choices on their merits and choose the solution that best matches the requirements
- The crucial and unique task of the engineer is to identify, understand, and interpret the constraints on a design in order to produce a successful result
- Constraints may include available resources, physical, imaginative or technical limitations, flexibility for future modifications and additions, and other factors, such as requirements for cost, safety, marketability, productibility, and serviceability
- By understanding the constraints, engineers derive specifications for the limits within which a viable object or system may be produced and operated



What are the Aspects Related to Engineering?

- Following a clear and disciplined development process
- Adopting a design methodology
- Creating an appropriate (mathematical) model of a problem that allows to analyse it
- Testing potential solutions
- Evaluating the different design choices and choosing the solution that best meets requirements
- Using of: prototypes, scale models, simulations, destructive tests, nondestructive tests, and stress tests



What are the Aspects Related to Software Engineering?

- Customization to the specific *kind of product*: Software
 - Specific software development processes tied to the *software lifecycle*
 - Specific methodologies
 - Specific kinds of model tied to the concept of *software product*
 - Testing potential solutions
 - Using of specific techniques for: prototypes, scale models, simulations, tests, and stress tests



What are the Aspects Related to the paradigm?

- The building blocks for creating the models
- The level of *thinking / abstraction*
- Functions, objects, agents lead to different ways of *thinking* both the problems and the solutions
 - The paradigm adopted leads to different levels of *model complexity*: complicated problems are well captured by objects and agents, while functions could lead to have very very complex models for representing the problem
 - In the same way the models of the solution are heavily influenced by the paradigm



9 Research Directions & Vision

10 Conclusions



Bibliography I



AgentLink III (2006).

Agent-Oriented Software Engineering Technical Forum Group (AOSE TFG).

<http://www.pa.icar.cnr.it/cossentino/al3tf3/>.



Artikis, A., Picard, G., and Vercoeur, L., editors (2009).

Engineering Societies in the Agents World IX, volume 5485 of *Lecture Notes in Computer Science*.

Springer.

9th International Workshop (ESAW'08), 24–26 September 2008, Saint-Étienne, France.

Revised Selected Papers.



Bernon, C., Camps, V., Gleizes, M.-P., and Picard, G. (2005).

Engineering adaptive multi-agent systems: the ADELFE methodology.

In *Agent Oriented Methodologies*, chapter VII, pages 172–202. Idea Group Publishing.



Bernon, C. and Capera, Davyand Mano, J.-P. (2008).

Engineering self-modeling systems: Application to biology.

In [Artikis et al., 2009], pages 248–263.

9th International Workshop (ESAW'08), 24–26 September 2008, Saint-Étienne, France.

Revised Selected Papers.



Bibliography II



Bernon, C., Cosentino, M., Gleizes, M. P., Turci, P., and Zambonelli, F. (2004).
A study of some multi-agent meta-models.

In [Odell et al., 2005], pages 62–77.

5th International Workshop (AOSE 2004), New York, NY, USA, 19 July 2004. Revised Selected Papers.



Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., and Perini, A. (2004).

Tropos: An agent-oriented software development methodology.

Autonomous Agent and Multi-Agent Systems, 3:203–236.



Brinkkemper, S. (1996).

Method engineering: engineering of information systems development methods and tools.

Information & Software Technology, 38(4):275–280.



Brinkkemper, S., Saeki, M., and Harmsen, F. (1999).

Meta-modelling based assembly techniques for situational method engineering.

Information Systems, 24(3):209–228.



Capera, D., Picard, G., and Gleizes, M.-P. (2004).

Applying ADELFE methodology to a mechanism design problem.

In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 1508–1509, New York, NY, USA. IEEE CS.



Bibliography III



Cernuzzi, L., Cossentino, M., and Zambonelli, F. (2005).
Process models for agent-based development.
Engineering Applications of Artificial Intelligence, 18(2):205–222.



Cernuzzi, L., Molesini, A., Omicini, A., and Zambonelli, F. (2010).
Adaptable multi-agent systems: The case of the Gaia methodology.
International Journal of Software Engineering and Knowledge Engineering.



Cervenka, R., Trencansky, I., Calisti, M., and Greenwood, D. (2005).
AML: Agent modeling language toward industry-grade agent-based modeling.
In [Odell et al., 2005], pages 31–46.
5th International Workshop (AOSE 2004), New York, NY, USA, 19 July 2004. Revised
Selected Papers.



Cossentino, M. (2005).
From requirements to code with the PASSI methodology.
In [Henderson-Sellers and Giorgini, 2005], chapter IV, pages 79–106.



Cossentino, M., Fortino, G., Garro, A., Mascillaro, S., and Russo, W. (2008).
PASSIM: A simulation-based process for the development of multi-agent systems.
International Journal on Agent Oriented Software Engineering, 2(2):132–170.



Bibliography IV



Cossentino, M., Gaglio, S., Garro, A., and Seidita, V. (2007a).
Method fragments for agent design methodologies: from standardisation to research.
International Journal of Agent Oriented Software Engineering, 1(1):91–121.



Cossentino, M., Gaglio, S., and Valeria, S. (2007b).
Adapting PASSI to support a goal oriented approach: A situational method engineering
experiment.
In *5th European Workshop on Multi-Agent Systems (EUMAS'07)*.



Cossentino, M., Gaud, N., Hilaire, V., Galland, S., and Koukam, A. (2010).
ASPECS: An agent-oriented software process for engineering complex systems.
Autonomous Agents and Multi-Agent Systems, 20(2):260–304.



DeLoach, S. A., Padgham, L., Perini, A., Susi, A., and Thangarajah, J. (2009).
Using three AOSE toolkits to develop a sample design.
International Journal of Agent Oriented Software Engineering, 3(4):416–476.



Fuggetta, A. (2000).
Software process: A roadmap.
In *22nd International Conference on on Software Engineering (ICSE 2000), Future of
Software Engineering Track*, pages 25–34, New York, NY, USA. ACM Press.



Bibliography V



García-Magariño, I., Gutiérrez, C., and Fuentes-Fernández, R. (2009).
The INGENIAS development kit: A practical application for crisis-management.
In *10th International Work-conference on Artificial Neuronal Networks (IWANN 2009)*,
Salamanca, Spain.



Ghezzi, C., Jazayeri, M., and Mandrioli, D. (2002).
Foundamental of Software Engineering.
Prentice Hall, second edition.



Grasia Group (2009).
INGENIAS meta model.
<http://grasia.fdi.ucm.es/main/?q=en/node/135>.



Hadar, I., Kuflik, T., Perini, A., Reinhartz-Berger, I., Ricca, F., and Susi, A. (2010).
An empirical study of requirements model understanding: Use case vs. Tropos models.
In *2010 ACM Symposium on Applied Computing (SAC 2010)*, pages 2324–2329, New
York, NY, USA. ACM.



Henderson-Sellers, B. (2003).
Method wngineering for OO systems development.
Communications of the ACM, 46(10):73–78.



Bibliography VI



Henderson-Sellers, B. (2005).

Creating a comprehensive agent-oriented methodology: Using method engineering and the OPEN metamodel.

In [Henderson-Sellers and Giorgini, 2005], chapter XIII, pages 236–397.



Henderson-Sellers, B. and Giorgini, P., editors (2005).

Agent Oriented Methodologies.

Idea Group Publishing, Hershey, PA, USA.



IEEE FIPA Design Process Documentation and Fragmentation Working Group (DPDF) (2009).

Homepage.

<http://www.pa.icar.cnr.it/cossentino/fipa-dpdf-wg/>.



Kruchten, P. (2003).

The Rational Unified Process: An Introduction.

Addison-Wesley Professional, 3rd edition.



MEnSA Project (2007–2008).

Methodologies for the engineering of complex software systems: Agent-based approach.

<http://www.mensa-project.org/>.



Bibliography VII



Molesini, A., Denti, E., and Omicini, A. (2009).
RBAC-MAS & SODA: Experimenting RBAC in AOSE.
In [Artikis et al., 2009], pages 69–84.
9th International Workshop (ESAW'08), 24–26 September 2008, Saint-Étienne, France.
Revised Selected Papers.



Molesini, A., Denti, E., and Omicini, A. (2010).
Agent-based conference management: A case study in SODA.
International Journal of Agent-Oriented Software Engineering, 4(1):1–31.



Molesini, A., Nardini, E., Denti, E., and Omicini, A. (2008).
Advancing object-oriented standards toward agent-oriented methodologies: SPEM 2.0 on SODA.
In Baldoni, M., Cossentino, M., De Paoli, F., and Seidita, V., editors, *9th Workshop "From Objects to Agents" (WOA 2008) – Evolution of Agent Development: Methodologies, Tools, Platforms and Languages*, pages 108–114, Palermo, Italy. Seneca Edizioni.



Numi Tran, Q.-N. and Low, G. C. (2005).
Comparison of ten agent-oriented methodologies.
In [Henderson-Sellers and Giorgini, 2005], chapter XII, pages 341–367.



Bibliography VIII



Object Management Group (2008).
Software & Systems Process Engineering Meta-Model Specification 2.0.
<http://www.omg.org/spec/SPEM/2.0/PDF>.



Odell, J., Giorgini, P., and Müller, J. P., editors (2005).
Agent-Oriented Software Engineering V, volume 3382 of *Lecture Notes in Computer Science*.
Springer.
5th International Workshop (AOSE 2004), New York, NY, USA, 19 July 2004. Revised Selected Papers.



Omicini, A., Ricci, A., and Viroli, M. (2006).
Agens Faber: Toward a theory of artefacts for MAS.
Electronic Notes in Theoretical Computer Sciences, 150(3):21–36.
1st International Workshop “Coordination and Organization” (CoOrg 2005),
COORDINATION 2005, Namur, Belgium, 22 April 2005. Proceedings.



OPEN Working Group (1997).
OPEN home page.
<http://www.open.org.au/>.



Bibliography IX



Padgham, L. and Winikof, M. (2003).
Prometheus: A methodology for developing intelligent agents.
In Giunchiglia, F., Odell, J., and Weiss, G., editors, *Agent-Oriented Software Engineering III*, volume 2585 of *LNCS*, pages 174–185. Springer.
3rd International Workshop (AOSE 2002), Bologna, Italy, 15 July 2002. Revised Papers and Invited Contributions.



Padgham, L. and Winikoff, M. (2005).
Prometheus: A practical agent oriented methodology.
In [Henderson-Sellers and Giorgini, 2005], chapter V, pages 107–135.



Padgham, L., Winikoff, M., Deloach, S., and Cossentino, M. (2009).
A unified graphical notation for AOSE.
In Luck, M. and Gomez-Sanz, J. J., editors, *Agent-Oriented Software Engineering IX*, pages 116–130. Springer.



Pavòn, J., Gómez-Sanz, J. J., and Fuentes, R. (2005).
The INGENIAS methodology and tools.
In [Henderson-Sellers and Giorgini, 2005], chapter IX, pages 236–276.



Bibliography X



Ralyté, J. and Rolland, C. (2001a).

An approach for method reengineering.

In Kunii, H. S., Jajodia, S., and Sjølvberg, A., editors, *Conceptual Modeling – ER 2001*, volume 2224 of *Lecture Notes in Computer Science*, pages 471–484. Springer.

20th International Conference on Conceptual Modeling, Yokohama, Japan, 27–30 November 2001. Proceedings.



Ralyté, J. and Rolland, C. (2001b).

An assembly process model for method engineering.

In Dittrich, K. R., Geppert, A., and Norrie, M. C., editors, *Advanced Information Systems Engineering*, volume 2068 of *LNCS*, pages 267–283. Springer.

13th International Conference (CAiSE 2001), Interlaken, Switzerland, 4–8 June 2001, Proceedings.



Seidita, V., Cossentino, M., and Gaglio, S. (2006).

A repository of fragments for agent systems design.

In *Workshop “From Objects to Agents” (WOA 2006)*, pages 130–137.



Seidita, V., Cossentino, M., and Gaglio, S. (2009).

Using and extending the SPEM specifications to represent agent oriented methodologies.

In Luck, M. and Gomez-Sanz, J. J., editors, *Agent-Oriented Software Engineering IX*, pages 46–59. Springer, Berlin, Heidelberg.



Bibliography XI



Seidita, V., Cossentino, M., Hilaire, V., Gaud, N., Galland, S., Koukam, A., and Gaglio, S. (2010).

The metamodel: a starting point for design processes construction.

International Journal of Software Engineering and Knowledge Engineering, 20(4):575–608.



Software Engineering Institute (SEI) (2006).

Standard CMMI appraisal method for process improvement (SCAMPI) a, version 1.2: Method definition document.

Technical report, Carnegie Mellon University.



Software Engineering Institute (SEI) (2010).

Capability maturity model integration (CMMI).

<http://www.sei.cmu.edu/cmmi/>.



Sommerville, I. (2007).

Software Engineering.

Addison-Wesley, 8th edition.



Bibliography XII



Sturm, A. and Shehory, O. (2004).

A comparative evaluation of agent-oriented methodologies.

In Bergenti, F., Gleizes, M.-P., and Zambonelli, F., editors, *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, chapter 7, pages 127–149. Kluwer Academic Publishers.



Susi, A., Perini, A., Mylopoulos, J., and Giorgini, P. (2005).

The Tropos metamodel and its use.

Informatica, 29(4):401–408.



Wooldridge, M., Jennings, N. R., and Kinny, D. (2000).

The Gaia methodology for agent-oriented analysis and design.

Autonomous Agents and Multi-Agent Systems, 3(3):285–312.



Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003).

Developing multiagent systems: The Gaia methodology.

ACM Transactions on Software Engineering and Methodology (TOSEM), 12(3):317–370.



Zambonelli, F. and Omicini, A. (2004).

Challenges and research directions in agent-oriented software engineering.

Autonomous Agents and Multi-Agent Systems, 9(3):253–283.

Special Issue: Challenges for Agent-Based Computing.



Bibliography XIII



Agent-Oriented Software Engineering

Multiagent Systems LM Sistemi Multiagente LM

Ambra Molesini & Andrea Omicini
{ambra.molesini, andrea.omicini}@unibo.it

Ingegneria Due
ALMA MATER STUDIORUM—Università di Bologna a Cesena

Academic Year 2010/2011

