Agent-Oriented Software Engineering

Multiagent Systems LS
Sistemi Multiagente LS

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

Ingegneria Due
ALMA MATER STUDIORUM—Università di Bologna a Cesena

Academic Year 2007/2008

Outline

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 developement after it has gone into use [Sommerville, 2007]

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 developement 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 enviroments
    - but this is not mandatory: we may use OO abstractions just because they are better, even for COBOL programming enviroments

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 physic 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

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

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

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]

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

- The use of meta-models to underpin object-oriented processes was pioneered in the mid-1990s by the OPEN Consortium [OPEN Working Group, 1999] leading to the current version of the OPEN Process Framework (OPF)

- The Object Management Group (OMG) then issued a request for proposals for what turned into the SPEM (Software Processing Engineering Metamodel) [SPEM, 2007]
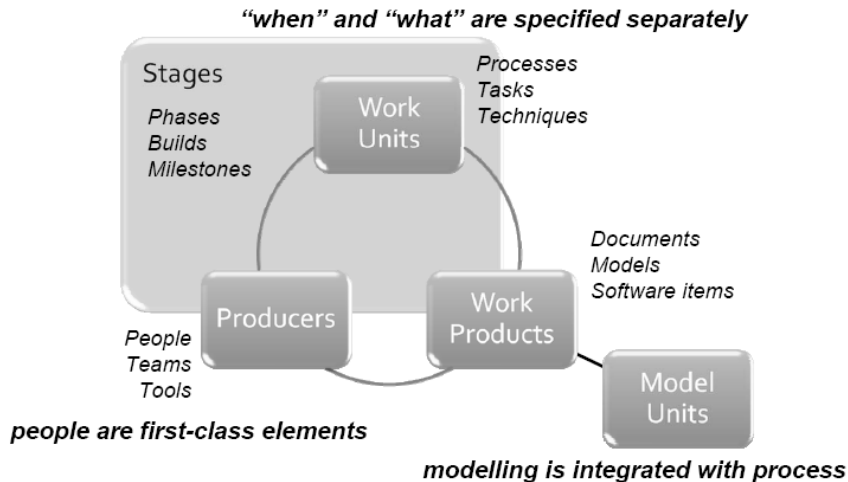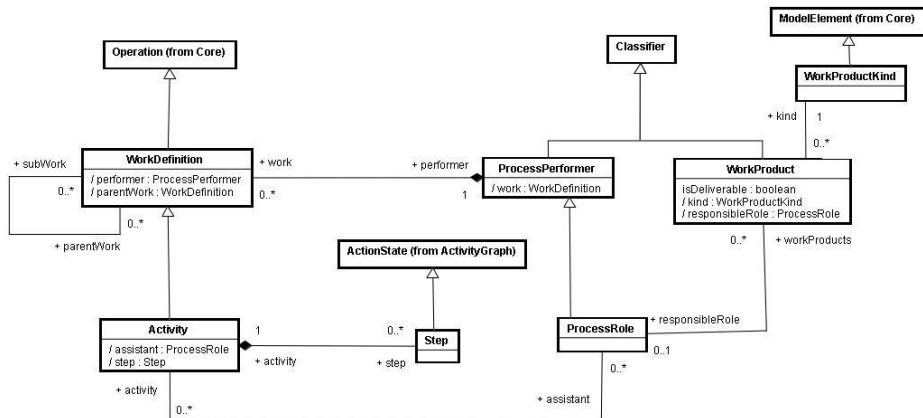
SPEM

- SPEM 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 Overview

Process Package [SPEM, 2007]

SPEM Elements

- A *WorkProduct* is anything produced, consumed, or modified by a process. It may be a piece of information, a document, a model, source code, and so on
- A *WorkProductKind* describes a category of work product, such as Text Document, UML Model, Executable, Code Library, and so on
- *WorkDefinition* is a kind of Operation that describes the work performed in the process

WorkDefinition SubClasses

- *Activity* – describes a piece of work performed by one ProcessRole. An Activity may consist of atomic elements called Steps
- Phase – is a specialization of WorkDefinition such that its precondition defines the phase entry criteria and its goal defines the phase exit criteria
- Iteration – An Iteration is a composite WorkDefinition with a minor phases
- Lifecycle – A process *Lifecycle* is defined as a sequence of Phases that achieve a specific goal. It defines the behavior of a complete process to be enacted in a given project or program

SPEM Elements

- A *ProcessPerformer* defines a performer for a set of WorkDefinitions in a process
- ProcessPerformer has a subclass, *ProcessRole*
- ProcessPerformer represents abstractly the whole process or one of its components, and is used to own WorkDefinitions that do not have a more specific owner
- *ProcessRole* defines responsibilities over specific WorkProducts, and defines the roles that perform and assist in specific activities
- *Guidance* provides more detailed information to practitioners about the associated ModelElement. For instance, Technique is a kind of Guidance. A Technique is a detailed, precise algorithm used to create a work product

SPEM's stereotypes [SPEM, 2007]

Stereotype	Notation
WorkProduct	
WorkDefinition	
Guidance	
Activity	
ProcessRole	
ProcessPackage	
Phase	
Process	
Document	
UMLModel	

Example
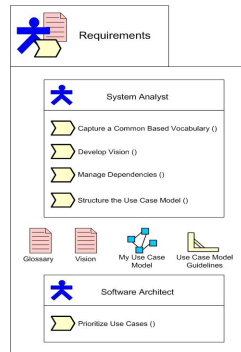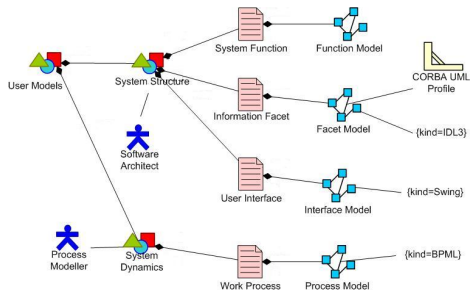
Example
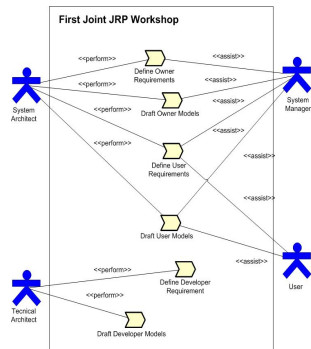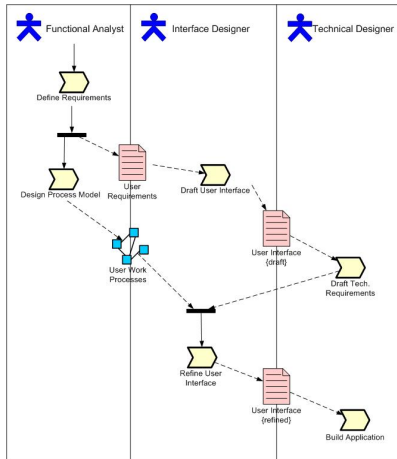
OPEN

- Object-oriented Process, Environment, and Notation (OPEN) [OPEN Working Group, 1999] 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]
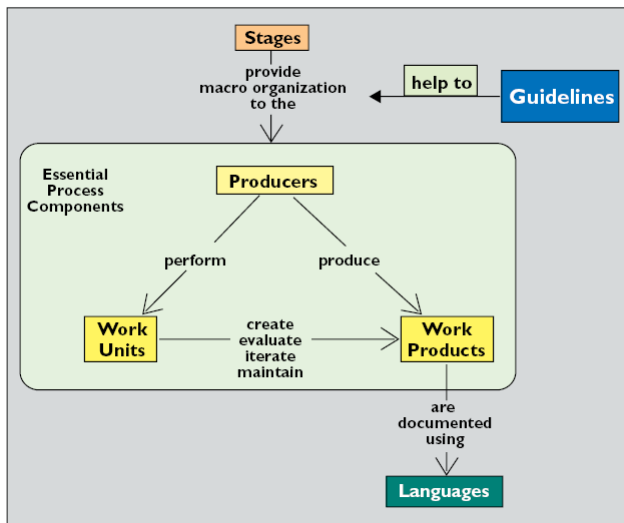
Work Product & Language & Producer

- 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

Methodologies

- As for software development, individual methodologies are often created with specific purposes in mind [Henderson-Sellers, 2005a]
  - particular domains
  - particular segments of the lifecycle
- Users often make the assumption that a methodology in 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 defitinions [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

- ... these definitions are different from the definitions we have given in the previous Section...

Method & Methodology
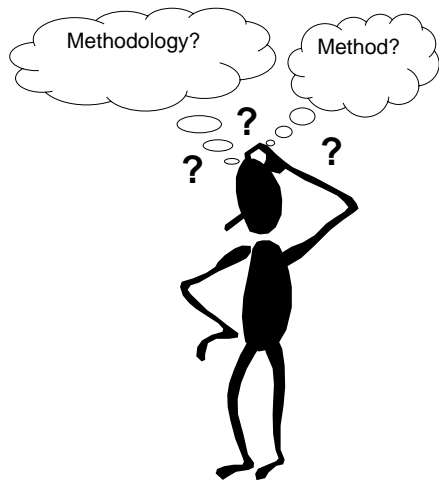
Method & Methodology
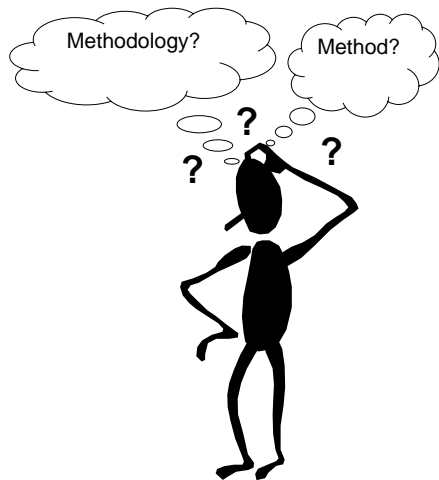
- All the concepts and ideas used in the Method Engineering are also applicable in our definitions of methodology and method

- Method Engineering tries to model methodological processes and products by isolating conceptual *method fragments*

- This fragments act as methodological "building blocks"

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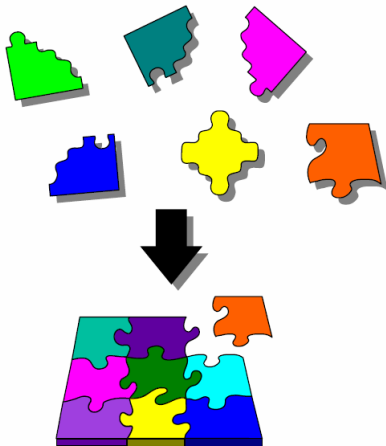
Method Engineering: Motivations
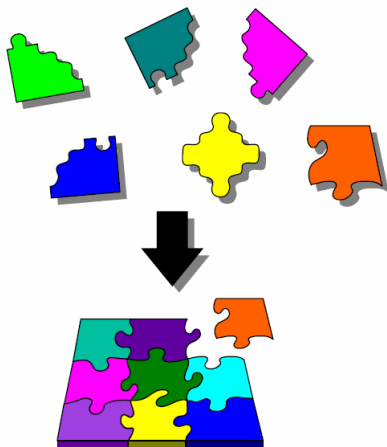
- Adaptability – to specific projects, companies, needs & new development settings
- Reuse – of best practices, theories & tools

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 guide-lines, 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]
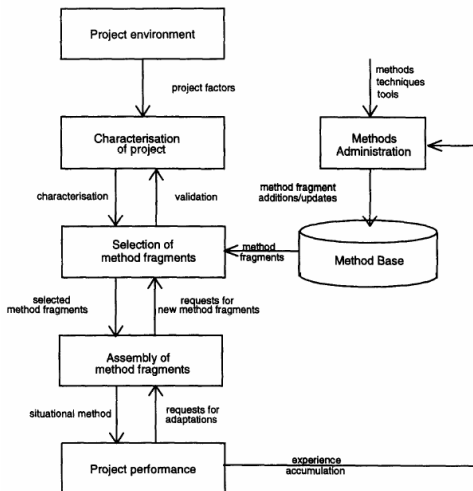
Situational Method Engineering I

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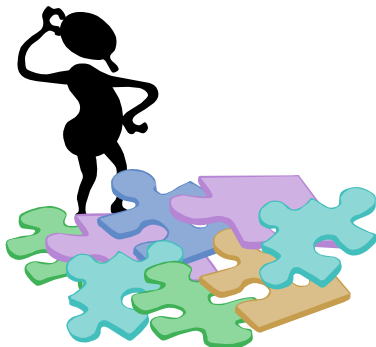
- Two important questions
    - How to represent method fragments?
    - How to assembly method fragments?
- To assemble method fragments into a meaningful method, we need a procedure and representation to model method fragments and impose some constraints or rules on method assembly processes

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]
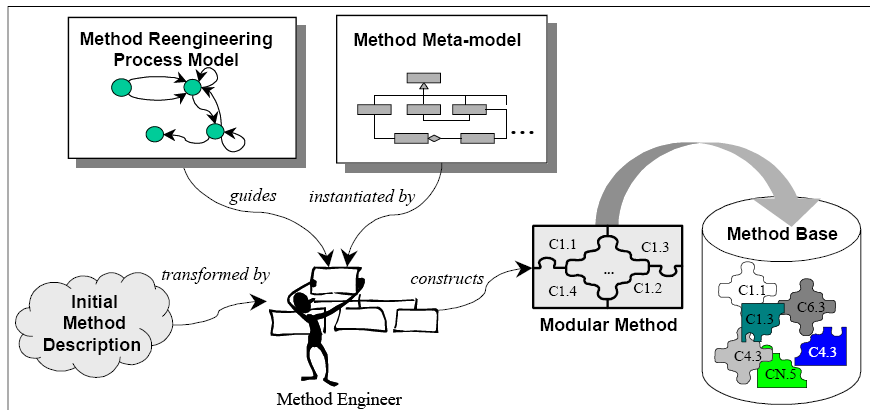
Method Reengineering

- 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]
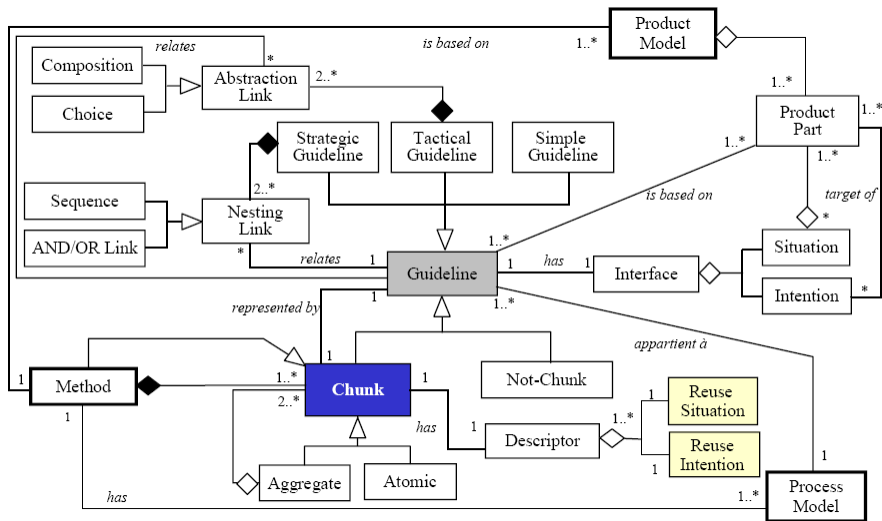
Method Meta-model

- 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

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]
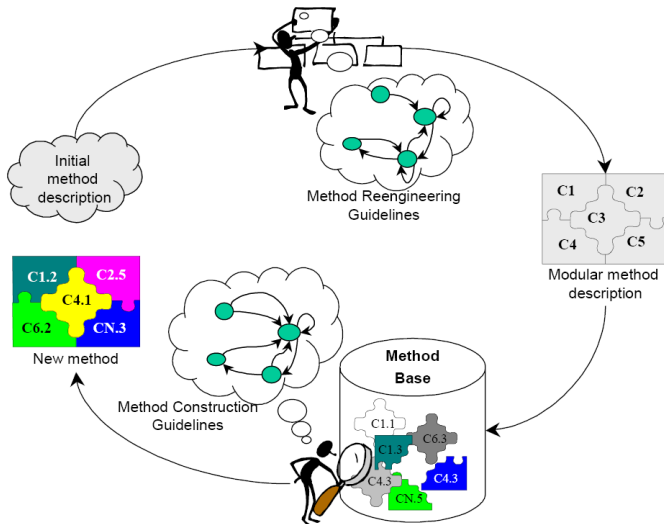
Assembly Map [Ralyté and Rolland, 2001b]
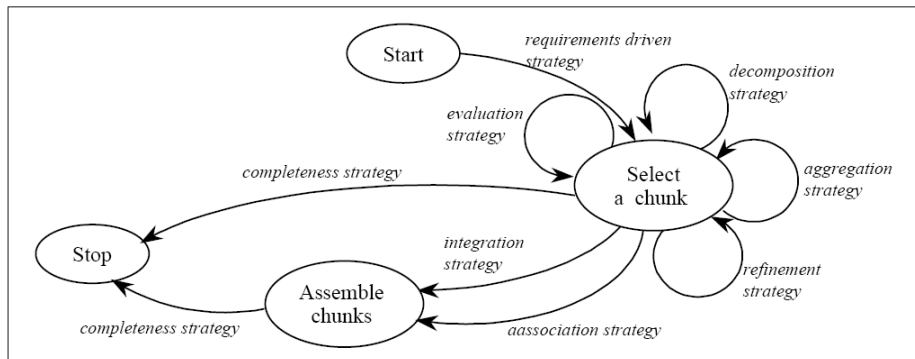
Integration Map [Ralyté and Rolland, 2001b]
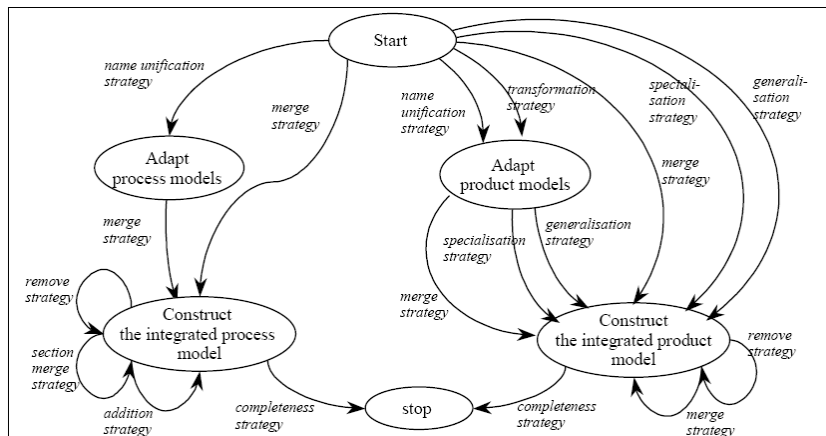
Association Map [Ralyté and Rolland, 2001b]
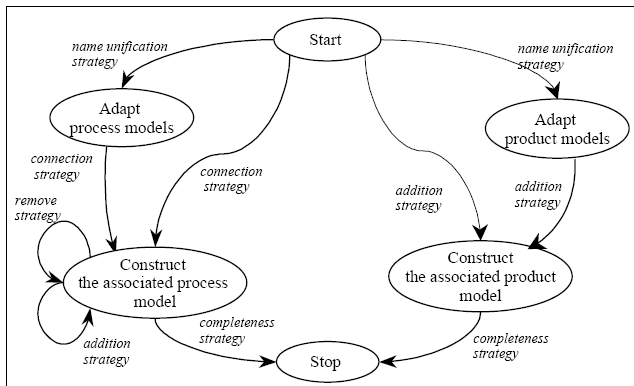
OPEN Process Framework [Henderson-Sellers, 2003]
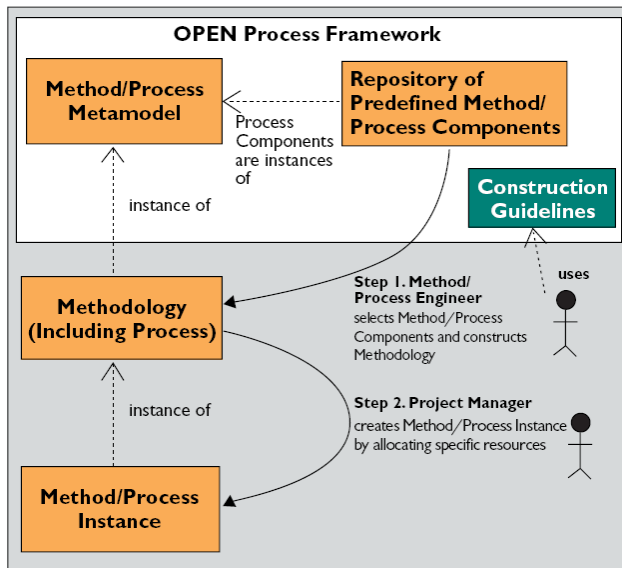
Usage Guidelines

- 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, 1999]
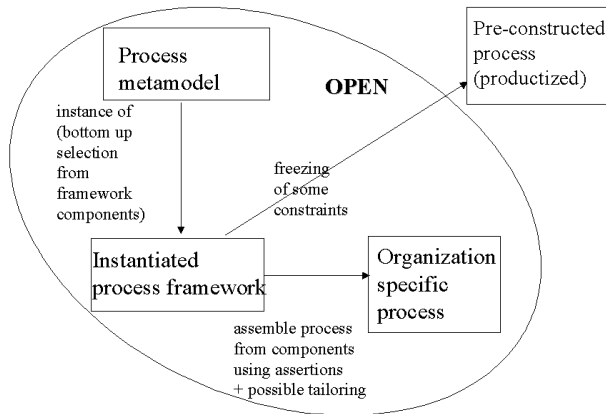
It is even possible to take sets of constraints to create "pre-tailored" styles of OPEN process

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]

	Activity					
Task	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

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

SE Implications of Agent Characteristics

- 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

SE Implications of Agent Characteristics

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

MAS Characterisation
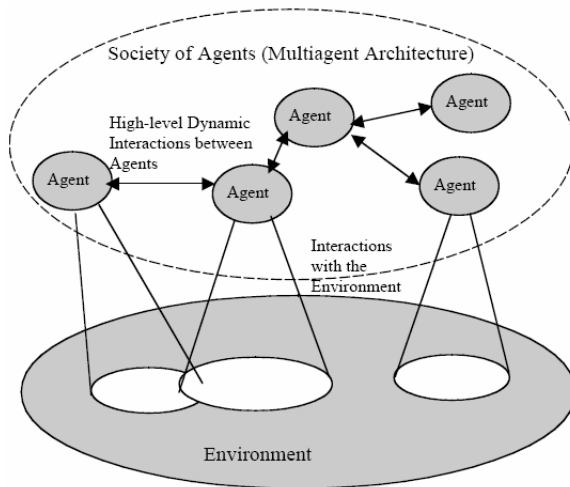
Agent-Oriented Abstractions

- 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:
  - The *local context* where an agent lives (e.g., a sub-organisation of agents) to handle mobility & opennes

Outline

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

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

Methodologies and technologies

- Today the engineer often works with technologies that do not support the abstractions used in the design of the systems
- For this reason 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 have mostly followed 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 have followed a *bottom-up approach*, evolving out of necessity from existing programming languages and development environments

Informatics Technologies Evolution

New abstractions

Programming Languages → Infrastructures → Software Engineering

Traditional

Agent-paradigm

Agent abstractions

Software Engineering → Infrastructures → Programming Languages
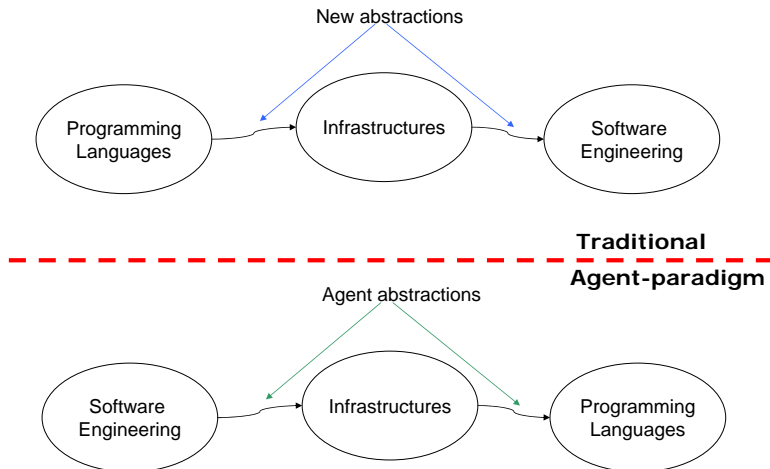
The Gap

- The gap between methodologies and infrastructures and languages can leads 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,]:
    - 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. . .
    - FIPA Method Engineering
    - OPEN (in short)

Outline

Method Engineering

- 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., 2007]

- 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

- 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

Method Engineering
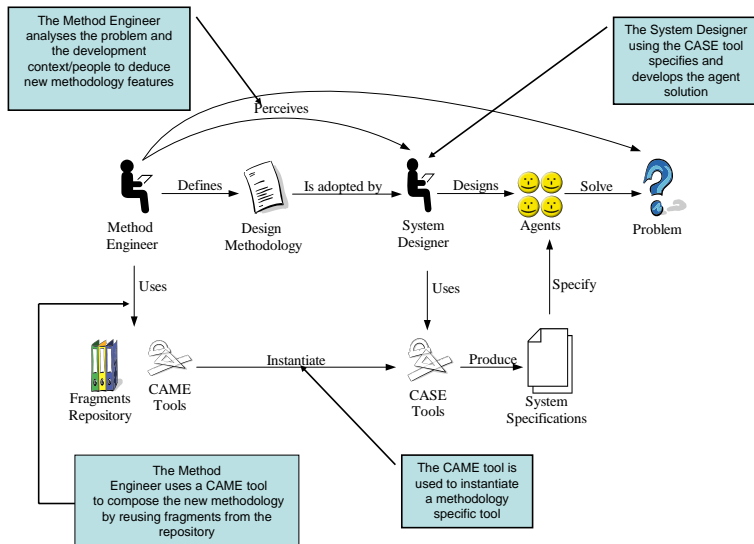
FIPA Methodology Working Group

- This approach has been adopted, in the past few years, by the FIPA Methodology Technical Committee (TC) (FIPA – Foundation for Intelligent Physical Agents)[Methodology Working Group,]
- FIPA had recently moved to the IEEE Computer Society under the name of IEEE FIPA Standards Committee and with this occurrence the activities of the Methodology TC were stopped
- The FIPA Methodology TC was constituted in 2003 with the aim of capitalising on the efforts of many researchers in the area of MAS design and contributing to the reuse of parts of existing methodologies (and the related knowledge), through an appropriate set of specifications

FIPA TC goals I

- Definition of the method fragments meta-model — it is necessary to formally represent method fragments in order to facilitate their identification, representation, integration and storage in the method base.

- Identification of the method base architecture — this is the method base needs of a technological infrastructure for the instantiation of the previously defined method fragment meta-model.

- Collection of method fragments — they can originate from the most diffused methodologies and other specific contributions. After formalisation, they can be introduced into the method base.

FIPA TC goals II

- Description of techniques for methods integration — it is necessary to define guidelines for methods integration in order to both construct the new methodology (by retrieving the method fragments from the method base and integrating them) and apply it to the real design work.

- A more ambitious goal was enabling the use of automating tools.

Tools: CAPE

- Computer-Aided Process Engineering (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.

Tools: CAME

- Computer-Aided Method Engineering (CAME) tools that could offer specific support for the composition/maintenance of a method fragment; these tools should enable the designer to define a method fragment according to the definition, provided by the FIPA Methodology TC, and the prescriptions coming from the method base. Besides, they would allow the modification of these fragments when assembling needs or other customisation requests emerge.

Tools: CASE

- Computer-Aided Software Engineering (CASE) tools that assist the designer in performing the development process based on the composed methodology. These tools should be the evolution of existing CASE instruments, since they enforce the execution of the design phases in the order defined at the time of methodology composition (according to the adopted process life-cycle and they guide the designer in profitably applying it.

Agent-Oriented Method Engineering Process
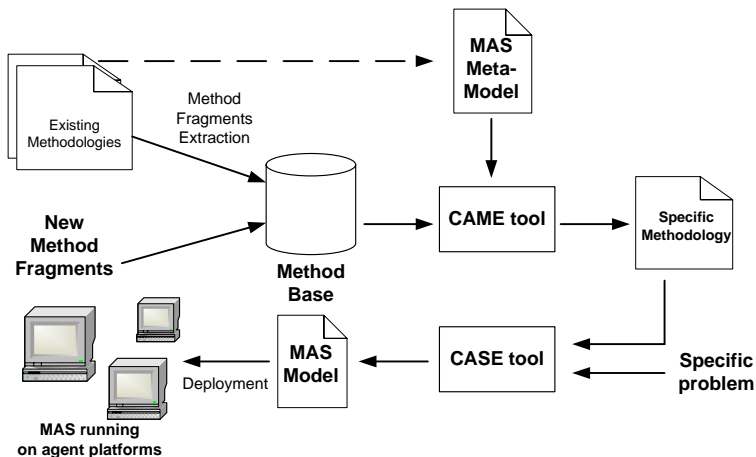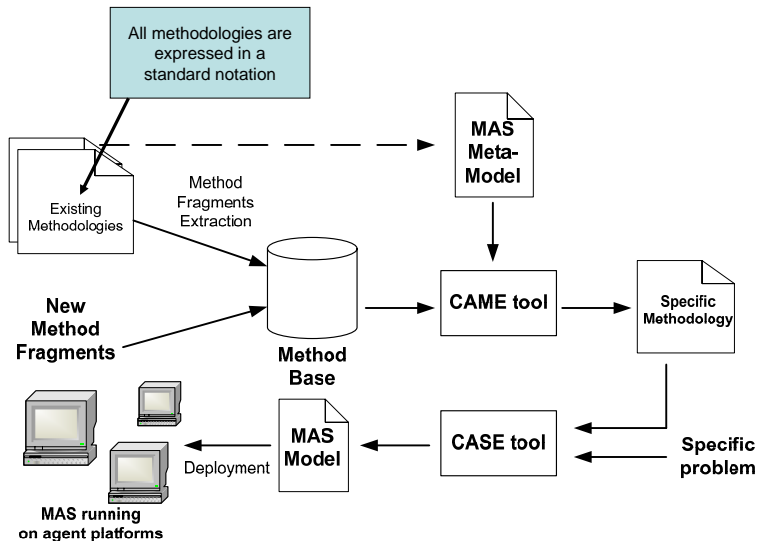
Agent-Oriented Method Engineering Process

Agent-Oriented Method Engineering Process
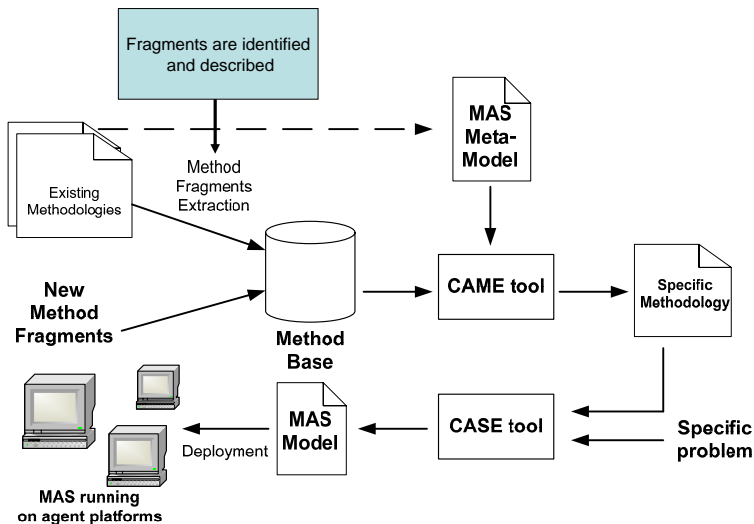
Agent-Oriented Method Engineering Process
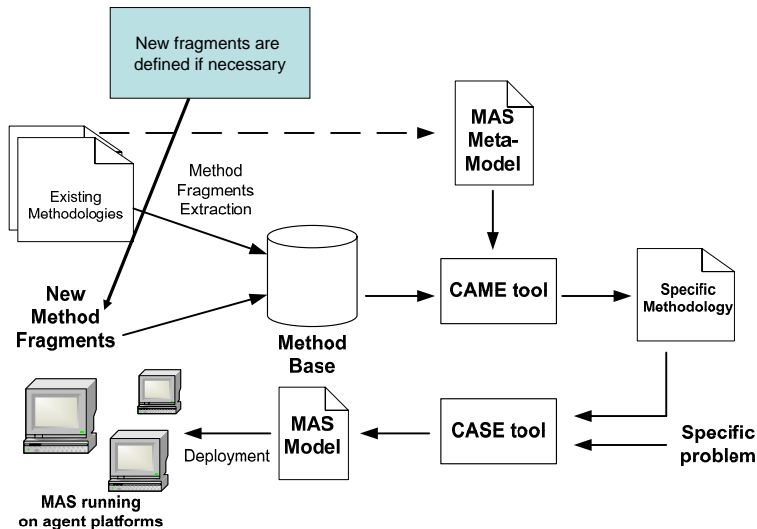
Agent-Oriented Method Engineering Process
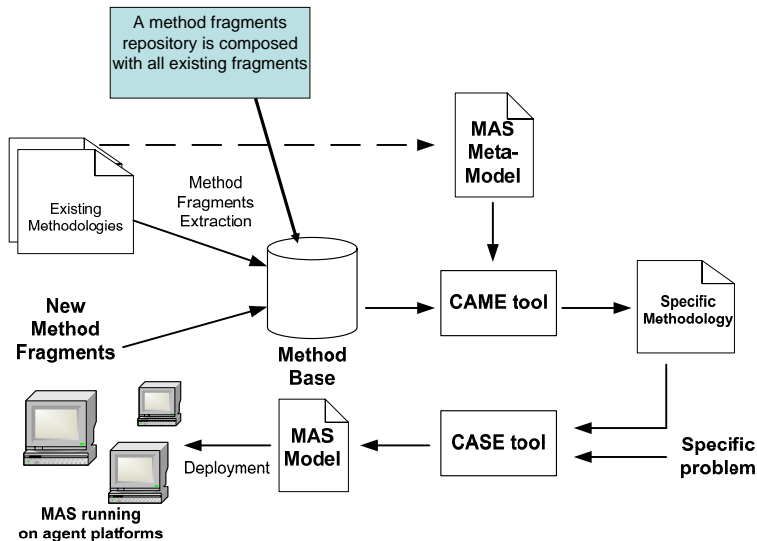
Agent-Oriented Method Engineering Process
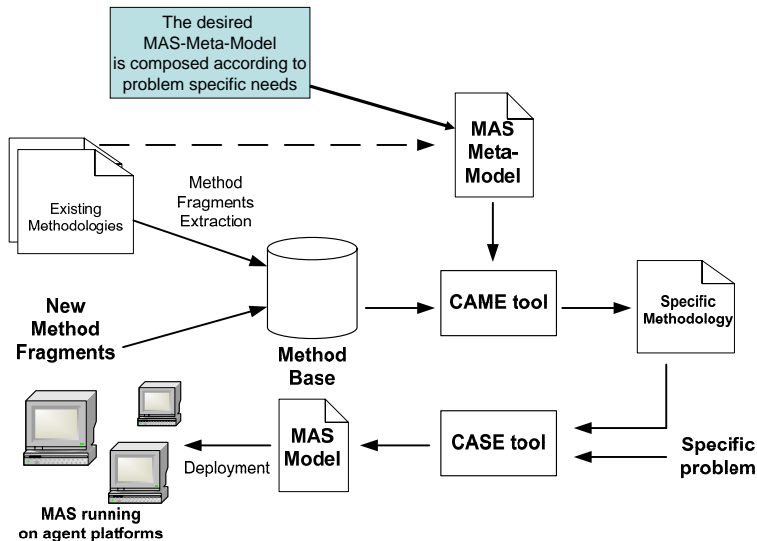
Agent-Oriented Method Engineering Process
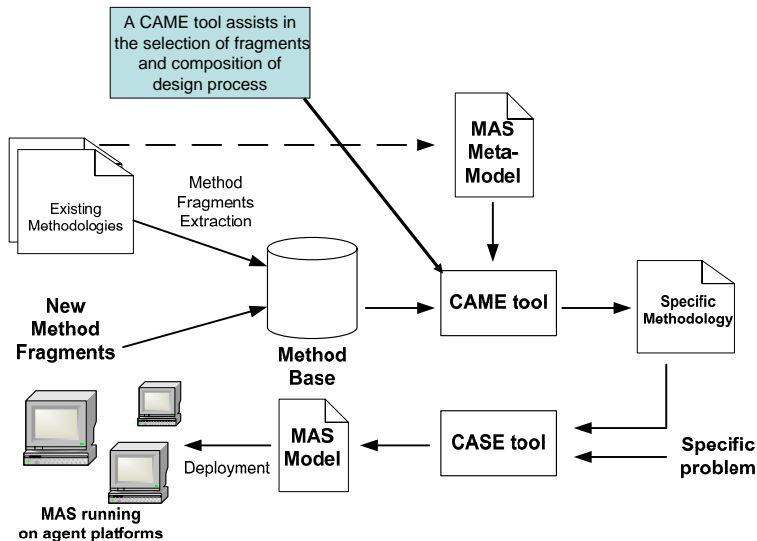
Agent-Oriented Method Engineering Process
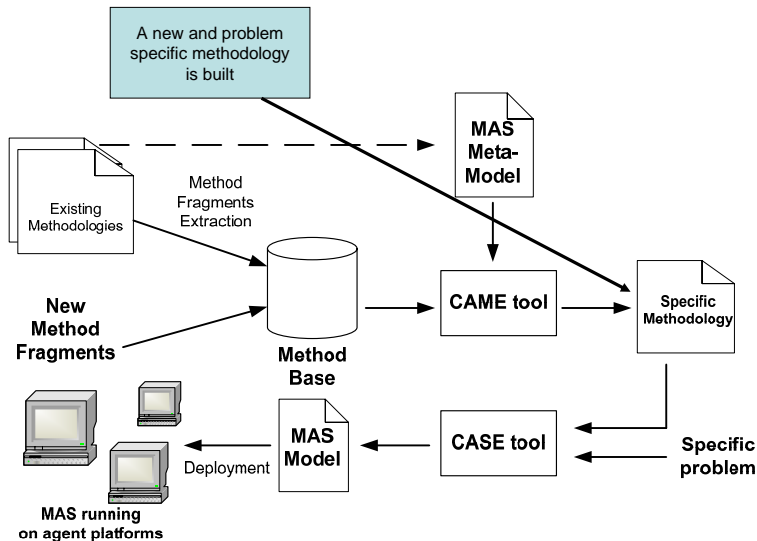
Agent-Oriented Method Engineering Process
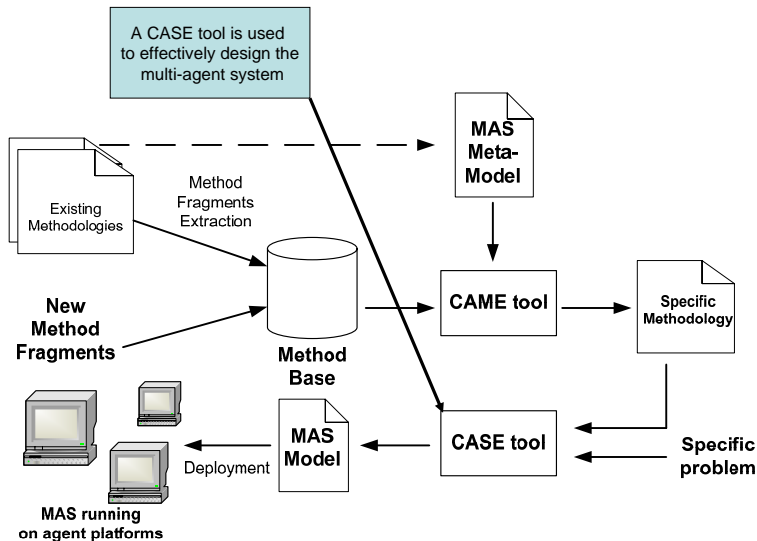
Agent-Oriented Method Engineering Process
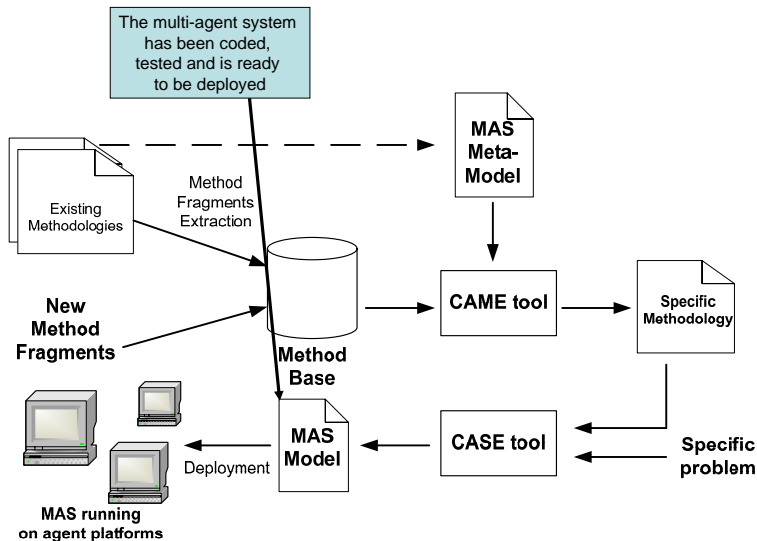
What is a Method Fragment?

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

- A portion of process
- One or more deliverables
- Some preconditions
- A list of concepts of MAS Meta-model
- Guidelines
- A glossary of terms
- Composition guidelines
- Aspects of the fragment
- Dependency relationships

Portion of Process

- A specification of the portion of the process, which defines what is to be done by the involved stakeholder(s) and in what order
- The fragment specification prescribes the use of SPEM for describing its procedural aspect
- According to SPEM, the FIPA fragment can be regarded as a process component

Deliverables

- One or more deliverables such as AUML/UML diagrams and text documents
- These should be part of the fragment specification in the form of a description of their structure (in order to clarify what is the expected output of the presented activities)
- Also a reference to the suggested (or adopted, in the original methodology from which this fragment has been extracted) modelling notation

Preconditions

- Preconditions represent a kind of constraint, specifying when it is possible to fire the activities specified in the fragment
- They are usually related to the required input data
- these preconditions can be thought of as similar to the preconditions in a contract between two classes
- In particular, the preceding fragment (or the n preceding fragments) is (are) responsible for establishing the conditions that will enable the successful execution of the following fragment
- The formalisation of these preconditions would allow the introduction of some kind of automatic assistance in the composition of the fragments, but a formal language has not been specified or adopted yet and the only considerations that can be easily automated according to this specification, concerns the required input set in terms of already-defined MAS meta-model components

Concepts of MAS Meta-model

- A list of components of the MAS meta-model to be defined or refined through the specified process (they belong to the MAS meta-model adopted by the methodology from which the fragment was extracted)

- This list could be void (this is, for instance, the case of a fragment whose purpose consists in selecting between two different paths in the design process according to the evaluation of some aspects of the actual design)

- However, all the fragments that have been identified up to now are concerned with some components to be defined/refined, thus showing that the community is, even now, still more concerned about a product-oriented identification of fragments than a process-oriented one

Guidelines

- Application guidelines that illustrate how to apply the fragment and the related best practices
- The same formalisation of these guidelines in the existing agent-oriented methodologies has its own specific importance, since otherwise, except for a few well-documented approaches, guidelines often remain bound to the personal knowledge of some skilled designers or the methodology creators

Glossary

- A glossary of terms used in the fragment
- This prevents misunderstandings if the fragment is reused in a context that is different from the original one
- In order to facilitate this part of the fragment documentation, the members of the TC discussed a list of definitions for many commonly used terms.

Composition guidelines & Aspects of the fragment

- Composition guidelines which describe the context/problem addressed by the specific fragment and that are behind the methodology from which they have been extracted

- Aspects of the fragment are textual descriptions of specific issues, such as the platform to be used for system implementation and application area; they help in delimiting the proper application field for the fragment

Dependency Relationships

- Dependency relationships are useful for assembling fragments
- When the fragments' granularity is fine grained (and the FIPA repository was conceived to allow the introduction of different-sized fragments), it is common to reuse more fragments from a specific methodology since their adoption probably corresponds to adopting some philosophy for the composition of a specific portion of the software engineering process

Method Fragments Integration

- *Method fragments integration* is the process of composition of the new software engineering process [Cossentino et al., 2007]
- Usually consists of two different and complementary phases:
    - the selection of the reused fragments from the method base and
    - their assembly, including the modification of fragments when necessary
- Several approaches exist in the literature to deal with these crucial phases, the FIPA Methodology TC members discussed this topic and mainly studied two basic approaches for the integration of methods during the construction of the agent-oriented software engineering process:
    - meta-model driven approach
    - development-process driven approach

Meta-model Driven Approach

- This is based on the MAS meta-model adopted by the designer for the development of a MAS for a specific problem in a specific application domain
- To build a software engineering process by exploiting the meta-model-driven approach, the designer has to:
    - choose or define the MAS meta-model suitable for the specific problem and/or the specific application domain
    - choose the method fragments that are able to produce the identified meta-model elements
    - define a development process characterised by a method fragments-execution order on the basis of the relationship existing among the meta-model elements produced by each fragment
- The obtained software engineering process is able to completely ensure the MAS meta-model instantiation for the given problem in a specific application domain.

Development-process Driven Approach

- this is based on the instantiation of a software development process in which each phase is carried out using appropriate method fragments selected on the basis of the supported activities and of the resulting work products
- To build a software engineering process by exploiting the development process-driven approach, the designer must:
    - choose or define a software engineering process life-cycle suitable for the specific problem and for the specific application domain
    - instantiate the development process by selecting, for each phase of the life-cycle, some suitable method fragments, chosen from the method base or even defined *ad hoc*
- The work products produced in a given phase might constitute the input for the subsequent phase, provided that they contain all the information required for initialising it.

Comparison: Meta-model-driven Approach

- The meta-model-driven approach
    - provides flexibility for the definition of many aspects of the MAS to be developed; this is probably the most suitable one if social rules coming from a specific domain play a relevant role in the problem to be solved
    - conversely, it is characterised by a difficulty in integrating different fragments, owing to the different semantics of the concepts they can represent in the meta-models subsumed by the methodologies from which they have been extracted
    - furthermore, the a *priori* selection and/or definition of the meta-model to adopt for the specific problem and/or application domain is a difficult and at the same time crucial task

Comparison: Development process-driven Approach

- The development process-driven approach
    - is characterised by the following advantage: flexibility for the construction of a software engineering process by means of the instantiation of each stage of the selected process life-cycle
    - On the other hand, the disadvantages are the following:
        - low flexibility of the MAS meta-model, since it results from the sum of elements defined by the selected method fragments
        - adaptation among the work products, which is sometimes difficult to achieve
        - having to choose and define the process life-cycle to instantiate for the specific problem and/or application context
        - low level of help in selecting the fragments that descend from the process life-cycle choice

Comparison of The Approaches

- Each one of the above-listed points represents an open problem and a challenge for the agent community

- the first one to be explored consists in some peculiarities that are related to the agent paradigm, the most important probably being the role that the agent social organisation plays in the composition of the new process

- The proposed approaches to the integration of method fragments are not mutually exclusive: hybrid approaches containing features of both of them might be defined as well

- An example of a process composition that mixed the two proposed approaches has been used to create one of the first agile processes for MAS design, PASSI Agile

A Refinement of the Method Fragment Proposal

- In a recent work [Cossentino et al., 2007] a refinement of the proposal was presented

- The authors consider the process as the set of steps to be performed in order to produce an output, the way of performing some activities, and the resources and constraints this requires

- it is now well recognised that a standard process does not exist, so each process is specific for a particular development context, which relates to resources, people and competence aspects, and for a problem type — it can in fact solve a specific problem or a family of related problems; these two elements constitute a precise indication of the requirements of the process

Software Engineering Process
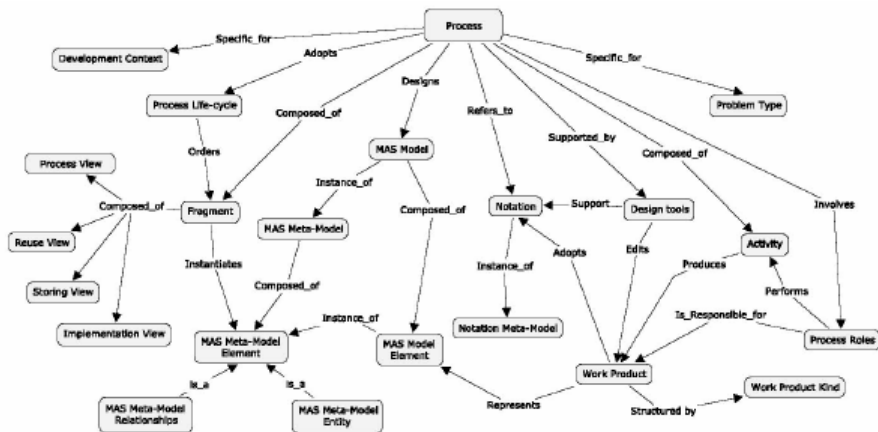
Process in Agent-Oriented Contex

- A process in the agent-oriented context aims at designing a MAS model whose elements (MAS model elements) are represented in the work products

- A MAS model is obviously an instance of a MAS meta-model that gives a structural representation, in terms of elements and relationships, of the concepts belonging to the system under construction

- The fragment is such a complex and fundamental element of the method engineering approach that it should be explored from several different points of view in order to achieve the deepest comprehension of its implications during design time

- More specifically, four different views are identified: process, storing, reuse and implementation

Fragment-Process View I
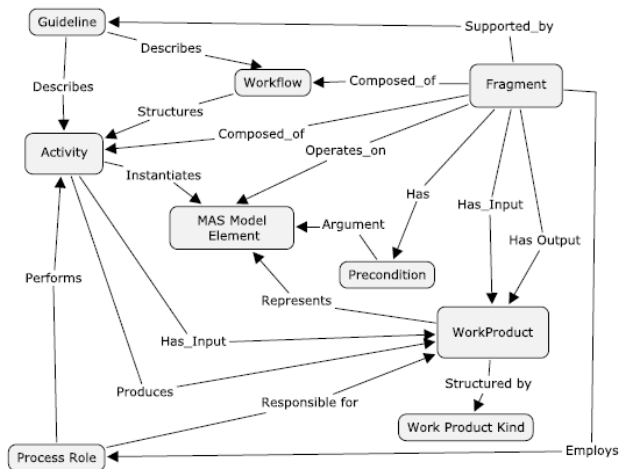
Fragment-Process View II

- The fragment-process view is aimed at representing the process-related aspects of the fragment

- The most important elements are workflow, activities and work products

- The workflow structures the activities and it is described using activity diagrams

- An activity has a work product as an input and produces other work products

- Each work product could be a graphical work product or a textual work product (free-text document or structured document)

Fragment-Reuse View

- This view is concerned with the reuse features of the fragment and lists the elements that could be helpful in reusing the fragment in the composition of a new software engineering process.
- The elements of the fragment meta-model that belong to this view are:
  - MAS Meta-model Element — this defines the scope of the fragment, the elements that it will instantiate in the produced work products
  - Aspect, Glossary, Composition Guideline, Fragment Dependency -– these have the meanings given by the FIPA Methodology TC

Fragment-Storing View I

- This view concerns the storage of the fragment in the method base and its retrieval.
- This view includes the following elements

    Phase — a specialisation of Work Definition that is usually built up from several finer activities. The need for phases is evident if we think that a fragment conceived for use in the early stages of the design process is unlikely to be useful in later phases such as coding or testing

    Process Role — it would make no sense in some specific development context to select fragments employing process roles not available to the intended developing team of the new software engineering process

Fragment-Storing View II

MAS Meta-Model Element — this is one of the central points of our
approach and appears in this view in order to support
the construction of a new SEP starting from the initial
definition of its MAS meta-model. As a consequence,
the method engineer can select all the fragments that
deal with the elements of this meta-model, thus
drastically reducing the dimension of the fragment set
he/she has to choose from

Fragment-Implementation View I
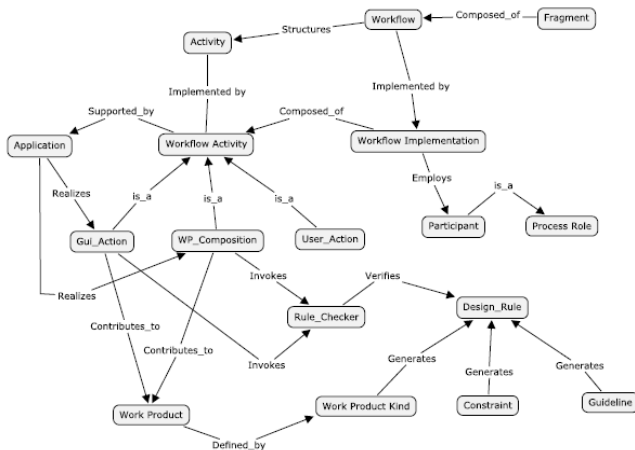
Fragment-Implementation View II

- This view strictly concerns the implementation of the main elements we explained in the process view: workflow, activity and work product
- The Workflow is implemented by a Workflow Implementation
- Each Activity is implemented by a Workflow Activity that corresponds to a real piece of work
- each work product is defined by a work product kind that generates a set of design rules depending on the kind itself, on some specific constraints and a set of guidelines

The OPEN Approach

- OPEN was originally developed for systems development in an object-oriented context

- However extending OPEN to support agent-oriented software development is relatively straightforward: it requires the identification of any new Tasks, Techniques, WorkProducts, Producers. . . [Henderson-Sellers, 2005a]

- In recent years a number of specific method fragments have been created for the use with the OPF meta-model and the relative repository [Henderson-Sellers, 2005b]

- These fragments have been derived from an analysis of a large number of stand-alone agent-oriented methodologies

The OPEN Approach

- Over the last two years, studies have been undertaken of what method fragments are needed to fully support agent-oriented software engineering methodologies
- These were added to a OPF repository that was originally not agent-oriented
- OPF repository was augmented of method fragments by those derived from a large number of stand-alone agent-oriented methodologies:
  - MaSE, Prometheus, Gaia, Cassiopeia, Agent Factory, MAS-Common-KADS, Tropos, PASSI and CAMLE
- Each of these fragments corresponds to one of the classes in the OPF meta-model
- The construction of new AO methodology is the same already illustrated in the context of traditional Method Engineering

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

Bibliography I

📄 Bernon, C., Cossentino, M., Gleizes, M. P., Turci, P., and Zambonelli, F. (2004).
A study of some multi-agent meta-models.
In Odell, J., Giorgini, P., and Müller, J. P., editors, *AOSE*, volume 3382 of *Lecture Notes in Computer Science*, pages 62–77. Springer.

📄 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.
Inf. Syst., 24(3):209–228.

Bibliography II

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

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

📄 Fuggetta, A. (2000).
Software process: a roadmap.
In *ICSE '00: Proceedings of the Conference on The Future of
Software Engineering*, pages 25–34, New York, NY, USA. ACM Press.

Bibliography III

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

📄 Henderson-Sellers, B. (2003).
Method engineering for oo systems development.
Commun. ACM, 46(10):73–78.

📄 Henderson-Sellers, B. (2005a).
Creating a comprensive agent-oriented methodology: Using method engineering and the OPEN metamodel.
In Henderson-Sellers, B. and Giorgini, P., editors, *Agent Oriented Methodologies*, chapter XIII, pages 236–397. Idea Group Publishing, Hershey, PA, USA.

Bibliography IV

📄 Henderson-Sellers, B. (2005b).
Evaluating the feasibility of method engineering for the creation of
agent-oriented methodologies.
In Pechoucek, M., Petta, P., and Varga, L. Z., editors, *Multi-Agent
Systems and Applications IV, 4th International Central and Eastern
European Conference on Multi-Agent Systems, CEEMAS 2005,
Budapest, Hungary, September 15-17, 2005, Proceedings*, volume
3690 of *Lecture Notes in Computer Science*, pages 142–152. Springer.

📄 MEnSA Project.
Methodologies for the engineering of complex software systems:
agent-based approach.
http://www.mensa-project.org/news.php.

Bibliography V

📄 Methodology Working Group.
Ieee-fipa methodology working group home page.
http://www.fipa.org/activities/methodology.html.

📄 OPEN Working Group (1999).
Open home page.
http://www.open.org.au/index.html.

📄 Ralyté, J. and Rolland, C. (2001a).
An approach for method reengineering.
In Kunii, H. S., Jajodia, S., and Sølvberg, A., editors, *Conceptual Modeling*, volume 2224 of *Lecture Notes on Computer Science*, pages 471–484. Springer.
20th International Conference (ER 2001), Yokohama, Japan, 27-30 November 2001. Proceedings.

Bibliography VI

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 *Lecture Notes on Computer Science*, pages 267–283. Springer.
13th International Conference (CAiSE 2001), Interlaken, Switzerland, 4-8 June 2001. Proceedings.

Sommerville, I. (2007).
Software Engineering 8th Edition.
Addison-Wesley.

SPEM (2007).
SPEM Software Process Engineering Meta-Model home page.
http://www.omg.org/technology/documents/formal/spem.htm.

Agent-Oriented Software Engineering

Multiagent Systems LS
Sistemi Multiagente LS

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

Ingegneria Due
ALMA MATER STUDIORUM—Università di Bologna a Cesena

Academic Year 2007/2008