UNIVERSIDAD POLITECNICA DE VALENCIA

PROS
Centro de Investigación en Métodos de Producción de Software

Technical Report




A State-of-the-Art Review on Method Engineering

Mario Cervera, Manoli Albert, Victoria Torres, Vicente Pelechano





Ref. #:	PROS-TR-2011-13				
Title:	A State-of-the-Art Review on Method Engineering				
Author (s):	Mario Cervera, Manoli Albert, Victoria Torres, Vicente Pelechano				
Corresponding author (s):	Mario Cervera, mcervera@pros.upv.es Manoli Albert, malbert@pros.upv.es Victoria Torres, vtorres@pros.upv.es Vicente Pelechano, pele@pros.upv.es				
Document version number:	1.0	Final version:	Yes	Pages:	34
Release date:	May 2011				
Key words:	CAME environment, metaCASE, Method Engineering Language				

Chapter 2
State of the Art

The term *Method Engineering* was first introduced in the mid-eighties by Bergstra et al in [Bergstra85], and was later used in other works such as [Kumar92], [Slooten93] and [Brinkkemper96]. From that moment, Method Engineering emerged as a promising way to tackle the adaptation of software production methods and tools to specific project needs.

Since the origin of Method Engineering two decades ago, this discipline has had an extensive history. Many works developed both at academia and industry have contributed to establish a solid theoretical basis in this field. In order to underpin this theory, a survey of the most relevant contributions is gathered in [HendersonSellers10].

Specifically, this chapter analyzes some of the most important Method Engineering proposals, addressing three topics: (1) Method Engineering approaches, (2) languages for building method specifications and (3) software tools supporting these approaches and languages. According to these topics, section 2.1 first presents different approaches for method definition. Then, in section 2.2, some of the most significant languages that have been proposed in the literature to perform this definition are described. Section 2.3 surveys some tools that have been developed to support the approaches and languages previously presented and, finally, section 2.4 draws some conclusions.

2.1. Method Engineering Approaches

Many Method Engineering approaches of different nature have been proposed during the last two decades. For instance, approaches such as [Brinkkemper99] or [Prakash97] tackle method construction as an assembly of method components. Others, however, focus on the spreading and sharing

of methodological knowledge rather than the definition and adaptation of methods. This is the case of the community based approach proposed in [Mirbel07], which aims at solving method usage problems by improving the practitioners' understanding of the method to apply. Furthermore, proposals such as [Deneckère98] propose the use of patterns for performing method extensions while others such as [Guzélian07] and [Iacovelli08] offer a service-oriented view of Method Engineering.

In view of this disparate scenario, this section aims to provide a survey of the most extended types of approach. In particular, these types have been classified according to the types proposed in [Ralyté03], which are: (1) the assembly-based approach, (2) the paradigm-based approach and (3) the extension-based approach. In order to illustrate the steps that must be followed to perform each of these approaches, the *Map* process meta-model proposed in [Rolland99] is used. This meta-model enables the creation of intuitive process models based on the notions of *intentions* to fulfill and *strategies* to achieve these intentions.

2.1.1. The assembly-based approach

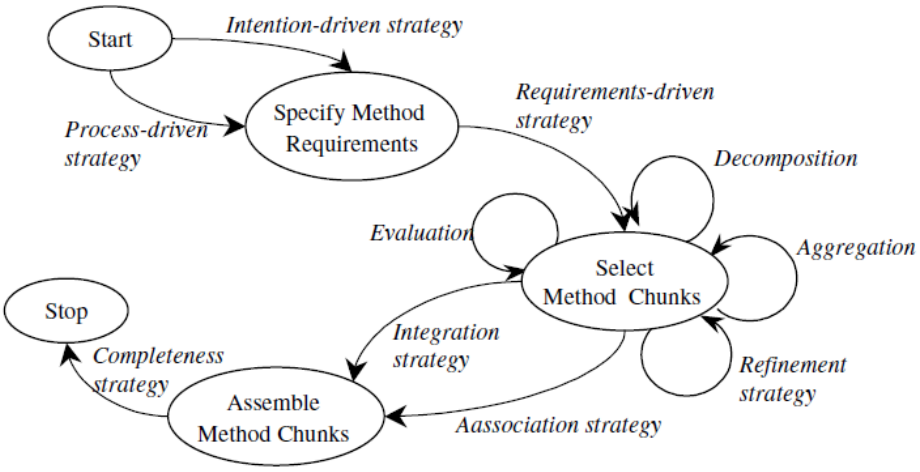

Fig. 2.1. Assembly-based approach (from [Ralyté03])

The assembly-based approach [Ralyté01, Ralyté03] consists in the construction of software production methods by means of the assembly of reusable method chunks (or fragments) that are stored in some method base

repository [Brinkkemper98, Harmsen97, Ralyté99]. Thus, methods are viewed as a collection of chunks that are "glued together" to form a method attuned to specific context needs. In order to show the different steps that must be followed to perform the method assembly, Fig. 2.1 shows this approach as a process model following the map notation.

The assembly-based approach is the most common of the three approaches. This is mainly due to the fact that this approach advocates for a modular vision of methods, which entails important advantages. Between these advantages, reusability is of high significance. Specifically, a modular vision of methods facilitates the reusability of their different parts, which directly leads to a reduction of the time required to define new methods. Furthermore, considering a method as an assembly of components also has a positive impact on its evolution qualities, such as maintainability and extensibility.

Some examples of relevant Method Engineering proposals that follow this approach are Brinkkemper's [Brinkkemper96, Brinkkemper98, Brinkkemper99] and Prakash's [Prakash97]. On the one hand, Brinkkemper mainly focuses on method fragment assembly techniques and their formalization by means of first-order logical formulas. In [Brinkkemper99] he stresses the need of imposing constraints in the assembly process in order to obtain meaningful methods. Most of the constraints that he proposes are syntactical, but he emphasizes the need of defining semantical constraints as well, which requires the formalization of the fragment semantics. He carries out this formalization by means of an ontology.

On the other hand, Prakash proposes an approach to formal method specification. This approach is based on three levels: the generic view (the most abstract view of a method, independent of the underlying paradigm), the meta-model and the method (obtained by instantiating the meta-model). These three layers represent an attempt to develop a comprehensive framework and architecture for methodology domain modeling. Specifically, in this approach a method is viewed as a collection of *method blocks*, which are defined as pairs <objective, approach>. The objective of a method block establishes what the block tries to achieve and the approach defines the technique that can be used to achieve the objective of the block. In addition, he also proposes different types of blocks, such as *product manipulation* and *constraint*

enforcement (for atomic methods), and *product composition* and *compositional-constraint enforcement* (for compound methods).

The nomenclature problem

In the Method Engineering literature, proposals that follow the assembly-based approach denote the atomic element from which methods can be assembled in different ways. For instance, Ralyté uses the term method chunk, while Prakash uses the term method block and Brinkkemper the term method fragment. In order to reach a consensus on the definition of this atomic element, in [HendersonSellers08a] a study of the different terms that have been proposed is presented. In general, the most accepted are *method fragment* and *method chunk*.

On the one hand, method fragments can be either *product fragments* or *process fragments* [Rolland96a, Brinkkemper96, Punter96, Harmsen97]. In general, a product fragment describes a product that is either consumed or produced during the method. A process fragment describes activities and procedures that must be executed to construct products. On the other hand, method chunks [Ralyté99, Ralyté01, Mirbel06] can be defined as the combination of a process fragment and a product fragment.

During the last decade, there has been much debate about the efficacy of a method chunk as compared to a method fragment. While method chunks offer some advantages, it seems that method fragments are quite more flexible. For instance, one advantage of method chunks is argued to be the speed of usage, since a smaller number of chunks is usually required to assemble a complete method. However, there is a potential disadvantage as a result of the fact that the process-product linkage present in method chunks is neither one-to-one nor unique in real-life scenarios. Thus, the separation between product and process that method fragments provide implies important advantages such as the possibility to relate one process fragment with many product fragments and the possibility to reuse one product fragment in the definition of many process fragments [HendersonSellers08a].

Specifically, in the proposal presented in this thesis the concept of method fragment is used. One of the reasons for this is the language used in the proposal, i.e. the SPEM standard (see section 2.2.6). In particular, the separation of product and process fragments allows method engineers to

leverage the clear separation between method product and process provided by SPEM.

2.1.2. The paradigm-based approach

The paradigm-based approach [Ralyté03, Ralyté05] is based on some initial idea expressed as a model or a metamodel that is called the *paradigm model* and supports the evolution of this paradigm model into a new model satisfying another engineering objective. In other words, the hypothesis of this approach is that a new method is obtained either by abstracting from an existing model or by instantiating a meta-model. Thereby, this approach uses meta-modeling as its underlying Method Engineering technique.

One of the results obtained by the meta-modeling community is the definition of any method as composed of a product model and a process model [Prakash99]. A product model defines a set of concepts, their properties and relationships that are needed to express the outcome of a process. A process model comprises a set of goals, activities and guidelines to support the process goal achievement and the action execution. Therefore, method construction following the meta-modeling technique is centered on the definition of these two models [Ralyté05]. This is illustrated in Fig. 2.2, wherein a map representing the paradigm-based approach is shown.

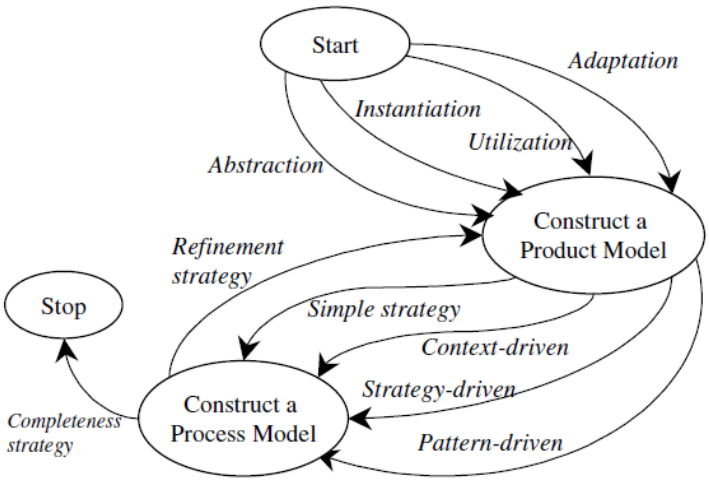

Fig. 2.2. Paradigm-based approach (from [Ralyté03])

The paradigm-based approach is the most generic of the three approaches presented in this survey. Since it is based on meta-modeling, it presents important benefits that are inherited from this technique. For instance, since methods are defined at a high level of abstraction, their understandability is increased (compared to, e.g, textually defined methods), thus contributing to facilitate their application in real ISD projects.

Some examples of relevant Method Engineering proposals that follow this approach are Rolland's [Rolland95] and Grundy's [Grundy96]. On the one hand, Rolland presents in [Rolland95] a proposal for defining *ways-of-working* in a systematic manner. A way-of-working is a process model that takes into account heuristic knowledge to guide humans performing systems development. Specifically, these process models are created by instantiation from a process meta-model that is called NATURE (see section 2.2.5). Furthermore, the product meta-model presented in [Schmitt93] enables the definition of the product part of these models.

On the other hand, Grundy [Grundy96] proposes a product-oriented approach from defining methods. Specifically, he defines a product meta-model called CoCoa that allows method engineers to define the design notations that enable the creation and manipulation of the method products. Furthermore, a process modeling environment called Serendipity is proposed for supporting the definition of process models that coordinate the development of the method products.

2.1.3. The extension-based approach

The extension-based approach [Ralyté03] consists in identifying typical extension situations and performing the required extension of the method by means of extension patterns. A pattern is a component that describes a recurrent problem [Deneckère98], which helps to identify the extension situation, and is defined with its associated solution (the guidelines to be followed when the pattern is applied). Specifically, this solution embodies the process chunk that is to be applied on a particular product [Deneckère98].

In view of this definition of the pattern concept, one may think that the process of extending a method is somewhat similar to the assembly of method fragments. Actually the main difference lies in the nature of the components

that participate in the assembly or the extension. In the former case, the components (i.e. the method fragments/chunks) can be directly used, whereas in the latter they cannot, that is, they have to be generated from the generic patterns.

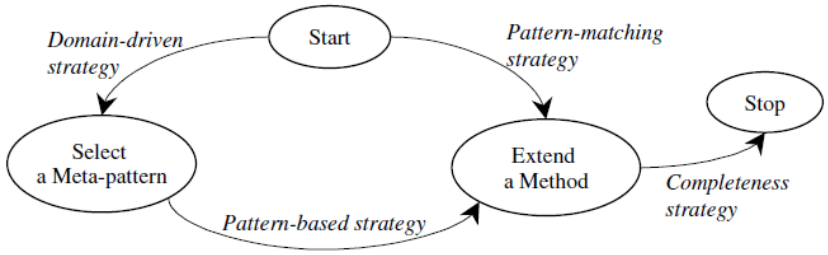

Fig. 2.3. Extension-based approach (from [Ralyté03])

Fig. 2.3 shows the map representing the process underlying the extension-based approach. Even though this approach is the less common of the three approaches, it also presents important advantages. For instance, it is oriented towards guiding the method engineer during the performance of method extensions, which means that it is an adequate approach for performing the adaptation of methods to context needs, one of the main goals of Method Engineering.

An example of proposal that suggests the use of patterns for performing method extensions is [Deneckère98]. Specifically, this work proposes a set of generic patterns for introducing temporal features (such as time constraints) to object oriented models.

2.1.4. General discussion

Method Engineering has a disparate history since many different approaches have been proposed over the past twenty years. However, all these approaches share a common goal: assisting the method engineer during the definition of methods and their adaptation to the context needs. In order to reach this goal, most proposals advocate for the assembly-based approach, but others promote other approaches, such as the paradigm-based or the extension-based. Together, all these proposals have contributed to establish a solid and wide theoretical basis in the area of Method Engineering. However, in spite of this sound basis, **there still remains a need for a Method Engineering proposal**

that takes all the method dimensions into account together. Currently, most of the Method Engineering proposals only focus on the *product* dimension (the products to be constructed during the method) and the *process* dimension (the process to be followed to obtain the products), but other dimensions are also important. These dimensions are basically the *tool* dimension (the software tools that provide support to the *product* and *process* dimensions) [Seligmann89], and the *people* dimension (the agents that make use of the *tools* in order to develop the method *products* following the method *process*) [HendersonSellers10].

This problem has already been noted in other works such as [Iacovelli08]. In order to fill this gap, the methodological framework presented in this master's thesis covers all these four dimensions of methods, as will be shown in chapter 3.

2.2. Method Engineering Languages

Meta-modeling is considered by the Method Engineering community as "the core technique in Method Engineering" [Ralyté03] as it provides an effective way to formalize the abstract syntax of the language that establishes the concepts, constraints and rules that are applicable in the construction of the software production methods. In particular, this subsection presents a survey of some of the most significant languages that have been proposed during the last two decades. For each of these languages a brief overview is given, and later in section 2.3, tools supporting them are described.

2.2.1. ASDM

The semantic data model notation ASDM [Heym92] is a forerunner of current Method Engineering languages and yet it provides a powerful means for representing ISD knowledge. Furthermore, it represents the first attempt to define method semantics, as noted in [Niknafs07]. An overview of the meta-model is presented in Fig. 2.4.

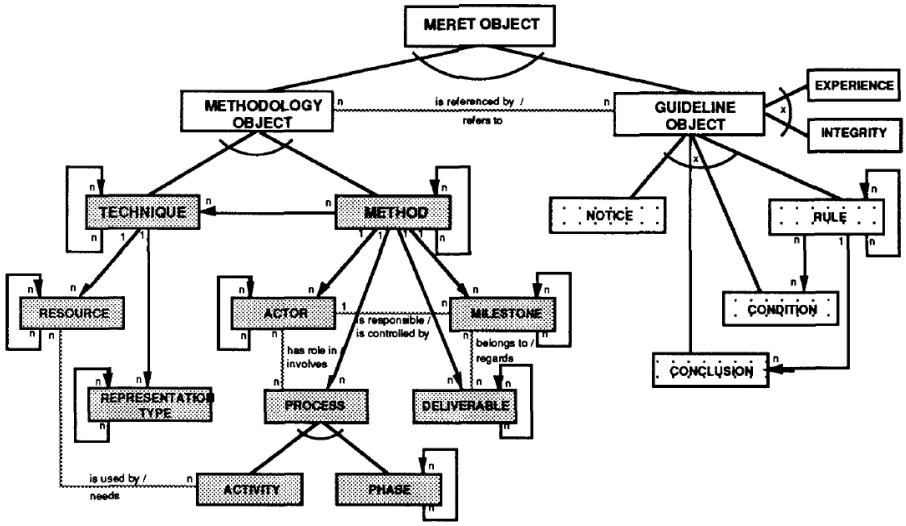

Fig. 2.4. ASDM meta-model (from [Heym92])

As shown in Fig. 2.4, the *MERET Object* is the most general object type. A MERET object can be either a *methodology object* or a *guideline object*. On the one hand, methodology objects embrace all description objects for the specification of a method: *techniques* (used to develop products), *actors*, *milestones*, *processes*, etc. Processes can be either *phases* or *activities* (elementary units of work). On the other hand, guideline objects reflect the more dynamic part of the method knowledge. For instance, a guideline can be a textual *notice* representing experiences from applying a specific part of the method, or an integrity *rule* represented as a horn clause.

In general, the ASDM meta-model provides adequate concepts for specifying software production methods in a product-oriented fashion. Furthermore, it provides a graphical notation that facilitates method comprehension. However, it just embodies a first step towards Method Engineering since it presents important deficiencies. For instance, it provides poor support to the specification of the process part of methods, which negatively affects the possibility of building complete CASE environments from ASDM method specifications.

2.2.2. GOP(P)RR

The GOPRR conceptual data model [Kelly96] is a Method Engineering language that has been specially designed to support the definition of techniques that can be used for the manipulation of the method products. In other words, it supports the definition of modeling languages, such as ER, DFD, UML Class Diagram, etc.

The name GOPRR is an acronym that stands for the metatypes the language operates on: *Graph*, *Object*, *Property*, *Role* and *Relationship*. This metatypes and their relationships are graphically illustrated in Fig. 2.5.

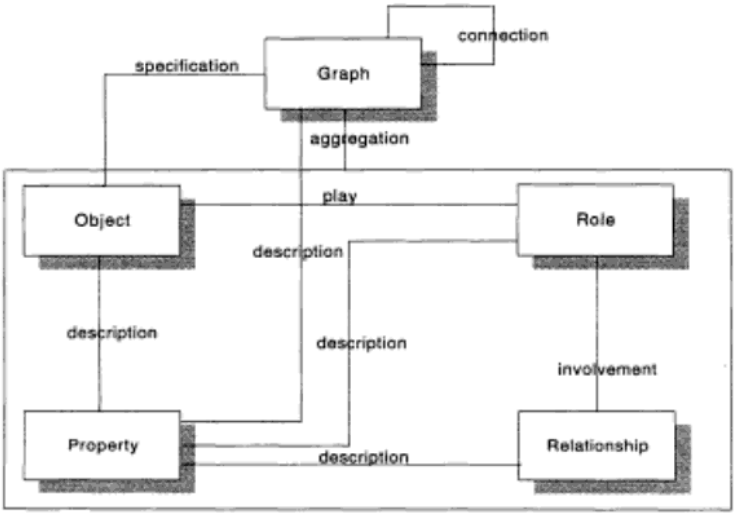

Fig. 2.5. The GOPRR meta-model (from [Harmsen96])

In particular, these concepts represent the following:

- Graph: A graph is a collection of objects and relationships among these objects via roles. An example of graph is a UML class diagram.
- Object: An object is an element that can be placed on its own in a graph. An example of object is a Class that belongs to an UML class diagram.
- Relationship: A relationship is an explicit connection between two or more objects. Relationships attach to objects via roles. An example of a relationship is an Association of a UML class diagram.

- Role: A role specifies how an object participates in a relationship.
- Property: A property is a describing or qualifying characteristic associated with the other types. An example of property is an Attribute that belongs to an UML class diagram.

Furthermore, the notion of *Port* is included in the GOPPRR language [GOPPRR], which represents an evolution of the GOPRR language. Specifically, a port is an optional specification of a specific part of an object to which a role can connect. Normally, roles connect directly to objects, and the semantics of the connection are provided by the role type. If you want a given role type to be able to connect to different places on an object with different semantics, you can add ports to the object's symbol.

To summarize, the GOPRR and GOPPRR languages represent an adequate means for defining modeling notations that can be later used in an integrated CASE environment for creating and manipulating method products. However, this approach presents important lacks. While the CASE tools obtained from GOP(P)RR specifications may provide complete support to the manipulation of method products, they overlook important aspects of ISD such as process enactment and code generation.

2.2.3. MEL and MDM

The Method Engineering Language (MEL) [Brinkkemper01] is a formal representation language that provides concepts and constructs for the textual description, selection and manipulation of method fragments. On the one hand, it provides syntactic constructs to compose from activities complex processes such as sequential execution, conditional branch, iteration, parallel execution and non-deterministic choice. On the other hand, it provides constructs for the detailed specification of the products that these processes need as input and deliver as output.

Furthermore, the semantic aspects of the product fragments can be specified by anchoring the fragment descriptions to an ontology, described by means of the Methodology Data Model (MDM) [Harmsen97]. Anchoring means that a method fragment is described in terms of well-defined basic concepts and associations between those concepts. In particular, the MDM ontology provides the following concepts (CN_0) and associations (A_0):

- CN_0 = {Activity, Actor, Association, Attribute, Attribute Type, Benefit, Business Area, Channel, Communication Protocol, Condition, Cost, Critical Success Factor, Data Flow, Data Collection, Decision, Dialogue, Event, External Entity, Field, Function, Goal, Group, Location, Node, Object, Object Class, Opportunity, Organizational Unit, Problem, Requirement, Role, Rule, Solution, State, Strength, System, Threat, Transition, Weakness}

- A_0 = {Abstraction, Aggregation, Alternative, Balance, Base, Capability, Change, Choice, Component, Connection, Constraint, Consumer, Contents, Dependence, Description, Effect, Employment, Expression, ExternalOutput, Imposition, Input, Interaction, Involvement, Manipulation, Message, Output, Performance, Place, Price, Producer, Product, Request, Resource, Responsibility, Screen, Site, Specialisation, Support, TransitionTrigger, Trigger, Usage}

In summary, MEL can be considered as a complete Method Engineering language. It provides constructs for the definition of both products and process fragments, and also for their manipulation and assembly. Furthermore, it partially covers the method people dimension through predefined property types such as "creator" and "responsible". Unfortunately, the high amount of properties and concepts make it hard to learn, and its textual nature hinders the understanding of the developed methods.

2.2.4. MRSL and MVM

The Method Requirements Specification Language (MRSL) [Gupta01] is a textual language for specifying method requirements in a technology-independent fashion. The main objective of this language is to enable method engineers to express method requirements in simple terms, avoiding the need to have expert knowledge about meta-models and how to instantiate them. These requirements can be later used to (semi)automatically obtain the final method specification and the CASE tool support.

MRSL is based on the Method View Model (MVM) meta-model, which is presented in Fig. 2.6. MVM method concepts are called *things*, and are partitioned into *product entities*, *links* and *constraints*. A link is any *thing* that connects two product entities together. Constraints are those *things* that can be

used by software engineers to specify properties of links and product entities. Finally, any *thing* that is not a link or a constraint is a product entity.

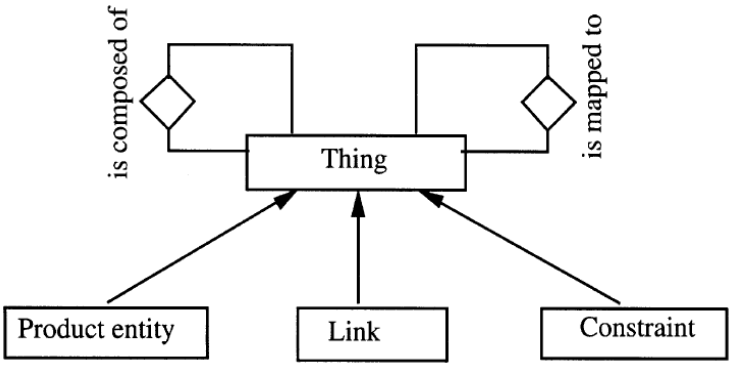

Fig. 2.6. The MVM meta-model (from [Gupta01])

In particular, this language is similar to the GOPRR and GOPPRR languages, in the sense that it supports the definition of modeling languages such as DFD, ER, etc. but not the definition of complete software production methods. Therefore, the CASE environments that can be obtained from method specifications that follow this language overlook important aspects of ISD such as process enactment and code generation.

2.2.5. NATURE

The NATURE[1] modeling formalism [Rolland94, Rolland96b] consists of a set of generic concepts and their relationships for constructing methods from a process perspective, and was designed with a certain philosophy in mind: process models must be contextual, that is to say, the process model must allow users to switch context in a flexible and easy manner. Specifically, a context is composed of the *situation* that is perceived by the method engineer and the specific intention (or *decision*) he/she has in mind. The NATURE meta-model (see Fig. 2.7) addresses these issues by making the notions of situation, decision and context explicit.

---

[1] Novel Approaches to Theories Underlying Requirements Engineering

Thereby, the notion of *context* constitutes the basic building block of NATURE process models. Contexts are defined as couples <situation, decision> that can be linked repeatedly in a hierarchical manner to define *trees*. A tree represents a structured piece of knowledge for supporting decision making in the process. In other words, it is a process fragment which aims at assisting the method engineer in making the most appropriate decision for the situation at hand. Finally, a collection of trees (i.e. hierarchies of contexts) is referred to as a *forest*, which represents the *method*.

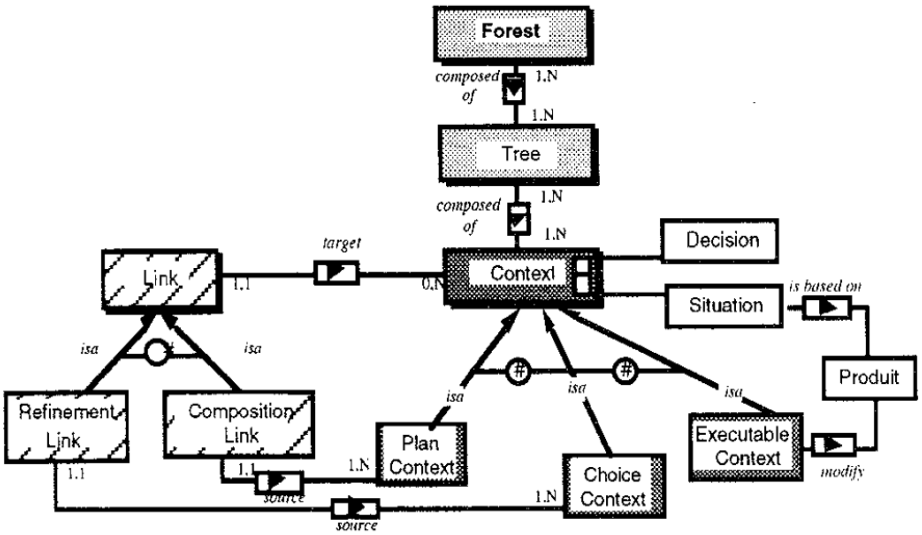

Fig. 2.7. The NATURE meta-model (from [Rolland96b])

To summarize, the NATURE approach represents a powerful means for specifying modular ISD processes that are easy to adapt to context changes and to assemble between each other to compose bigger processes. However, it must be used in combination with a product meta-model in order to specify complete software production methods. Furthermore, this approach does not support the *people* and *tool* dimensions of methods.

2.2.6. SPEM 2.0

In view of the diversity of Method Engineering languages that was emerging in the literature, the OMG[2] proposed the definition of a formal framework for the definition of software production methods and their components. The result is the standard language SPEM (Software & Systems Process Engineering Meta-Model) [SPEM]. Specifically, this section focuses on its version 2.0, released on April 2008.

The SPEM 2.0 meta-model is structured into seven main meta-model packages as depicted in Fig. 2.8. In general, meta-model classes are introduced in lower packages as simply as possible, and then, they are extended in higher packages via the merge mechanism. By means of this mechanism additional properties and relationships can be added in order to realize more complex process modeling requirements.

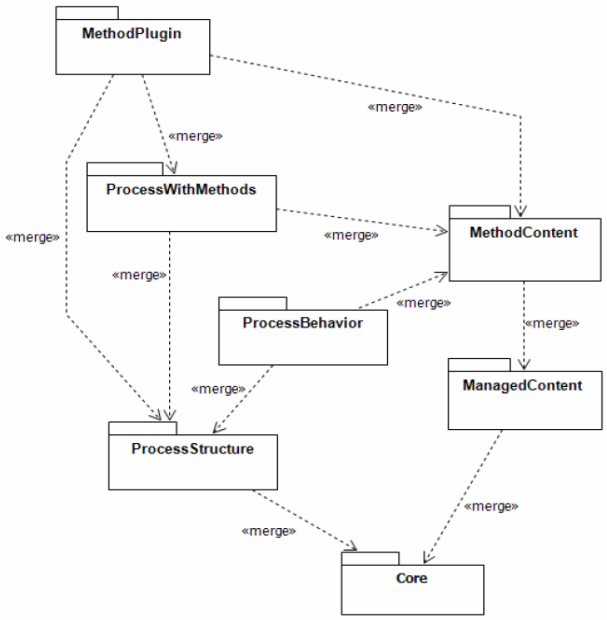

Fig. 2.8. Structure of the SPEM 2.0 meta-model (from [SPEM])

[2] Object Management Group, http://www.omg.org/

- **Core:** This package contains the classes and abstractions that build the base for classes in all other meta-model packages.
- **Process Structure:** It contains the base classes for all process models. Specifically, a SPEM 2.0 process is represented by a breakdown structure composed of *Activities* that reference the performing *Role* classes and the input/output *WorkProduct* classes. Furthermore, it provides mechanisms for process reuse, e.g. process patterns.
- **Process Behavior:** This package contains the classes that enable the use of behavioral models for extending the static breakdown structures built by means of the Process Structure package. However, it does not define its own behavior modeling approach, but rather provides 'links' to existing externally-defined behavior models. For example, a process defined with the Process Structure concepts can be linked to UML 2 Activity diagrams that represent the behavior of such process.
- **Managed Content:** It contains classes for managing the textual documentation of processes (i.e. it enables the association of guidance elements with process structure elements). For instance, a SPEM 2.0 process can be comprised of a combination of instances of the *Guidance* class with a process structure using the relationships defined in this package.
- **Method Content:** This package defines the core elements of every method such as Roles, Tasks, and Work Products. Then, processes would reuse these method content elements and relate them into partially-ordered sequences that are customized to specific types of projects. As a result, SPEM methods provide a clear separation of method content definitions and development processes.
- **Process With Methods:** This package provides the classes that are needed to integrate (i.e. reference) method content into the processes defined using the Process Structure package. These classes can store the changes made to the method content classes that only apply in the specific process.
- **Method Plugin:** This package introduces concepts for managing maintainable, large scale, reusable and configurable libraries or repositories of method content and processes.

In general, SPEM represents an adequate language for Method Engineering since it not only covers the *product* and *process* dimensions of methods but

also the other two, *people* and *tool* (by means of primitives such as *Role*, *RoleSet* and *ToolDefinition*.). Furthermore, it is oriented towards the modular definition of ISD processes, facilitating their assembly from existing parts (a characteristic that is directly related to the Method Engineering principles). However, industry adoption is being slow mainly due to the lack of supporting tools.

2.2.7. ISO/IEC International Standard 24744

The ISO/IEC 24744 [ISO/IEC07] is an International Standard that defines a meta-model for the technology-independent specification of development methodologies in any area, although it is weighed towards software development methodologies. Its scope covers, inter alia, concepts such as *work units*, *work products*, *producers*, *stages* and *model units* (see Fig. 2.9). In addition to the meta-model, a graphical notation is provided to allow method engineers to represent complete methods using graphical constructs.

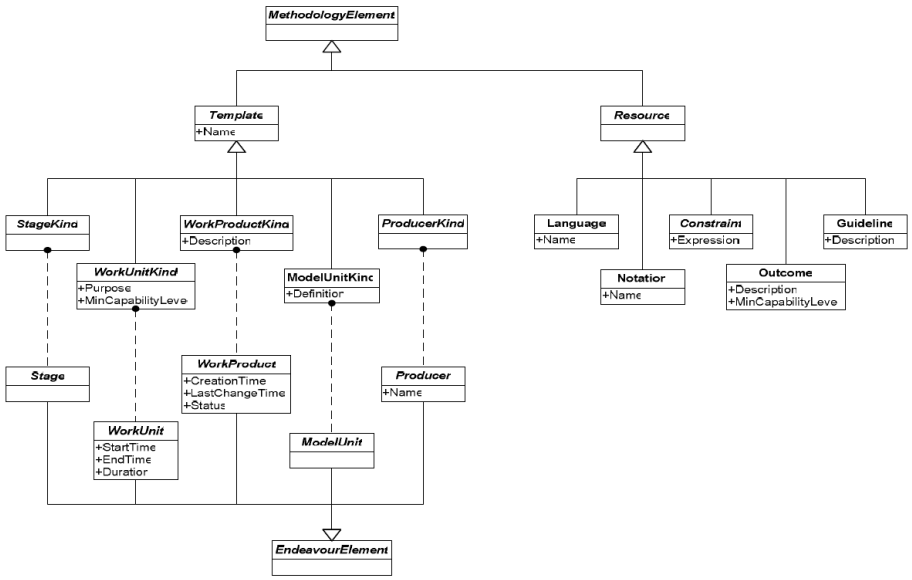

Fig. 2.9. Overall architecture of ISO/IEC 24744 (from [HendersonSellers08b])

One of the main novelties of the standard is the introduction of the powertype pattern concept of Odell [Odell94, GonzalezPerez06a, GonzalezPerez06b, GonzalezPerez08, HendersonSellers08b] as a core

element in the meta-model. Specifically, a powertype pattern is a pair of elements. The instances of one of them reside in the method domain and the others in the enactment domain. The elements in the enactment domain represent actual elements in use by the people on a particular project (e.g. actual tasks). On the other hand, the elements in the method domain represent method elements as they are specified in the model (e.g. instances of the meta-class Task). The main advantage of this approach is that it allows some attributes of the powertype to be inherited by the element in the method domain with values already allocated to them, while others remain "traditional" attribute specifications that get their value in the enactment domain.

In summary, this language embodies another standardization effort in the field of method definition. It represents a rather complete language for Method Engineering since it covers the *process*, *product* and *people* dimensions of methods. One of its distinctive characteristics is that it provides primitives for specifying elements that reside either on the instance level (i.e. enactment level) or the method level. This approach allows some attributes defined in the meta-model to be given values in any of both levels.

2.2.8. General discussion

This survey illustrates that there is a wide diversity of languages and all of them have their advantages and drawbacks. This conclusion is also drawn in several studies, such as [Harmsen96] and [Niknafs08]. These studies conclude that there is not ultimate Method Engineering language and, therefore, the choice of the language depends on the specific purpose and goals that one wants to achieve.

In order to contribute to improve this situation, standardization efforts are being made, e.g. the SPEM [SPEM] and ISO/IEC 24744 [ISO/IEC07] initiatives. These standards aim to provide languages dedicated to method specification that do not present the deficiencies found in previous proposals.

Unfortunately, while these standards represent adequate means to perform the definition of software production methods, **Method Engineering proposals that make use of these standards are still non-existent**. Indeed, in [HendersonSellers10] it is predicted that one of the likely topics for

research initiatives in the next years will be a new generation of CAME tools based on internationally standardized methodology meta-models.

In order to fill this gap, the methodological framework presented in this master's thesis proposes the use of the SPEM standard for the construction of the method specifications. Specifically, chapter 3 describes in detail how these specifications are built and how they are later used for the (semi)automatic generation of the CASE tool support. Then, chapter 4 gives implementation details of a CAME environment that supports these tasks.

2.3. Method Engineering Tools

The Method Engineering lifecycle is a complex and error-prone process that cannot be properly performed without automated tool support. The first Method Engineering supporting tools date back to the early days of Method Engineering, when the first academic prototypes were first introduced [Niknafs08].

In general, there are two different types of tools: Computer Aided Method Engineering (CAME) environments [Kumar92, Harmsen97, Rolland97, ArniBloch01, Saeki03] and MetaCASE tools [Martin94, Roger97, Isazadeh97, Englebert99]. Specifically this subsection presents, for each of these categories, a brief description and a survey of some of the most significant tools that have been developed during the last two decades.

2.3.1. Computer Aided Method Engineering (CAME)

Computer Aided Method Engineering (CAME) environments aim at supporting the definition of software production methods by means of languages such as NATURE or ASDM (see section 2.2). Thus, CAME environments are mainly focused on the method design phase of the Method Engineering lifecycle.

Fig. 2.10 illustrates the general architecture of CAME environments. As shown in the figure, CAME environments are made up of two parts: (1) the CAME part and (2) the CASE part.

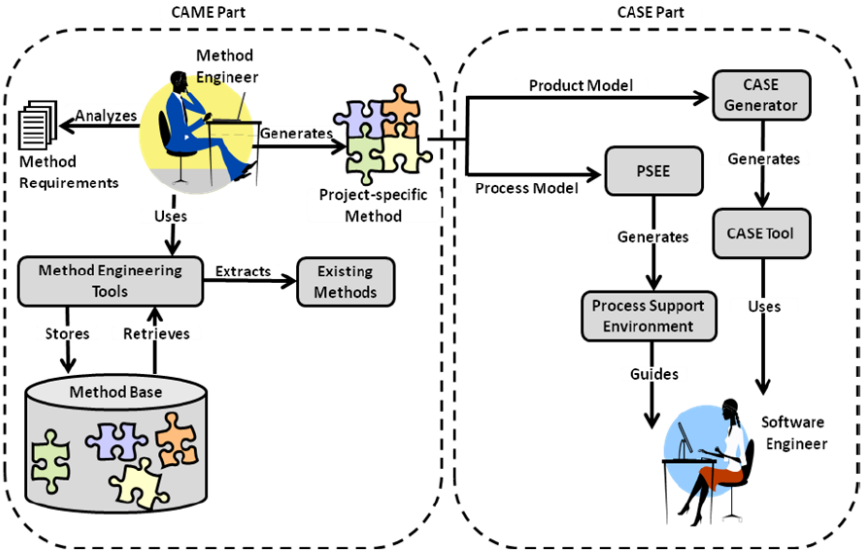

Fig. 2.10. General architecture of CAME environments (from [Niknafs08])

On the one hand, the CAME parts offers facilities for method definition. Some examples of the functionalities that must be provided in this part are the following:

- Storage of method fragments/chunks in a repository (typically called Method Base).
- Definition of properties that enable the search and retrieval of method fragments/chunks from the repository.
- A query language for accessing the contents of the repository.
- Composition of method fragments/chunks.
- Support and guidance for the method engineer.

On the other hand, the main goal of the CASE part is to produce CASE tools and process support environments that enable the enactment of the method specified in the CAME part. For this purpose, the CASE part takes the method specification as input and offers means to manually or semi-automatically produce these tools.

MERET

The Methodology Representation Tool (MERET) [Heym92] can be seen as the first approach towards a CAME tool for the specification, storage and further development of ISD knowledge. Specifically, it supports the specification of methods in a product-oriented fashion by means of the semantic data model ASDM, which is detailed in section 2.2.1. In addition, it addresses the integration of integrity rules and consistency checks on the method specifications.

One of the main drawbacks of this tool is that it only supports the specification of software production methods, lacking CASE tool generation capabilities. Furthermore, due to the product-oriented nature of ASDM, it provides poor support for defining the process dimension of methods.

Decamerone

Decamerone [Harmsen95] is a CAME tool that provides facilities for specifying, storing and selecting method fragments, and for assembling them into a method. To perform these tasks, the tool provides the language MEL, described in section 2.2.3.

Fig. 2.11 shows the architecture of decamerone, which is divided into two parts: the CAME part and the CASE part. On the one hand, the CAME part is dedicated to the method design and contains the following components:

- **The user interface:** provides the required tools to perform the specification, selection and assembly of method fragments by means of the language MEL.
- **Method Base Management System (MBMS):** is the kernel of Decamerone. It provides the operations that are necessary to interact with the Method Base repository.
- **The MEL interpreter:** translates MEL specifications into MBMS function sequences.

On the other hand, the CASE part contains the required tools for the enactment of the method: a CASE tool repository, a process manager and a user interface that provides the editors that enable the system specification.

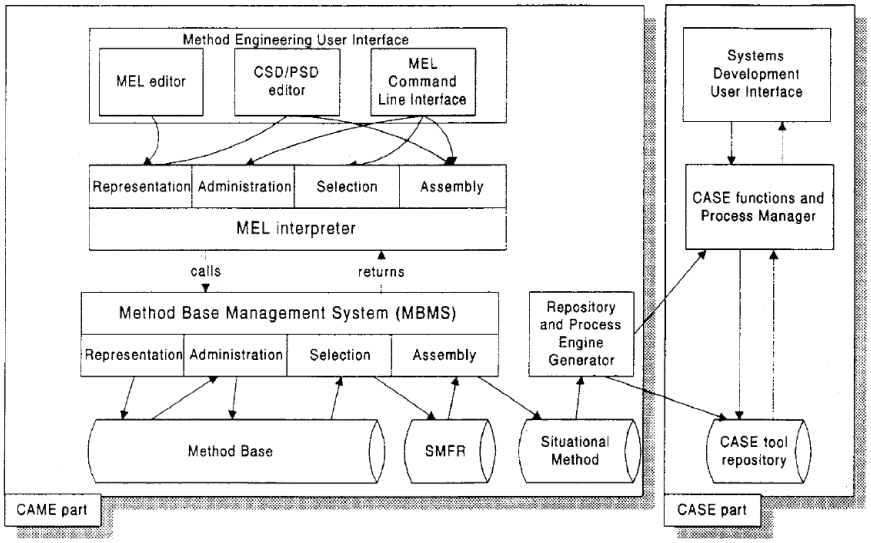

Fig. 2.11. Architecture of Decamerone (from [Harmsen95])

In summary, Decamerone is a rather complete CAME environment. It supports the definition of methods by means of the MEL language, which not only allows the method engineer to define product and process fragments but also offers constructs for their manipulation (selection, storage, assembly, etc.). Furthermore, Decamerone supports the definition of the semantics of method fragments by means of the MDM ontology, and the generation of CASE tools that support both the product and process parts of methods. However, it also presents some deficiencies. For instance, it provides a poor graphical meta-model, and the textual nature of MEL complicates the understanding of the specified methods. In addition, the generated CASE tools lack code generation capabilities (i.e. the creation of method products by means of automatic tasks).

MENTOR

MENTOR [Si-Said96, Plihon96] is a CAME environment that aims at improving the productivity of method engineers by facilitating the construction of project-specific methods. It is based on the NATURE contextual approach, which is described in section 2.2.5.

Fig. 2.12 illustrates the architecture of MENTOR, which is composed of four main components:

- **The Method Engineering Environment:** this component contains viewers, editors and a generator. The viewers allow the method engineer to browse method fragments. The editors enable the graphical description of both product and process parts of methods. Finally, the generator aids in the automatic instantiation of predefined generic patterns.
- **The Application Engineer Environment:** this component represents the CASE part of MENTOR and contains the product editors that permit the development of the system specification. Furthermore, it contains a traceability tool that keeps track of product and process traces, and a process change manager that keeps coherent the element used during the process enactment when the process is modified.
- **The Guidance Engine:** this is the core component of MENTOR. It guides the method engineer in the performance of the Method Engineering tasks and enables the enactment of the specified process model.
- **The repository:** is structured in three levels that are interrelated: (1) the meta level, (2) the method level and (3) the workspace level. These levels contain respectively the product and process meta-models, method fragments, and process models and products under development.

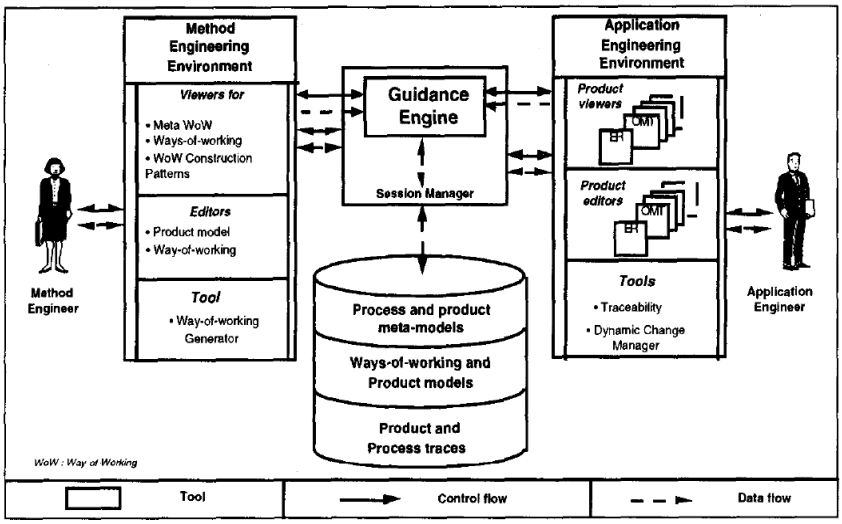

Fig. 2.12. Architecture of MENTOR (from [Si-Said96])

To summarize, MENTOR is one of the most complete CAME tools. First of all, it supports the specification of method requirements, which is usually overlooked in most CAME environments. Furthermore, it supports both the assembly-based and paradigm-based approaches for method specification, and the construction of CASE tools that support the specified methods. However, it also presents some drawbacks. For instance, the generated CASE tools lack code generation, since they are only composed of product editors, a traceability tool and a process change manager. Furthermore, its graphical design is not intuitive, which negatively affects its usability.

Method Editor

Method Editor [Saeki03] is a CAME environment that uses UML as its meta-modeling language. In particular, the product part of methods is specified by means of the UML Class Diagram and the process part by means of the UML Activity Diagram. The specified methods are used by a *diagram generator* and a *navigator generator*. These tools perform the generation of the CASE environment that supports the method. This CASE tool is composed of a series of diagram editors that enable the creation and manipulation of the method products and browsing pages that guide the software engineer through the enactment of the method process.

In summary, Method Editor is one of the few CAME environments that support standard techniques such as UML. It is rather complete since it supports the specification of methods and the generation of CASE tools. Moreover, it provides a very intuitive graphical design. However, the main drawback of Method Editor is that, even though it supports CASE tool generation, these CASE tools only contain graphical editors that enable the creation/manipulation of the method products, and navigation pages that guide through the method process. Other aspects such as code generation or consistency checkers should be considered.

2.3.2. MetaCASE

Traditional CASE tools provide support to a single software production method. However, one fixed method simply cannot work for all software development projects and organizations as they differ significantly from one another and evolve over time. Therefore, CASE environments should be

adapted to meet the context needs, but this is not possible because the tools that support the methods are "hard-coded" in the environment.

The metaCASE technology aims at solving this problem. To achieve this goal, metaCASE tools add an additional level above the method level (see Fig. 2.13) in order to provide the ability to specify at a high level of abstraction the tools that are required to support the method, and then generate the CASE environment from these specifications.

As a result, unlike CAME environments (which are focused on the method design), metaCASE tools concentrate on the CASE tool construction (the method implementation phase of the Method Engineering lifecycle).

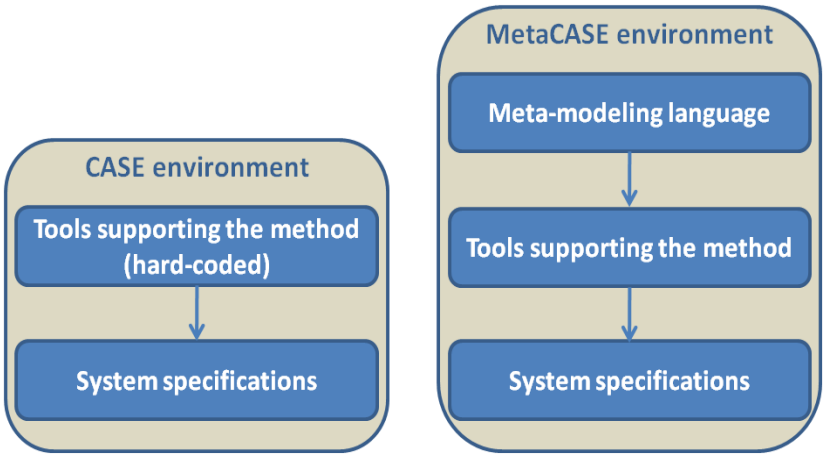

Fig. 2.13. CASE tool versus metaCASE tool

MetaEdit+

MetaEdit+ [Kelly96] is, up to our knowledge, the only Method Engineering tool that has been commercialized. It is a metaCASE environment based on the conceptual data model GOPPRR, described in section 2.2.2. By means of this language, MetaEdit+ enables the specification at a high level of abstraction of the modeling languages (in MetaEdit called "methods") that have to be supported by the CASE tool under construction.

MetaEdit+ consists of several tool families. In particular, the family of tools that enable the specification of methods is the *Method Management Tools* family (see Fig. 2.14). This family is composed of the following tools:

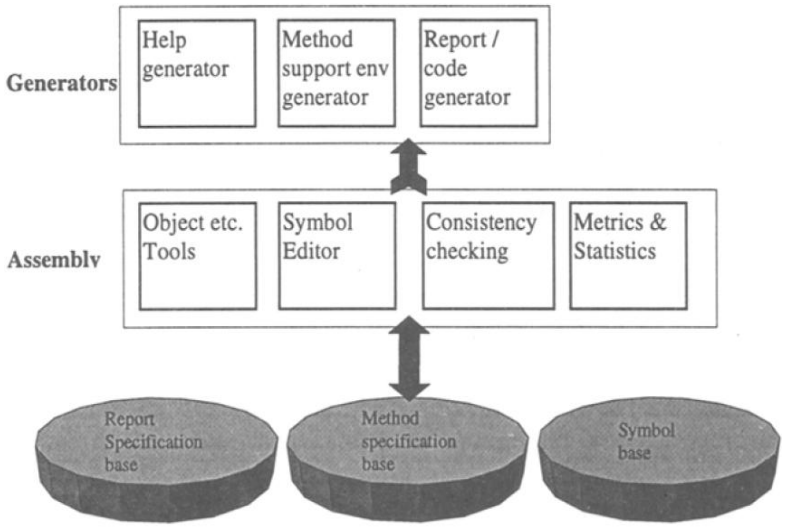

Fig. 2.14. Method Management Tools in MetaEdit+ (from [Kelly96])

- **The Method Base:** this repository stores method fragments and the symbols used for representing object types.
- **The Method Assembly System:** consists of the specialized tools that are needed for method assembly, such as *meta-model editors*, which allow the method engineer to specify methods by means of the GOPPRR language.
- **The Environment Generation System:** this system consists of the generators that take as input the method specifications and deliver the CASE tools.

In general, MetaEdit+ embodies an efficient solution for defining your own modeling languages. It is easy to use, well documented and has an intuitive graphical design. However, MetaEdit+ (and all metaCASE environments in general) falls short in providing adequate support to Method Engineering. This is due to the fact that these tools are focused on supporting CASE tool construction and overlook one fundamental aspect of Method Engineering: the definition of software production methods.

MERU

The Method Engineering Using Rules (MERU) [Gupta01] metaCASE environment offers the method engineer the textual language MRSL, which is

built upon the MVM meta-model (see section 2.2.4). This language allows the method engineer to specify the method requirements in a technology-independent fashion. The document that is produced is called *Method Requirements Specification* (MRS) and is used to (semi)automatically obtain the final method specification and finally the CASE tool support.

MERU, like MetaEdit+, embodies an adequate tool for supporting the definition of modeling languages. Specifically, it provides a high number of features, such as process enactment support and method requirements specification. However, it lacks the possibility to specify software production methods that can assist the execution of real ISD projects.

2.3.3. General discussion

The survey presented in this section shows that in many Method Engineering initiatives CAME and metaCASE tools are developed in order to provide software support to their proposals. However, CAME and metaCASE technology is still immature, since most of these environments just represent incomplete prototypes that have only been used for academic purposes [Niknafs08]. The main problem with these tools is that, in general, **CAME and metaCASE environments provide inadequate coverage of the Method Engineering lifecycle**. The main reason for this is that, on the one hand, CAME environments generally focus on the method design and, on the other hand, metaCASE environments concentrate on the method implementation. That is to say, CAME tools usually provide rich ways to specify software production methods but offer limited (or non-existent) CASE tool generation capabilities. On the other hand, metaCASE tools provide adequate means for building CASE environments but lack the possibility to define software production methods that can be enacted in real projects.

In order to fill this gap, the methodological framework presented in this thesis equally encompasses the method design and the method implementation. Therefore, the CAME environment that has been developed to support the proposal not only provides means for performing the definition of software production methods, but also for (semi)automatically obtaining the CASE tool support.

2.4. Conclusions

The survey presented in this chapter illustrates that the Method Engineering literature is very extensive. In particular, some of the most significant Method Engineering approaches, languages and tools have been presented and the following shortcomings have been identified:

Shortcoming 1. There still remains a need for a Method Engineering proposal that takes all the method dimensions into account together (i.e. the *product*, *process*, *tool* and *people* dimensions). Most of the existing proposals cover the product and process parts of methods, but the tool and people dimensions are almost completely overlooked.

Shortcoming 2. Method Engineering proposals that make use of standards for method definition (such as SPEM and ISO/IEC 24744) are still non-existent.

Shortcoming 3. CAME and metaCASE environments provide inadequate coverage of the Method Engineering lifecycle. In general, CAME environments focus on the method design and metaCASE environments on the method implementation.

The methodological framework proposed in this master's thesis addresses these shortcomings. Specifically, it proposes the use of the SPEM standard for performing the method design (**shortcoming 2**). This standard adequately supports the definition of methods that cover the product, process and people dimensions. The tool dimension is covered by means of the use of technical fragments [Harmsen97] (**shortcoming 1**). Furthermore, in order to equally encompass the method design and implementation (**shortcoming 3**), the framework is founded on an MDD infrastructure that is based on meta-modeling and model transformation techniques. The meta-modeling techniques enable the definition of methods as models, and the model transformations enable to (semi) automatically obtain CASE tools from these models.

References

[ArniBloch01] Arni-Bloch, N.: Towards a CAME Tools for Situational Method Engineering. In Proceedings of the 1st International Conference on Interoperability of Enterprise Software and Applications, Geneva (2001)

[Bergstra85] Bergstra, J., Jonkers, H., Obbink, J.: A Software Development Model for Method Engineering. In: Roukens J., Renuart J. (eds.) Esprit 1984: Status Report of Ongoing Work. Elsevier Science Publishers, Amsterdam, (1985)

[Brinkkemper96] Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Information and Software Technology, 38, 275-280, (1996)

[Brinkkemper98] Brinkkemper, S., Saeki, M., Harmsen, F.: Assembly Techniques for Method Engineering. CAiSE '98: Proceedings of the 10th International Conference on Advanced Information Systems Engineering, Springer-Verlag, 381-400 (1998)

[Brinkkemper99] Brinkkemper, S.; Saeki, M., Harmsen, F.: Meta-Modelling Based Assembly Techniques for Situational Method Engineering. Information Systems, 24, 209-228 (1999)

[Brinkkemper01] Brinkkemper, S., Saeki, M., Harmsen, F.: A Method Engineering Language for the Description of Systems Development Methods. CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering, Springer-Verlag, 473-476 (2001)

[Deneckère98] Deneckère, R., Souveyet, C.: Patterns for Extending An OO Model with Temporal Features. In OOIS'98 Proceedings, C. Rolland, G. Grosz, Eds. Springer-Verlag, London, 201-218 (1998)

[Englebert99] Englebert, V., Hainaut, J.-L.: DB-MAIN: A Next Generation Meta-CASE. Information Systems, 24, 99-112 (1999)

[GonzalezPerez06a] Gonzalez-Perez, C., Henderson-Sellers, B.: A Powertype-Based Metamodelling Framework. Software and Systems Modeling, 5, 72-90 (2006)

[GonzalezPerez06b] Gonzalez-Perez, C., Henderson-Sellers, B.: On the Ease of Extending a Powertype-Based Methodology Metamodel. In Meta-Modelling and Ontologies. Proceedings of the 2nd Workshop on Meta-Modelling, WOMM'06, P-96, 11-25 (2006)

[GonzalezPerez08] Gonzalez-Perez, C., Henderson-Sellers, B.: Metamodelling for Software Engineering. Wiley Publishing (2008)

[GOPPRR] MetaCASE Consulting: Method Workbench User's Guide, MetaCASE Consulting, Jyväskylä, Finland (2005), http://www.metacase.com/support/40/manuals/mwb40sr2a4.pdf

[Grundy96] Grundy, J. C., Venable, J. R.: Towards an Integrated Environment for Method Engineering. In proceedings of the IFIP 8.1/8.2 Working Conference on Method Engineering, Hall, 45-62 (1996)

[Gupta01] Gupta, D., Prakash, N.: Engineering Methods from Method Requirements Specifications. Requirements Engineering, 6, 135-160 (2001)

[Guzélian07] Guzélian, G., Cauvet, C.: SO2M : Towards a Service-Oriented Approach for Method Engineering. In: the 2007 World Congress in Computer Science, Computer Engineering and Applied Computing, in Proceedings of International Conference Information and Knowledge Engineering IKE'07, Las Vegas, Nevada, USA (2007)

[Harmsen95] Harmsen, F., Brinkkemper, S.: Design and Implementation of a Method Base Management System for a Situational CASE Environment. Proceedings of the Second Asia Pacific Software Engineering Conference, IEEE Computer Society, 430 (1995)

[Harmsen96] Harmsen, F., Saeki, M.: Comparison of Four Method Engineering Languages. Proceedings of the IFIP TC8, WG8.1/8.2 Working

Conference on Method engineering : Principles of Method Construction and Tool Support, Chapman & Hall, Ltd., 209-231 (1996)

[Harmsen97] Harmsen, A.F.: Situational Method Engineering. Moret Ernst & Young (1997)

[HendersonSellers08a] Henderson-Sellers, B., Gonzalez-Perez, C., Ralyté, J.: Comparison of Method Chunks and Method Fragments for Situational Method Engineering. ASWEC '08: Proceedings of the 19th Australian Conference on Software Engineering, IEEE Computer Society, 479-488 (2008)

[HendersonSellers08b] Henderson-Sellers, B., Gonzalez-Perez, C.: Standardizing Methodology Metamodelling and Notation: An ISO Exemplar. Information Systems and e-Business Technologies, Springer Berlin Heidelberg, 5, 1-12 (2008)

[HendersonSellers10] Henderson-Sellers, B., Ralyté, J.: Situational Method Engineering: State-of-the-Art Review. Journal of Universal Computer Science, 16, 424-478 (2010)

[Heym92] Heym, M., Österle, H.: A Semantic Data Model for Methodology Engineering. In: 5th Workshop on Computer-Aided Software Engineering, pp. 142-155. IEEE Press, Los Alamitos (1992)

[Iacovelli08] Iacovelli, A., Souveyet, C., Rolland, C.: Method as a Service (MaaS). RCIS, 371-380 (2008)

[Isazadeh97] Isazadeh, H., Lamb, D. A.: CASE Environments and MetaCASE Tools, (1997)

[ISO/IEC07] ISO/IEC 24744: Software Engineering: Metamodel for Development Methodologies. International Standards Organization/ International Electrotechnical Commission, Geneva (2007)

[Kelly96] Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. CAiSE, 1-21 (1996)

[Kumar92] Kumar, K., Welke, R. J.: Methodology Engineering: A Proposal for Situation-Specific Methodology Construction. Challenges and Strategies for Research in Systems Development, John Wiley & Sons, Inc., 257-269 (1992)

[Martin94] Martin, C.: MetaCASE: dream or reality. Electro'94 International Conference, 195-199 (1994)

[Mirbel06] Mirbel, I., Ralyté, J.: Situational method engineering: combining assembly-based and roadmap-driven approaches. Requirements Engineering, 11, 58-78, (2006)

[Mirbel07] Mirbel, I.: Connecting method engineering knowledge: a community based approach. Situational Method Engineering, 176-192 (2007)

[Niknafs07] Niknafs, A., Asadi, M., Abolhassani, H.: Ontology-Based Method Engineering. International Journal of Computer Science and Network Security, 7, (2007)

[Niknafs08] Niknafs, A., Ramsin, R.: Computer-Aided Method Engineering: An Analysis of Existing Environments. CAiSE, 525-540 (2008)

[Odell94] Odell, J.J.: Power Types. Journal of Object-Oriented Programming 7(2), 8-12 (1994)

[Plihon96] Plihon, V.: MENTOR: An Environment Supporting the Construction of Methods. Asia-Pacific Software Engineering Conference, IEEE Computer Society, 0, 384 (1996)

[Prakash97] Prakash, N.: Towards a Formal Definition of Methods. Requirements Engineering. 1: Vol. 2. - pp. 23-50 (1997)

[Prakash99] Prakash, N.: On method statics and dynamics. Information Systems, Elsevier Science Ltd., 24, 613-637 (1999)

[Punter96] Punter, H.T., Lemmen, K.: The MEMA Model: Towards a New Approach for Method Engineering. Information and Software Technology 38(4), 295-305 (1996)

[Ralyté99] Ralyté, J.: Reusing Scenario Based Approaches in Requirement Engineering Methods: CREWS Method Base. DEXA '99: Proceedings of the

10th International Workshop on Database & Expert Systems Applications, IEEE Computer Society, 305 (1999)

[Ralyté01] Ralyté, J., Rolland, C.: An Assembly Process Model for Method Engineering. CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering, Springer-Verlag, 267-283 (2001)

[Ralyté03] Ralyté, J., Deneckère, R., Rolland, C.: Towards a generic model for situational method engineering. CAiSE'03: Proceedings of the 15th international conference on Advanced information systems engineering, Springer-Verlag, 95-110 (2003)

[Ralyté05] Ralyté, J., Rolland, C., Ayed, M. B.: An Approach for Evolution-Driven Method Engineering. Information Modeling Methods and Methodologies, 80-101 (2005)

[Roger97] Roger, J. E., Suttenbach, R., Ebert, J., Uhe, I.: Meta-CASE in Practice: a Case for KOGGE. Springer , 203-216 (1997)

[Rolland94] Rolland, C.: Modeling the Evolution of Artifacts. ICRE'94: Proceedings of the 1st International Conference on Requirements Engineering, Colorado, 216-219 (1994)

[Rolland95] Rolland, C., Souveyet, C., Moreno, M.: An approach for defining ways-of-working. Information Systems, Elsevier Science Ltd., 20, 337-359 (1995)

[Rolland96a] Rolland, C., Prakash, N.: A proposal for context-specific method engineering. Proceedings of the IFIP TC8, WG8.1/8.2 working conference on Method engineering : principles of method construction and tool support, Chapman & Hall, Ltd., 191-208 (1996)

[Rolland96b] Rolland, C., Plihon, V.: Using Generic Method Chunks to Generate Process Models Fragments. Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96), IEEE Computer Society, 173 (1996)

[Rolland97] Rolland, C.: A Primer For Method Engineering. Proceedings of the INFORSID Conference, 10-13 (1997)

[Rolland99] Rolland, C., Prakash, N., Benjamen, A.: A Multi-Model View of Process Modelling. Requirements Engineering Journal 4(4), 169-187 (1999)

[Saeki03] Saeki, M.: CAME: The First Step to Automated Method Engineering. In OOPSLA'03: Workshop on Process Engineering for Object-Oriented and Component-Based Development, 7-18 (2003)

[Schmitt93] Schmitt, J.R.: Product Modeling in Requirements Engineering Process Modeling. IFIP TC8 International Conference on Information Systems Development Process, North Holland, (1993)

[Seligmann89] Seligmann, P.S., Wijers, G. M., Sol, H.G.: Analyzing the structure of I.S. methodologies, an alternative approach. In Proc. of the 1st Dutch Conference on Information Systems, Amersfoort, The Netherlands (1989)

[Si-Said96] Si-Said, S., Rolland, C., Grosz, G.: MENTOR: A Computer Aided Requirements Engineering environment. Advanced Information Systems Engineering, Springer Berlin / Heidelberg, 1080, 22-43 (1996)

[Slooten93] Slooten, K. v., Brinkkemper, S.: A Method Engineering Approach to Information Systems Development. In: Proceedings of the IFIP WG8.1 Working Conference on Information System Development Process, North-Holland Publishing Co., 167-186 (1993)

[SPEM] Software & Systems Process Engineering Meta-model (SPEM) OMG Available Specification version 2.0. OMG Document Number: formal/2008-04-01