Generic Models for Engineering Methods of Diverse Domains

N. Prakash[1] and M. P. S. Bhatia[2]

[1]JIIT A10, Sector 62 NOIDA 201307,India
praknav@hotmail.com
[2]NSIT Sector 3, Dwarka, New Delhi 110045, India
bhatia_mps@hotmail.com

Abstract. The 3-layer architecture for methods consisting of the generic model, meta-model, and method layers is considered and the advantages of introducing the generic layer are brought out. It is argued that this layer helps in method selection, construction of methods in diverse domains and has the potential of rapidly engineering methods. A generic model is introduced and then instantiated from meta-models from the Information Systems and Robotics and Project Planning domains. Thereafter methods are produced by an instantiation of these meta-models. Computer based support for the 3-layer architecture is considered and notion of a generic-CASE tool is introduced.

1 Introduction

A number of meta-models (Gro97, Pra97a, Pra97b, Pra99a, Pra99b, Rol95, Sou91) have been developed in the last decade for a range of uses

- Understanding conceptual models of the 1970s and 1980s
- Identifying commonalties between models
- Representing product and process aspects of methods
- Understanding process models
- Representing quality aspects of methods
- Developing techniques for Requirements Engineering
- Investigating Data Warehouses

Indeed meta-models have emerged as a principal means to understanding the activities seen in the area of Information Systems. Meta-modelling has been used in the development of CASE tools including those for Requirements Engineering. It has also been used for Method Engineering and more specifically, in the development of CAME tools.

In method engineering, meta-models provide an abstraction of method concepts, inter-relationships between these, and constraints binding these together into a coherent system that forms a method. However, as noted in (Pra97a), whereas meta-models provide useful abstractions of methods, they do not clearly articulate the

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 612-625, 2002.

essential nature of the methods they deal with and do not address the critical issue of what *is* a method. Consequently, the class of methods being modelled in a meta-model determines the nature of methods assumed by the meta-model. Thus, in their early period, meta-models emphasised only the product aspects of methods. Later, the dichotomy of product and process aspects was modelled or the process aspects were emphasised.

A comprehensive 3-layer framework was proposed (Pra99a) in which the highest layer is the generic layer where the nature of a method is articulated, the middle layer is the meta-model layer and, finally, the lowest layer is the method layer. The successive layers were seen as instantiations of their immediately higher layer. This framework was illustrated with a generic model whose instantiation resulted in a meta-model of a method that, in turn, was instantiated as a method. Subsequently, the method engineering activity to support this and an associated CAME tool called MERU (Gup01), have been developed.

In this paper, we focus on the generic layer and raise two questions. The first is concerning the usefulness of the generic layer itself: What does it achieve? what does it get for us? As indicated earlier, we believe that the generic layer deals with the essential nature of methods and makes explicit the domain of these methods. It helps in the following ways:

Since it makes explicit the nature of the domain, it promotes the development of methods and processes specific to that domain. We expect this to lead to greater method acceptability among application engineers.

Again, by making explicit the nature of domain, it helps us in understanding and comparing different domains.

It helps in meta-method engineering. A meta-model can be checked out for consistency and completeness under the generic view. This can be facilitated by constructing Generic-CAME tools.

A Generic tool can be developed to produce CASE tools directly from domain specific models thus speeding up the construction of domain specific tools.

The generic approach can also be used for simulating real process and real systems before they are actually implemented. For example, in the Robotics domain, the entire robot can be conceptualised, and a method produced. Using CASE support, application processes can be built to check that the robot does, in fact, handle the proposed range of tasks. If not then the robot can be re-conceptualised. Thus, before constructing the real robot, its viability can be fully established.

The second question we raise here is regarding the generic model inhabiting the generic layer. What is the system of concepts that makes the generic model truly generic? In other words, can the generic model be used for methods of a wide range of domains, not just the IS domain? Indeed, if such a generic model can be found then the domain of IS would contribute significantly to other domains.

In this paper we first present a generic model. Thereafter we consider three meta-models, one in the domain of Information Systems, Robotics and Project Planning. We show that these could be defined by instantiating the generic model. We then consider the method level for the Information Systems, the Robotics and the Project Management domains by instantiating the meta-models. The advantages of these approaches in the three cases are then brought out.

The layout of the papers is as follows. In the next section, the generic model is presented. Thereafter in sections 3, 4 and 5 respectively the Information Systems, Robotics and Project Planning meta-models are presented. Also, an instantiation of these to yield methods is considered in these sections. The differences between the meta-models of the three domains are highlighted. In section 5 we outline the nature of the computer support that can be provided. Generic-CASE tools at higher level of abstraction then the meta-CASE tools of today. The generic-CASE tool is at two levels.

2 The Generic Model

The Generic View of a method (Pra99b) is based on the view of a method as an artifact that provides the capability to construct a product. A method has two aspects to it, its static and dynamic properties. Static properties of interest, from a modeling point of view, are those that describe the holistic structure of the artifact, identify the component parts of the artifact, and bring out the inter-relationships between these components. The dynamic properties of artifacts are those that describe the interaction of the artifact with its environment. This interaction may be viewed in holistic terms, as the interaction of the entire artifact, or in terms of its components.

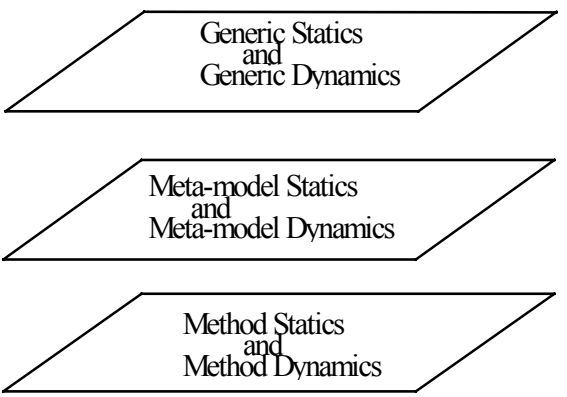

Fig. 1: The Three-Layered Architecture

As shown in Fig. 1 a method is organised in three layers, the generic layer, the meta-model, and the method layer. Each lower layer is an instantiation of its immediately higher layer.

In the *generic* view, the interest is on the intrinsic notion of a method, namely, (a) what it is, (b) what it can do, (c) how this capability is achieved, (d) what kinds of methods exist and what are their inter-relationships. Thus, the generic view looks at methods in meta-model independent terms and seeks to uncover their very essence. In this paper, we will deal with method statics: generic, meta-model, and method. The genericity of method dynamics is the subject of another paper.

Generic Method Statics

In this section we provide an overview of generic method statics. Details can be found in (Pra97a) and (Par97b).

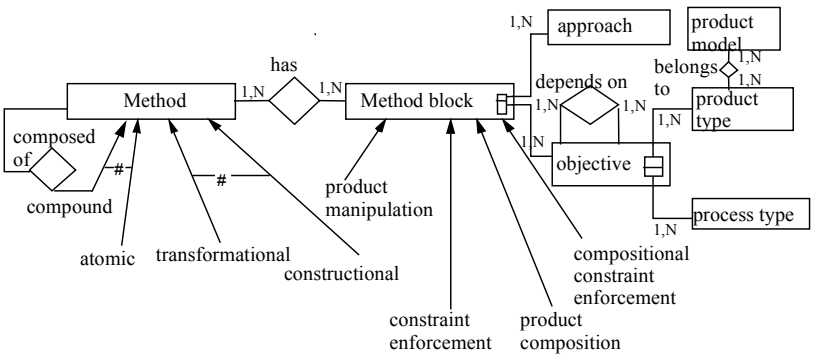

Fig. 2: Generic Method Statics

As shown in Fig. 2, there are two kinds of methods, *transformational* and *constructional*. A *transformational* method is used for transforming a product, expressed in one or more product models, into a product of other product model(s). In contrast, a *constructional* method is used whenever a new product, expressed in one or more product models, is to be constructed.

Any method can be *atomic* or compound. An *atomic* method deals only with those products, which are expressed, in exactly one product model. A *constructional* method, which builds products for the ER model, is *atomic* since the product is expressed in exactly one model. Similarly, the transformational method for converting an ER product into a relational product is atomic since each of the products is expressed in exactly one product model. On the other hand, as shown in Fig. 2, a compound method is composed out of other simpler methods. For example, the *constructional atomic* methods to construct the OM, the FM, and the DM products of OMT, compose the *constructional compound* method of OMT.

We view a method (Fig. 2) as a set of *method blocks*. When expressed in terms of method blocks, *compound* methods are composed of *method blocks*, which belong to several methods whereas all *method blocks* of an *atomic* method belong to the *atomic* method only. A method block is a pair, <objective, approach>. The objective of a method block tells us what the block tries to achieve. The approach tells us the technique that can be used to achieve the objective of the block. An objective itself is a pair <product type, process type>.

As shown in the figure, product types belong to a product model. For each product type of the product model, the process types that act upon it are associated with it to yield the objectives of the method. Method blocks do not exist independently of one another. Instead they exhibit dependencies among themselves. In (Pra97a) we have postulated four kinds of dependencies. Details can be found there.

3 The Information Systems Meta-model

The Information Systems domain aims to build teleological systems that mirror the real world. This domain deals with passive systems embedded in an environment. The environment sends a stimulus to the Information System, which responds to it. The domain is passive in the sense that it cannot perform an action without a stimulus. This makes the domain well suited to the development of transaction based systems.
In this section, we outline the method meta-model statics of (Pra97a). The meta-model is an instantiation of the generic view. A decision is instance of the method block, a purpose of objective, a purpose-approach (p-approach for brevity) of approach. Therefore, a method is a set of decisions and a decision is a pair, <purpose, p-approach>.

3.1 Meta-model Statics

There are two kinds of product types: conceptual structures and fixed structures. Similarly there are two kinds of process types, production manipulation which operates on the former and quality enforcement for manipulating the latter. These have been dealt with in (Pra97a) and (Pra99b) respectively.

As shown in Fig 3, conceptual structures can be partitioned into two clusters. The first cluster classifies them as either simple or complex. The second cluster partitions conceptual structures into disjoint classes of structures called constraint, definitional, constructional, link, and collection of concepts respectively.

A simple conceptual structure cannot be broken up into any components. It is atomic. Complex conceptual structures are constructed out of others which may themselves be complex or simple.

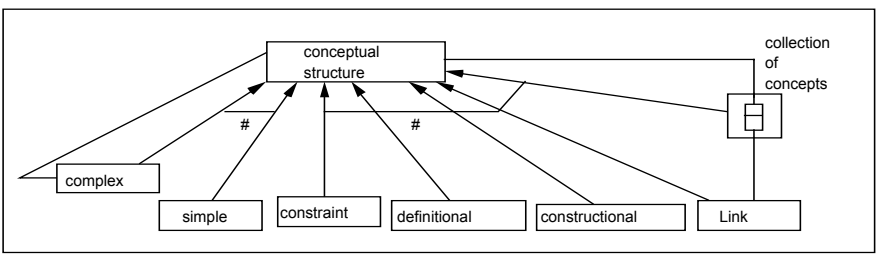

Fig. 3: The Conceptual Structure

There are five kinds of conceptual structures:

- Constructional: Constructional structures are used to represent the architecture of the product. For example, in order to display a conceptual schema expressed in ER terms, Chen uses the notions of entity and relationship respectively.
- Link: Product models use links extensively. For example, ISA links and aggregation links.
- Collection of Concepts: These are constructed whenever structures are connected by links. Aggregations, specialisation hierarchies, and subtype hierarchies are examples of such collections.

- Definitional: They define the properties of conceptual structures.
- Constraint: Constraints impose application-related integrity constraints on conceptual structures. For example, such a constraint could say that the ages of employees should be less than 65 years.

3.1.1 Fixed Structures

Fixed structures (Pra99a) are those which cannot be created or destroyed by application engineers. They identify the criteria that must be met by a good quality product. There are four kinds of fixed structures as shown in Fig. 4.

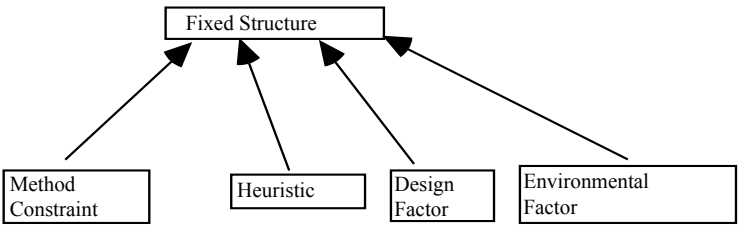

Fig. 4: The Fixed Structure

Environmental factors are fixed structures that specify the properties of the product as part of the environment in which it is embedded. These factors look after the various 'ilities' of software engineering, reliability, maintainability, usability, re-usability etc.

Design factors define method aims that are integral to method use. When translated into products, the resulting product is good because it meets method aims. For example, modularity is essential in object-oriented models and therefore is a design factor for such models. Heuristics are defined in methods to control the quality of the product. They provide rules of the thumb for good quality products and identify the bounds/criteria for acceptable products under the method.

Method constraints (Pra97a) deal with the restrictions that make conceptual structures well defined and well formed. There are four kinds of method constraints, completeness, conformity, fidelity, and consistency constraints.

3.1.2 The Operation

Operations identify the set of process types that operate upon product types to provide product manipulation and quality checking capability to application engineers. Operations are classified into two four classes (Fig. 5)

- Basic Life Cycle : for the creation and deletion of instance of structures
- Relational: for establishing relationships between concepts
- Integration: for building constructional/transformational products.
- Fixed Structure Enforcement: for quality enforcement

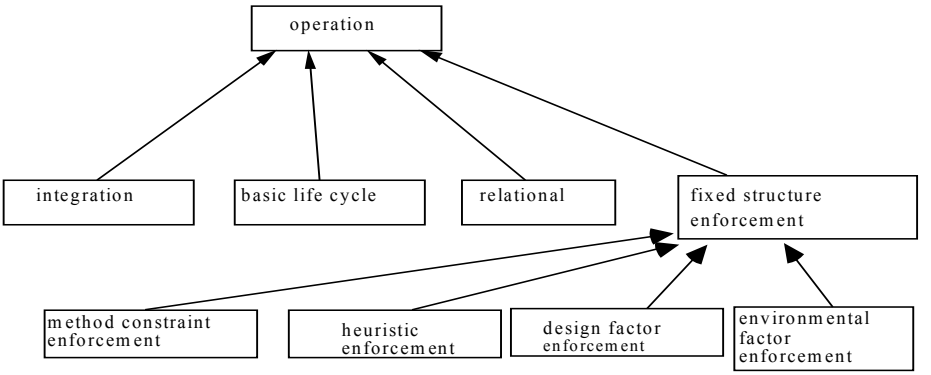

Fig. 5: The Operation

There are two operations in the basic life cycle class, create and delete. The relational class of operations is axiomatically defined and this definition can be found in (Pra97a)

3.1.3 Defining Decisions

A decision is an instance of a method block. In the meta-model, a decision has two components to it, the purpose and the p-approach. The purpose of the decision is a statement of what the decision wants to achieve. There can be many ways of achieving the purpose and this aspect is embodied in the p-approach. Thus,

Decision = < Purpose, P-approach >

A purpose is an instance of the objective of the generic model. It is a pair

Purpose = <structure, operation>

Thus, the following are purposes

<simple constructional structure, create>
<simple constructional structure, simple constructional structure, ISA link, relate>

3.2 Validating the Information System Meta-model

The validity of the foregoing meta-model has been established in (Gup01) and we only outline this here. Consider a method having the following concepts:

Cardinality, functionality, attributes, primary key, entity, relationship.

An instantiation of the meta-model is as follows

- Simple definitional concepts: cardinality, functionality
- Complex definitional concepts: attribute, primary key
- Simple constructional concepts: entity
- Complex constructional concept: relationship

A part of the total set of purposes is given below. The full set of purposes can be found in (Gup01).

<attribute, create>
<entity, create>
<relationship, create>
<entity, attribute, attach>
<entity, relationship, associate>

4 The Robotics Meta-model

In this section we consider building methods for the domain of Robotics. Unlike the Information Systems domain that we term as passive, we view the Robotics domain as active. Information systems are transaction-based and respond to stimuli from their environments. In contrast, robots perform specific tasks spontaneously and are programmed with the method to carry out these tasks. It is in this sense that the Robotics domain is active.

A robot can behave differently depending on the process that it is programmed to carry out. However, a robot addresses a class of tasks, and processes can be defined that fall into this class. Robots are similar to models like ER: just as a number of processes can be defined to build ER schemata so also a number of processes can be defined for a robot to carry out specific tasks. This is shown in the bottom row of Table 1. Just as it is possible to build meta-models that are an abstraction of the methods in the Information Systems domain, it is possible to build meta-models for robots that abstract their essential features into meta-models. This correspondence is shown in Table 1. Finally, the common properties of meta-models are abstracted out into the generic model of section 2.

Table 1. Correspondence between IS and Robotics Domains

Information Systems	Robotics
Generic Model	Generic Model
Information Systems Meta-models	Robotic Meta-models
IS Method like ER, SHM	Robot like Mitsubishi Industrial Robot

Our Robotics meta-model is presented in Figs. 6 to 7. In the rest of this section we show that this meta-model itself is an instantiation of the generic model and by doing so, we establish the genericity of the generic model. Thereafter, we instantiate the meta-model with the Mitsubishi Industrial Micro Robot System RV-IV (Mitsu) to show that the meta-model is valid.

The instantiation of the generic model is shown in Table 2.

Table 2. : The Instantiation of the Generic Method

Generic Concept	Robotics Meta-model Concept
Product type	Object
Process type	Action
Objective	Ability

The notion of an object is elaborated in Fig. 6. As shown, an object can be either an Operator or a Product. Operator refers to a component of a robot that has to be controlled, for example, its arm. Product is the object on which the operator acts. It is the real object which is to be manipulated by the operator, for example, a physical thing that has to be picked up, a screw that is to be tightened etc. Fig. 6 also shows that Object has attributes. Attributes capture the different properties that Operator/Product may have. For example, an operator which is the arm of a Robot may have as attributes the speed and direction of movement.

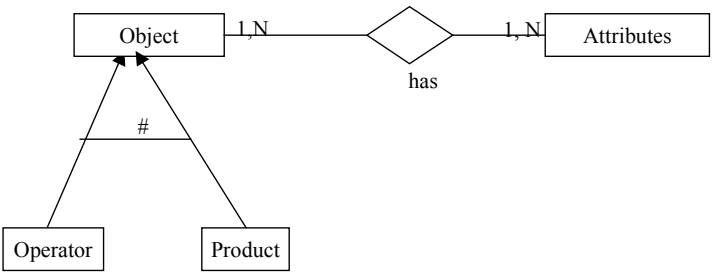

Fig. 6: The Object Meta-Concept

The notion of Action is shown in Fig. 7. There are two kinds of action, primitive and compound. Primitive actions cannot be decomposed into simpler ones whereas a compound action is built of other simpler actions. An example of an action is Move Robot Arm.

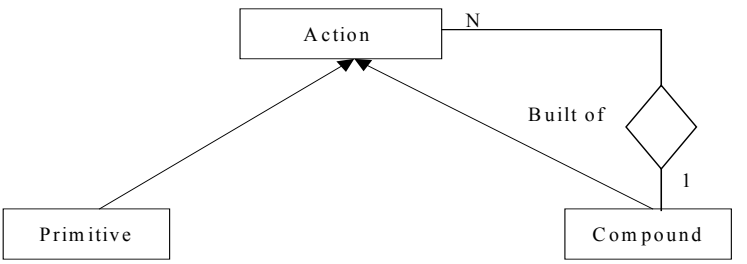

Fig. 7. The Action meta-concept

As shown in Table 2, an ability is an instantiation of objective. It is a pair, <object, action>.

4.1 Validating the Meta-model

The Mitsubishi Industrial Micro Robot System RV-IV robot can be integrated with a PC and can be used to pick and place physical objects. It is possible to control the robot through parameters like speed of movement and direction etc. The arm of this robot is an instance of Operator. It has the attributes speed, gripper_status, and pressure. Product is any item that has to be picked up and placed in some position by the robot. The instantiation of the Robotic meta-model is given in Table 3. We have

listed out only some of the primitive and compound actions of this robot. Full details can be found in (Mitsu).

Table 3. Instantiation of the Robotic Meta-model

Meta-model concepts	Concepts of Robot System
Operator	Robot arm
Product	Item (to be picked)
Attribute	Speed, gripper_status, pressure
Primitive Action	MP(Move Position), MS(Move Straight), MT(Move Tool), GC(Grip Close), GO(Grip Open), GP(Grip Pressure)
Compound Action	Movemaster program

The set of abilities defines the full instruction set of the robot and can be used to simulate the behaviour of the robot. In this sense the abilities are a specification of the functionality of the robot. A partial set of abilities of the robot is given below.

<Robot arm, MP>
<Robot arm, MS>
<Robot arm, GC>
<Robot arm, GO>
<Item, MT>
<Item, GP>

5 Project Planning Meta-model

To establish the genericity of the generic model, we now consider another passive domain, that of project planning and management. Project planning involves the completion of numerous activities (project components) by various resources-men materials, machines, money, and time –so that a project on paper is translated into concrete reality.

As for the other two domains, we will adopt the three-layer architecture for methods. At the highest level is the generic model. Below this is the meta-model that, in this case, will abstract out the common properties of project planning methods. At the lowest level, are the methods themselves that are instantiations of the project planning meta-model. The instantiation of the generic view with a project planning meta-model is shown in Table 4.

Table 4. The Instantiation of the Generic Method

Generic Concept	Project Planning Meta-model Concept
Product type	Project Concept
Process type	Operation
Objective	Capability

The Project Planning meta-model is shown in Fig. 8 and 9. There are two kinds of project concepts, components and constraints. Constraints are similar to method constraints and impose restrictions on well-formedness and well-definedness of the project plan. Components are concepts that are used in the representation of the project plan. These can be activities or events. Activities are long-duration actions that are triggered by events. The Figure shows that components have attributes. Thus, for example, an activity has a cost, a start time and an end time.

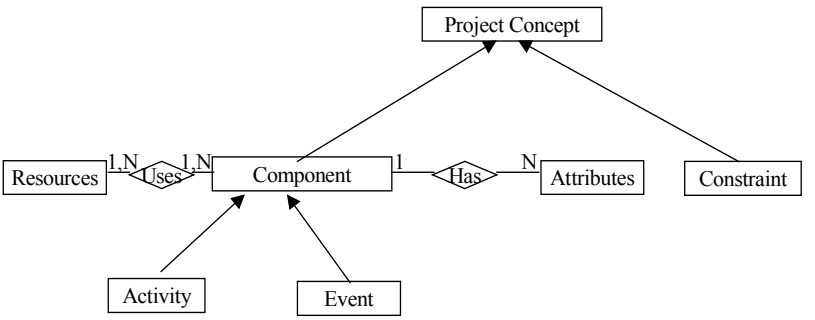

Fig. 8. The Project Concept of the Meta-model

The operations that can be performed on project concepts are shown in Fig. 9. There are two kinds of operations, those that enforce constraints and those for manipulating components. The latter are partitioned into basic life cycle and combinational classes of operations. Basic life cycle operations are used for creating/deleting components whereas combinational operations allow components to be related to one another. For example, the start event of an activity can be associated with it by a combinational operation.

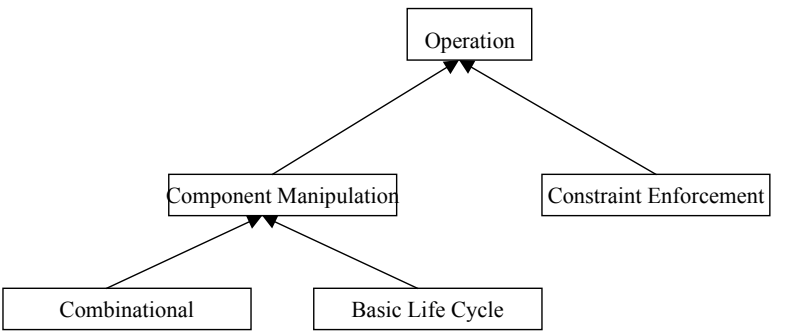

Fig. 9: Operations of the Project Planning Meta-Model

5.1 Validating the Meta-model

The activities of a project have inter-relationships arising from physical, technical, and other considerations. For proper planning, scheduling, and control of activities, network techniques have been found quite useful. We consider the construction of

such networks here. There are two basic network techniques: PERT and CPM. The orientation of PERT is 'probabilistic' and CPM is ' deterministic'. A project in both these can be broken into a well defined set of jobs or activities. The common characteristics of both these techniques are the imposition of an ordering on the activities that can be performed. The instantiation of the project planning meta-model for network construction is shown in Table 5.

Table 5. : Instantiation of the Project Planning Meta-model

Meta-model concepts	Project Planning Method Concepts
Constraint	1. Precedence/succession constraint: For every 2. action there is a preceding and succeeding 3. event. 4. No loops are allowed. 5. Not more than one activity can have the same 6. Proceeding and succeding events.
Activity	Activity
Event	Event
Attribute	Cost
Resource	Person
Combinational Operation	Attach attribute to project component Associate resource with project component Relate a start event and an end event with Activity
Basic Life Cycle Operation	Create event/activity Delete event/activity

Some of the capabilities of the project planning , method are as follows:

<activity, create>
<activity, delete>
<event, create>
<event, delete>
<person, create>
<activity, cost, attach>
<activity, person, associate>
<activity, event, event, relate>

6 Providing Computer Support

In this section we outline the manner in which computer assistance can be provided in the task of constructing methods of any domain. Our proposal is to raise the level of meta-CASE tools/CASE shells to generic CASE tools. These tools have the property that they can generate methods for diverse domains: Information Systems as well as others like the Robotics and Project Planning domains.

Method engineering has relied on the notion of meta-models to build CAME tools. Meta-CASE tools or CASE shells have a CAME tool as their front-end and a CASE generator as a back-end. Given that the generic view of section 2 has the capability to deal with a range of domains, we are in the process of designing Generic-CASE shells

with a generic part added in front of the CAME part. This part is used to instantiate the generic model, possibly, with the meta-model of the domain of interest. Thus, for example, the Generic tool can be used to instantiate the generic model with the Robotics meta-model of section 4. This capability can be used to develop generic-CASE that can produce methods and associated CASE tools of any domain.

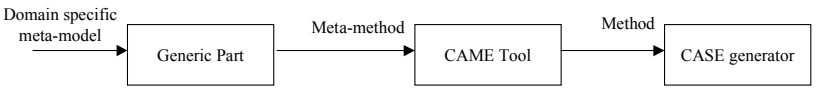

Fig. 10. The Three Stages of the Generic-CASE Tool

Fig. 10 shows that the Generic tool takes the specification of a domain specific meta-model as input and outputs the meta-method in terms of the set of abilities. This set is the definition of the full capability available with the meta-model and therefore with the meta-method. The meta-method is then given to the CAME tool from which the method is produced. Whereas the meta-method is domain-dependent, the method is dependent on the meta-method itself. The CASE generator produces a tool that makes the method usable.

The first step in the construction of the generic-CASE is a language in which meta-models can be defined. This language, the Generic Specification Language (GSL), expresses the meta-concepts and the generic concept to which they correspond. The interpretation system behind this language generates the set of objectives that form part of the meta-method.

7 Concluding Remarks

We have shown that the generic layer can help in instantiating meta-models of diverse domains. Of course, the generic layer can be used for different meta-models of a given domain as well. Thus, for the Information Systems domain, it can be used to instantiate process meta-models or integrated process-product meta-models. This genericity can be viewed as replacing meta-model dependence with generic model dependence.

Dependence of methods on meta-models implies that the features and limitations of meta-models get carried over to methods. In this sense, meta-model dependence constrains methods. The introduction of the generic layer and generic-CASE tools is an attempt at reducing the effect of this meta-model dependence. First, with generic-CASE tools, it shall be possible to build meta-models faster. This will encourage the development of meta-models that are tailor-made to produce the needed methods. Second, it shall be possible to take the output of the generic part of the generic-CASE, experiment with it and determine whether or not the meta-model provides the features that it was supposed to provide. Thus, we see the generic-CASE tool as providing a comprehensive structure for situational method engineering.

We shall observe similar effects in domains other than Information Systems. Thus, for example in the Robotics domain, the robot engineer has to first determine the nature of the robot to be produced and then construct the robot. Use of the generic-CASE tool here helps in a number of ways (a) it makes the robot engineer concentrate

on the task of defining robot characteristic and not its construction (b) it is possible to check out the usefulness of the robot meta-model before committing to a robot (c) once the functionality of a specific robot has been output by the tool, then it is possible to check out the robot before actually proceeding to construct it. Thus, the generic-CASE shall help in designing the needed robot.

We are now working on the generic-CASE tool itself. A draft generic specification language that will be input to the generic part of the tool has been designed and is under test. Thereafter, the actual task of generic-tool construction shall be undertaken. We shall use the meta-CASE MERU as the back-end of the generic-CASE.

References

(Cha95) Chandra P., Projects Planning, Analysis, Selection, Implementation and Review. Tata McGraw-Hill

(Gro97) Grosz G., *et al.* Modelling and Engineering the Requirements Engineering Process: An Overview of the NATURE approach. Requirement Engineering Journal, 2(3): 115-131

(Gup01) Gupta D., and Prakash N., Engineering Methods From Their Requirements Specification, Requirements Engineering Journal, 6, 3, 133-160

(Mitsu) Industrial Micro-Robot System model RV-M1 MovemasterEX manual, Mitsubishi

(Pra94) Prakash N., A Process View of Methodologies, in Advanced Information Systems Engineering, Wijers, Brinkkemper, and Wasserman(eds.), LNCS 811, Springer Verlag, 339-352

(Pra96a) Prakash N., and Sabharwal S., Building CASE tools for Methods Represented as Abstract Data Types, OOIS'96, Patel, Sun, and Patel (eds.), Springer, 357-369

(Pra96b) Prakash N., Domain Based Abstraction for Method Modelling, Ingénierie Des Systèmes d'Information, AFCET/HERMES 4(6), 745-767

(Pra97a) Prakash N., Towards a Formal Definition of Methods, Requirements Engineering Journal, Springer, 2, 1, 23-50

(Pra97b) Prakash N., and Sibal R., Computer Assisted Quality Engineering: A CASE for Building Quality Products, Proc. First Intl. Workshop on The Many Facets of Process Engineering, Gammarth, Tunisia, 25-35

(Pra99a) Prakash N., and Sibal R., Modelling Method Heuristics for Better Quality Products, Advanced Information Systems Engineering, Jarke M. and Oberweis A. (eds.), 429-433

(Pra99b) Prakash N., On Method Statics and Dynamics, Information Systems Journal, Vol. 24, No.8, 613-637

(Rol95) Rolland C., Souveyet C., and Moreno M., An Approach for Defining Ways of Working, Information System Journal, 20, 4, 337-359

(Sou91) Souveyet C.: Validation des Specifications Conceptuelles d'un Systeme d'information, Ph.D. Thesis, Universite de Paris VI