A PROCESS VIEW OF METHODOLOGIES

Naveen Prakash

Division of Computer Engineering, Delhi Institute of Technology
Kashmere Gate, Delhi 110006, India

email : np@dit.ernet.in

Abstract. It is argued that a methodology provides primitives using which the developer can construct the desired process. From the point of view of a process, a methodology is a passive device which only specifies a set of steps and step transition constraints. A step of a methodology can be modelled in terms of the triplet, <situation, decision, action>. A methodology is a collection of such triplets. The decisions of triplets articulate different kinds of methodology primitives like the creation, deletion, and modification of methodology-specific situations. Under this model, the development process becomes a decision making activity rooted in a methodology. The validation constraints of a methodology constrain this activity by regulating the transition from one decision to another.

I Introduction

For the last few years we have been trying to develop a model for the information system development process (Rol93a, Rol93b). In this work, we are viewing the development process as a decision making activity. Dowson(Dow87) classified development processes into three classes:
- the activity-oriented group,
- the product-oriented group,
- the decision-oriented group.

Our approach belongs to the decision-oriented group.

As argued in (Rol93a, Rol93b) we consider the process in contextual terms. A context is defined as a couple <situation, decision>. This suggests a strong coupling between the decision of a context and the situation in which this decision is taken. As a result, the process model answers the question, when is a decision taken? In addition, the model introduces the notion of macro and micro contexts. The former offer alternatives in decision-making whereas the latter cause product transformations.

It is our belief that the development process is rooted in a methodology. Because of this, the decisions and associated situations comprising a process are of two kinds
- methodology-specific,
- process-specific.

The methodology-specific aspect of a process is directly concerned with the construction of the product. As an example, consider the decision to create an entity of the ER schema. The process-specific aspect does not directly affect the product but

helps in building the process structure. An example of this is the case where the developer decides to backtrack to an earlier context of the process.

In this paper, we investigate the methodology-specific aspect and leave the process-specific aspect for another paper.

Traditionally, tools which support methodologies incorporate in them a certain technique which the developer is asked to follow. In order to make tools more responsive to developer preferences of process techniques, the need for a meta-level has been felt. These levels have been of two kinds : those that deal only with the product aspects of methodologies and those which deal with both, the product and process aspects. Smolander(Smo91) uses the Object-Property-Role-Relationship model for this meta-level. Rolland and Souveyet(Rol90) propose an object-oriented framework for this meta-level.

Chen and Nunamaker(Che89) use a semantic network to represent knowledge about process techniques used by developers. Brinkkemper(Bri90) proposes to separately model the product aspect and the process aspect of a modelling technique. For these he uses NIAM coupled with first order calculus for specifying validation rules. Wijers(Wij91) has tried to represent knowledge about development strategies and rules constraining allowable specification structures. The meta-level of Wijers is an 18-tuple consisting of a task structure, a concept structure, a set of verification rules, a set of procedures, a set of information places, a set of place labels, a set of decision rules, and of relations which relate these sets with one another. Verhoef(Ver93) uses the Predicate Set Model(PSM) of Hof(93) and LISA-D to build a formalism in which to express a methodology.

However, it seems possible to use other formalisms as well. For example, set theory(Ahi87) could also be used for modelling product aspects whereas Predicate-Transition Nets(Gen 87) and Entity Process Models(Hum 89) could be used for modeling the process aspects.

As far as we see it, the question of the meta-level is linked to the larger question of how one views a methodology. For Wijers, a methodology provides appropriate development structure by identifying useful strategies and techniques for specific problem classes. This, perhaps, explains the emphasis on the representation of knowledge about development strategies and rules constraining allowable specification structures. Brinkkemper(Bri90) looks upon a methodology as prescribing development in a hierarchy of ordered steps, which have to be performed in order to deliver the parts of a system in a proper manner.

According to us, a methodology does not identify any strategy of development. Rather, a methodology is a passive device: it provides a set of steps and step transition constraints which need to be enforced. If strategies exist then these are discovered by developers and they are enforced in practice through steps and step transitions.

To show this, we develop a methodology meta-level. This meta-level considers a methodology to be a set of triplets of the form

<situation, decision, action>

where

- the situation is either an elementary concept of the model (an entity or a relationship of the ER model) or a more complex substance built out of these elementary concepts (a dynamic transition of Remora).

- the decision expresses the intention of the action to be taken on the situation. For example, one may want to create an entity. In this case, the decision is 'create'.
- the action specifies the manner in which the decision shall be enforced. It is a procedure which lays down the acts to be performed.

There exists a triplet corresponding to each step that can be performed in a methodology. So by analysing a step, it is possible to identify the situation, decision, and action of the triplet corresponding to it. Let there be a step S and let its corresponding triplet be T. The situation part of T corresponds to the structure that is of interest in S, the decision part captures the intention of the step, and the action part of T specifies the way in which the decsion shall be realised.

The layout of the paper is as follows. In the next section we consider the properties of a methodology. Thereafter, we highlight our view of the relationship between a product, a process, and a methodology. Then, we introduce the notion of a triplet and consider its components in detail. We deal with the manner in which situations and decisions arise and also the way in which the set of steps of a methodology can be determined. Finally, in section IV, we consider step transition and show how a methodology imposes constraints on step transitions.

II What is a Methodology

The traditional view of methodologies is that they are based on a model and consist of a number of steps which must be executed in some order. Let us consider both these aspects of methodologies in turn.

The model of a methodology provides the basic building blocks in terms of which the product is expressed. In addition there are certain constraints which must be enforced. These have been expressed in four classes of validation controls in Remora (Rol88). These controls are introduced below together with examples taken from the Remora methodology

a) conformity controls which ensure that the norms imposed on the conceptual schema by the model are met. A rule of conformity control relates different concepts of the model together. For example, given the concepts of c-event, c-object, and c-operation the following conformity rule relates these together:
Every c-event constitutes the state change of exactly one c-object and triggers at least one c-operation. ---(1)

b) consistency controls which ensure that there are no contradictions in the description and also that it is not possible to introduce any contradictory expressions in it. For example consider
- If a c-object is modified by many operations in a given dynamic transition then all these operations must be conditionally triggered and these conditions must be pair-wise mutually exclusive. ---(2)

c) completeness controls which ensure that the conceptual schema respects all the rules of completeness in a representation. For example the following is a completeness rule:
Every c-object modified by a described c-operation and ascertained by a described c-event must be itself described in the conceptual schema.---(3)

d) fidelity controls which ensure that the conceptual schema introduces faithfully all the phenomena sought to be represented in it. For example, a fidelity rule can ask for the identification of dynamic cycles and subsequent examination for accepting or rejecting them.

Now, let us turn to the second aspect of methodologies, that of steps. In specifying the set of steps, it is usual to explicitly specify only those steps which deal with the *creation* of the concepts of the model and in *amplifying* their properties. For example, methodologies do not have steps to retract previously created concepts, to transform, say, an entity into a relationship, or to do constraint checking. Finally, the question

assuming that a transition is to be made, what are valid transitions?

is also left implicitly answered.

We look upon a methodology as a quadruple

< M, MC, S, ST >

where M is the model in terms of whose concepts the IS product is to be expressed, MC is the set of model constraints, S is the set of steps that are available in the methodology, and ST is the set of step transitions. For example, we could define the ER based methodology above as having

- M, the ER model
- MC, the constraints of the ER model
- S, the set of steps as follows:

{ Define entitites, Define relationships, Define weak entities and relationships, Define keys, Define cardinalities of relationships, Define roles, Define value sets, Define attributes, Define functionality of attributes }

- ST, the step transitions, which specifies the possible step transitions. Some elements of this set are the transitions from

define entities	to	define relationships
define entities	to	define attributes
define relationships	to	define cardinalities of relationships

We show later the exact manner in which ST can be determined.

III A Methodology as a Triplet

We consider a step of a methodology to be a triplet

< s, d, a >

where s is the situation, d is the decision and a is the action. When a step is performed, then an instance of the type, s may be created, deleted, modified, or corrected, and, additionally, it gets reflected in the product under development.

A *situation* is the element which is the object of the step being modelled. A *decision* reflects the intention of the step. It specifies the objective that the step can achieve with regard to the situation. An *action* specifies the executive action that must be performed in order to give effect to the decision. It performs a transformation on the product. Performing it changes the product and may generate new/modified instances of situations which, in turn, are subjects of steps.

As an example, consider the step, 'define subtype of an entity'. This can be represented as < entity, subtype, A >. Here, the object of the step, the situation, is 'entity'; the intention of the step, the decision, is to 'subtype' the entity; and the action, A, which enforces this decision, consists of defining the subtype structure and also of the

modification of relationship structures entered into by the generic entity type. As a result of the execution of < entity, subtype, A > an instance, e of type entity, gets affected and a new situation results. This new situation can be acted upon through other steps of the methodology.

The Situation

We define a situation as follows:
A situation is either an elementary concept of the model or a more complex substance built out of these elementary concepts
The terms, concept and complex substance, used in this definition need further clarification. We need to identify what concepts and complex substances are and how do they arise. Let us consider the notion of a concept first.

It has been shown that one can develop a meta-model of the model, M, of a methodology(Rol90). Such a meta-model gives us a precise definition of what a concept is. We partition the set of concepts of M into two classes of elements, structural and definitional elements. Structural elements of M identify the structures that are permitted under it whereas the definitional elements identify the amplifications that are required to be made to define the structural elements properly.

There are two kinds of structural elements, primitive and dependent. Primitive structural elements are those which are not dependent upon other elements for their existence whereas dependent structures need other structural elements to exist. For example, the ER model can be considered to specify two structures, entity and relationship. An entity is a primitive structural element whereas the relationship is a dependent one since it is dependent on entities for its existence.

As one continues building structural elements from one another, a time comes when one is left with a set of structural components which themselves are not components of any other structure. It is possible for two or more such structural elements to be mutually dependent upon one another. Such structural elements shall be referred to as peer elements and they shall be dealt with in detail when the conformity constraints are considered. As an example of peer elements in Remora, consider the c-events and c-operations triggered by it. These are mutually dependent upon each other : it is not possible for a c-event to be defined without any c-operations. Conversely, it is not possible to have c-operations which are not associated with a c-event.

Definitional elements of a conceptual model are associated with its structural elements. Just as there are two kinds of structural elements, there are two kinds of definitional elements, primitive and dependent. Primitive definitional elements are those which do not require any other definitional element for them to be well defined. Dependent definitional elements require other definitional elements in order to be well defined. For example, with the structural elements, entity and relationship of the ER model, we have 'attribute' as a definitional element. Additionally, the properties of these attributes, whether these are mono or multi-valued, must be known. The nature of functionality of attributes is the definitional element associated with attributes. Thus, we have attributes and the nature of their functionality as the definitional elements of the ER approach. The former are dependent definitional elements whereas 'functionality' of attributes is a primitive definitional element.

Now, let us consider complex substances. A complex substance is a collection of concepts which has meaning in a methodology. A collection of concepts carries meaning if either one of the following holds
- the collection is a well defined part of the product, can be identified as such, and can be the object of a decision. As an example consider the notion of the dynamic transition of Remora.
- the collection of concepts are subjects of one or more constraints of the model. For example, consider the following consistency constraint of Remora
If the c-events triggering multiple c-operations modifying the same c-object are independent of one another or if they are chronologically dependent on one another then the triggers of these c-operations must be pairwise mutually exclusive.
Here we have two complex substances
a) c-events triggering multiple c-operations modifying the same c-object but independent of one another
b) c-events triggering multiple c-operations modifying the same c-object but chronologically dependent on one another

It must be noted that the collection of objects forming a situation is structural in nature. Further, this collection is a dependent structural substance.

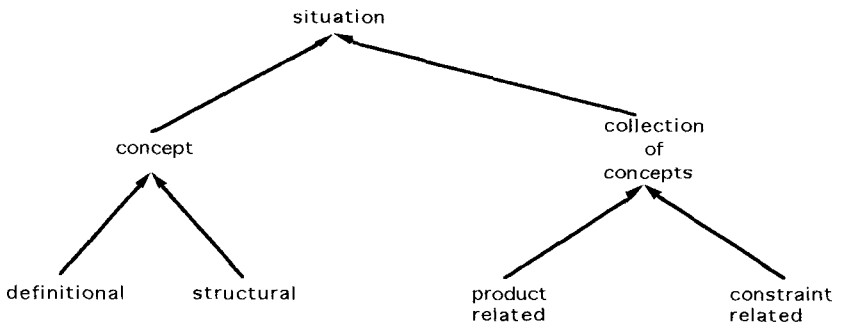

Fig 1 : The situation

The notion of a situation is illustrated in Fig. 1 We can phrase this view of the situation as follows:

A situation is
- either a structural or a definitional concept of the model
- a collection of concepts which either holds individual interest as a part/whole of the product or which is of interest in constraint enforcement.

Notice that this definition considers the entire product, that is, the product in a certain state of development, as a situation.

The Decision

The decision component of the triplet identifies the intention of a step of the methodology. Since it is an intention to create new situations in a methododlogy, we have the create class of decisions. Anything that can be created can be modified or deleted. Thus, we also have these two additional classes of decisions. Under the modification class, it is possible to group together all those decisions which cause a change in a situation that has already been defined. For example, modification could include in it, the decision sub-classes as follows:

- update : intent to update a situation. For example, update a dynamic transition by changing the trigger of an operation.
- historise : intent to keep the history of a situation. For example, keep a history of the orders delivered.
- subtype : intent to specialise a situation.
- retype : intent to change the form of a situation. For example, the change of a relationship into an attribute or the change of an relationship into an entity.

In addition, decisions related to validation controls of a methodology have to be dealt with. There are two ways of defining validation controls, depending upon the way one interprets the intention of these controls. The first of these looks upon a control as a checking device : the intention of a control is to check that a certain condition is indeed satisfied in a situation. Thus, we have a class of decisions, called check, which check the validity of a situation and whose action part returns the boolean values, TRUE or FALSE. The second way of interpreting validity controls is to look upon them as making a positive, active contribution to the situation. In this view, the intention of a control is to build a "correct" situation. Thus, we have the class of decisions, called correct, whose intention is to correct a given situation, if there are any errors in it. If not, then the intention of the decision is vacuously satisfied. The action part of the correct class of decisions returns a correct situation.

We prefer to take the second of these two options. This is because we believe that when a validity control is applied then knowledge about the correctness of the situation is desired. However, this knowledge is in a wider context : the developer desires to correct the flaws, if any, in the situation. Thus, it makes sense to accept the class of decisions, correct.

The decision classes can be organised as shown in Fig. 3. The entire set of decisions can be subdivided into the create, delete, modify, and correct classes. The modify class has under it the classes specialise, update, historise, and retype. The correctness class of decisions is subdivided into four groups, one group corresponding to each of the different kinds of validation controls. They are named according to the kind of validation control they deal with. Each class has its own instances. For example the create class has instances, create entity, create relationship etc. Similarly, the conformity class has an instance for each conformity constraint defined on the model.

Decisions may be atomic or complex. An atomic decision is one which cannot be broken up into simpler decisions. A complex decision is constructed out of other decisions which may be atomic or complex.

The designer of a methodology has the choice to define a certain decision as atomic or complex. If the exact sequence of manipulations to be performed in order to give effect to a decision, D, is pre-determined, then it is possible to define the action of D completely. In this case, D is atomic : it is implemented in its action and is not built over any other decision. On the other hand, it is possible that the exact sequence of manipulations to give effect to D is not known or that more than one alternative exists. In both these cases, the designer of the methodology may choose not to commit to any specific procedure of implementation of D. The alternative is to consider D as composed of a number of decisions A, B, C, ... such that the intention of D is realised through the intentions of its components. Then, it is clear that D is a complex decision, built out of the component decisions, A, B, C,.....

Consider an example. Let there be users who borrow books. Let it be required to specialise users into borrowers and non-borrowers. It is possible to consider this as a complex decision built out of

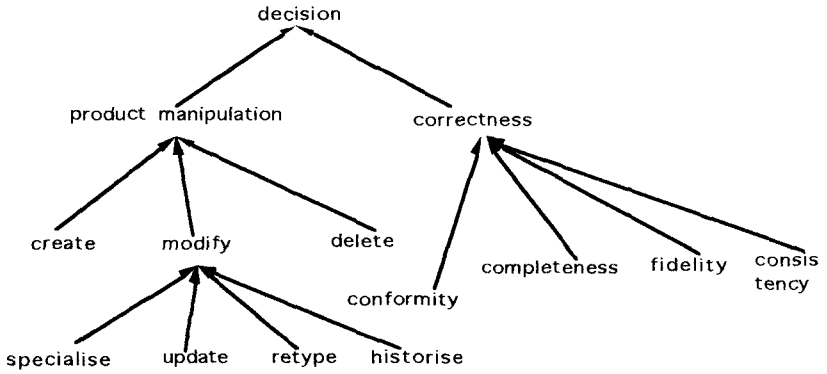

Fig. 2 : Class Hierarchy of Decisions

create entity	{create borrower}
create borrower as a subtype	{subtype borrower of user}
create entity	{create non-borrower}
create non-borrower as subtype	{subtype non-borrower of user}
define partition constraint between entities	{correct situation}
remove role	{remove the role user borrows}
create role	{define role borrower-borrows}

On the other hand, it is possible to treat the foregoing as an atomic decision. It can be argued that the action of specialise is the following procedure

- create entity borrower
- make it a subtype of user
- create entity non-borrower
- make it a subtype of user
- define the partition constraint between borrower and non- borrower
- remove the role of user as borrows
- create the role borrows of borrower

This sequence of operations is frozen into the action of specialise. Consequently there is no need to consider specialise as built over other decisions : it becomes atomic.

The essential difference between a complex decision and an atomic one is that in the former, the methodology designer gives freedom to the process developer to define his/her own process. As a result, the decision is considered to be composed of a set of component ones. The developer has only to choose the particular order of these to give effect to the complex decision. In an atomic decision, on the other hand, no such freedom is given.

The Action

The action component of the triplet makes explicit the enforcement mechanism of a decision. It provides a procedure to be followed for this purpose.

To be well defined, the action must encode in it

1. Knowledge about the procedure to be followed to enforce the decision.
2. Knowledge about the part of the product that can potentially get affected.
3. Knowledge about the conditions that must be met in order for the procedure of (1) above to be carried out.

Consider property (1) above. For an atomic decision, an action can be defined provided the exact sequence of acts needed to give effect to it is known. The definition of an action is committed to by the designer of a methodology. A methodology designer leaves the definition of the action associated with a complex decision to be made by the developer. The developer is provided knowledge of the component decisions and, through the activity of decision making, selects one decision after another from among these. In this way, the action of the complex decision is determined through the actions associated with its component decisions.

It is to be noticed that development of the action of a complex decision amounts to the definition of a process. However, unlike that for an atomic decision, this process is defined by the users of a methodology.

Now, let us consider the second property above, that which relates to the part of the product affected by an action. We formalise this using the notion of an environment(Pra92).

The static environment of a structural element, E, consists of
- E itself,
- the dependent elements of E, and
- the elements which have E as their dependent element.

A situation is capable of demonstrating two kinds of dependencies
- direct structural involvement wherein it is structured out of other situations. For example, the situation, a relationship, shows direct involvement with the situation, an entity.
- peer structural involvement wherein the situation is structured out of other peer elements. For example, the dynamic transition of Remora is structured out of peer situations, c-event, c-object, and c-operation.

The effect of an action can be felt over the static environment of the situation it acts upon. The action to delete an entity, shall cause the entity to be deleted but this shall have an impact on, for example, the relationships it participates in. In this sense, the scope of an action is the static environment of the situation it acts upon.

An action has two main parts
- the condition,
- the body.

The condition part of the action specifies the pre-conditions that must hold for the body to be executed. The body specifies the procedure to be performed. If the pre-conditions do not hold then the decision is not enforceable since the body shall not be executed. A decision is considered as *effective* when the body of its action has been executed. It is said to be *ineffective* otherwise.

III.1 Defining Steps

We are now in a position to consider the manner in which steps can be represented as triplets. Steps of a methodology that create, modify, or delete situations can be

represented as triplets in a straight forward manner. This has been illustrated in the foregoing with the step, create entity. However, the expression of steps dealing with validity controls is not obvious. We deal with this here.

1. Completeness Constraints

Completeness constraints can be expressed in terms of the notion of a situation introduced earlier as follows:

- Every primitive definitional element that has been been assigned values from a domain specified by the conceptual model is complete. Consider the functionality of attributes as a primitive definitional element which take values from the domain { mono-valued, multi-valued}. Then the assignment of a value from this domain defines the functionality completely.
- Every dependent definitional element is complete if all its base definitional elements are complete. Let attribute be a dependent defintional element dependent upon the primitive element, functionality. Then attribute is complete when functionality is complete.
- Every primitive structural element is complete if all its definitional elements are complete. Let there be a primitive structural element called entity which has definitional elements key and attribute respectively. Then entity is complete if key and attribute are complete.
- Every dependent structural element is complete if all its base structures are complete and if all its definitional elements are complete. Let there be a dependent structural element called relationship which is dependent upon the structural element, entity, and which has the definitional elements, cardinality, and attribute. Then relationship is complete if entity, cardinality, and attribute are all complete.
- A complex situation is complete if all its structural elements are complete. Let there be a product consisting of two entities, Employee and Department, and of a relationship, Employed, between them. This product is complete provided Employee, Department, and Employed are all complete.

The decision class relevent to the completeness constraint(see Fig. 3) is the completeness class. We define a triplet for each of the rules of completeness formulated above. The triplets corresponding to these rules are as follows:
- <primitive definitional element, complete1, action>
- <dependent definitional element, complete2, action>
- <primitive structural element, complete3, action>
- <dependent structural element, complete4, action>
- <complex situation, complete5, action>

The issue of enforcement of completeness then reduces to the selection of the appropriate triplet at an appropriate time in the development process.

2. Conformity Constraints

Conformity constraints specify constraints that must hold across peer elements. As for completeness constraint these can be represented in situational terms. For example, the constraint (1) of section II can be represented as a ternary relationship, c-ev form, between c-event, c-object, and c-operation repectively. Alternatively, it can be represented as a system of two binary relationships: the relationship, state change relating c-event and c-object, and the relationship, trigger, relating c-event and c-operation, respectively. These two possibilities are shown in Fig. 4.

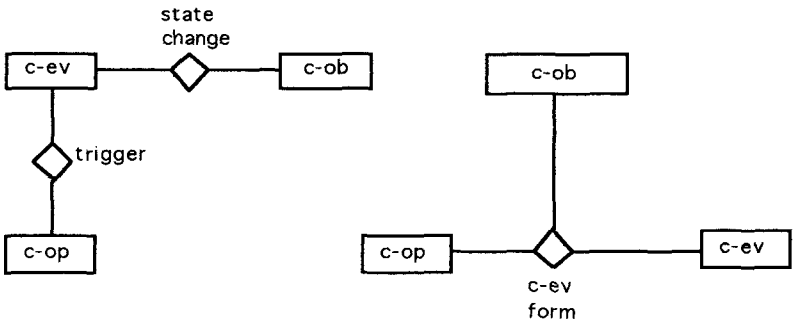

Fig. 3 : Two Forms of the Conformity Constraint

The triplet corresponding to the constraint (1) when viewed as a binary relationship is

<c-ev form, create, action> ------(1a)

where create(c-ev form) is considered as an atomic decision.

When viewed as a system of two relationships, we have

<c-ev form, create, action>: ------(1b)

<state change, create1, action>

<trigger, create2, action>

Here, create(c-ev form) is considered as a complex decision built over create1(state change) and create2(trigger).

3. Consistency Constraints

These constraints have two parts to them. The first part specifies the situation over which the property specified in the second part holds. For example in constraint 2 of section II the conditional part identifies the situation, s1, in which the property of mutually exclusive triggers must hold. The enforcement of this can be expressed as a triplet as follows

<s1, consistency, action>. ------(2a)

It is possible to treat consistency as an atomic decision if the procedure for enforcement is contained in the action. If not, then it is a complex decision

<s1, consistency, action>. ------(2b)

< mut ex trigger, create, action>

where mut ex trigger is the situation, mutually exclusive trigger.

4. Fidelity Constraints

Fidelity constraints are defined to ensure that the developed product reflects the real world faithfully. The validity of these situations cannot, quite often, be determined automatically and real world knowledge is required to do so. However, a methodology highlights these special situations. As an example, consider isolated nodes of the static graph of Remora. It is not clear whether the existence of such nodes is good or bad. The developer must examine these and accept/reject them. One can represent fidelity constraints as triplets by identifying the situation whose fidelity is to be enforced. For example, the isolated node constraint can be expressed as

< node, fidelity, action>.

IV. Transition Between Decisions

In conformity with our position that a methodology does not, in fact, impose a process on the developer but only offers a set of primitives, we postulate that any step of a methodology can be invoked at any time. In triplet terms, this means that any decision can be taken at will. Thus, a developer may decide to create a relationship before the entities participating in it are created. The application of the completeness constraint on the relationship can thereafter result in the creation of the entities. Alternatively, the developer may first create the entities and then the relationship. A methodology cannot impose either of the two strategies on the developer.

A methodology does not however, provide unlimited freedom to the developer in decision making. In a most elementary sense, there is a dependency between all decisions of the decision classes modify and delete upon the create class of decisions. Thus, nothing can be modified and/or deleted unless it has been first created. However, decisions exhibit more intricate kinds of dependency among themselves. These dependencies are on account of the constraints specified in the model.

Constraints say that after a decision has been taken some other decisions must be taken. However, when these decisions are taken is not specified. In other words, the place in the decision making activity where the choice of these decisions is made is a matter of the strategy adopted.

The foregoing notion of decision dependency can be formalised using the dependency relationship as follows

A -----> B

This says that the decision B is dependent upon the decision A. Here, B need not be distinct from A.

Decisions exhibit transitive dependency among them. Thus, if we have

product manipulation decisions -----> correct decisions and
correct decisions -----> product manipulation decisions

then product manipulation decisions are transitively dependent upon themselves. More informally, the creation/modification/deletion of a situation asks for decisions to manipulate the resulting situations. As an example consider the decision to create an entity class. The constraint decisions dependent upon these are

create(entity) -----> completeness decision
-----> conformity decision

Now, the completeness decision of a methodology could specify that

complete(entity) -----> create attributes
-----> create key

This, by transitivity, implies that

create(entity) -----> create attributes
-----> create key

Now, let us consider the manner in which product manipulation decisions depend upon constraint decisions.

a. Dependency due to Completeness Decisions

The rules of completeness defined in section III.1 can be used to construct the dependent decisions of a decision. Consider the rule

Every dependent structural element is complete if all its base structural elements are complete and if all its definitional elements are complete.
This implies that the creation of base as well as definitional elements is dependent upon the creation of the dependent structural element.

Applying this to entities of an ER based methodology with specialisation, we have

create(entity) -----> create attributes
-----> create key

If the foregoing approach to identifying dependent decisions is applied to each rule of completeness of section III.1, then we shall obtain the entire set of dependent decisions of a methodology arising out of completeness constraints.

b. Dependency due to Conformity Decisions

Dependency of product manipulation decisions on conformity decisions arises when the conformity decision is considered to be a complex decision. A determination of the dependencies of product manipulation decisions on conformity can be done by a systematic identification of all complex conformity decisions. There is a dependency of the component decisions on the conformity decision.

For example, consider (1b) above. The implied dependencies are

create(c-ev form) -----> create1(state change)
-----> create2(trigger)

c. Dependency due to Consistency Decisions

In a manner similar to that for conformity decisions, a determination of the dependencies of product manipulation decisions on consistency can be done by a systematic identification of all complex consistency decisions. There is a dependency of the component decisions on the consistency decision. To see this consider (2b) above The following holds

consistency(s1) -----> create(mut ex trigger)

d. Dependency due to Fidelity Decisions

Fidelity triplets have been defined with the purpose of bringing special situations to the attention of the developer. As such, it is recognised that suppport for these in the framework of a methodology cannot be provided. The developer is expected to identify the strategies for handling these situations.

In this sense, the methodology does not identify any decisions specifically dependent on fidelity constraints

V. Conclusion

The prevalent view of methodologies is that the process of development is a part of a methodology. Instead, we have shown that a methodology provides a set of primitives. These primitives, when suitably augmented with process-specific primitives help in building the process. The exact manner in which the process is constructed is the way of working of this developer. Thus, the way of working, or the strategies actually followed in development are external to a methodology.

It has long been felt that the product and the process aspects of development are linked to each other. As we see it, the linkage lies in the fact that a triplet specifies a decision to be taken on a situation. When an entire process is built up using process and methodology primitives, then this linkage between the product and the decision gets carried over to it. The decision making activity, in so far as it causes decision selection, establishes the link between the product and the process.

References

(Ahi87) Ahituv N, A metamodel of information flow: a tool to support information systems theory, CACM, 30,9, 781-791

(Bri90) Brinkkemper S, Formalisation of information systems modelling, Ph.D. thesis, University of Nijmegen, Thesis Publishers, Amsterdam

(Che89) Chen M and Nunamaker JF, MetaPlex: An integrated environment for organisation and information systems development, Proc. tenth Int. Conf. On Info. Syst., Boston, Mass

(Dow87) Dowson M, Iteration in the software process, Proc. Ninth Intl. Conference on Software Engineering, Monterey, California.

(Gen87) Genrich H, Predicate/transition nets, in Petri Nets Central models and their properties, Brauer et al (eds.), LNCS, 1254, 207-247

(Hof93) Hofstede AHM ter and Weide Th P van der, Expressiveness in conceptual data modelling, Data and Knowledge Engineering, 10(1), 65-100

(Hum89) Humphrey WS and Kellner MI, Sotware process modelling:principles of entity process models, in Proc. eleventh Int. Conf. on software engineering, IEEE, Pittsburgh

(Pra92) Prakash N, An object oriented methodology for information systems design, in Information systems concepts:improving the understanding, Falkenberg et al(eds.), 53-86, North Holland

(Rol 88) Rolland C et al, Conception des systems d'information, Eyrolles, Paris

(Rol90) Rolland C and Souveyet C, An object-oriented framework for information systems, in Data Management, Prakash N (ed.), 280-303, Tata McGraw Hill, N. Delhi

(Rol 93a) Rolland C and Prakash N, Reusable Process Chunks, Proc. DEXA, LNCS vol720, 655-666

(Rol 93b) Rolland C and Prakash N, Modeling decisions in the requirement engineering process, Proc. CISMOD, New Delhi, 229-242

(Smo91) Smolander K et al, MetaEdit: A flexible graphical environment for methodology modelling, in Andersen R et al(eds.), Proc. CAiSE, Trondheim, LNCS vol 498, 168-193

(Ver93) Verhoef TF, Effective information modelling support, Ph.D. Thesis, Delft University of Technology, 1993

(Wij91) Wijers GM, Modelling support in information systems development, Ph.D. Thesis, Thesis Publishers, Amsterdam