22

Software Process Model Specification*

M. Baldi and M. L. Jaccheri
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24, 10129 Torino, Italy

Abstract

The E^3 process model specification method is presented. It consists of a process modeling language based on classes, associations, and inheritance with a graphic representation and a presentation technique based on views and paths. The specification of a real process model is given as example.

Keyword Codes: D.2.0, D.2.2, D.2.6
Keywords: Software Engineering, Tools and Techniques, Programming Environments

1. INTRODUCTION

A software process is the set of real world entities that contribute to the production and maintenance of software systems. A software process model[1] is the description of a software process. A part of a process model is referred to as a process model fragment. A *software process model template* provides a common description for a class of process models [1-2].

E^3 is an academic project that has the goal to provide a methodology to specify, design, execute, and evolve software process models[2]. The E^3 methodology is a meta-process for the production of process models that consists of three meta-phases: specification, design, and implementation. Support for both evolution and reuse is provided in each phase. Each phase exploits a well defined set of object oriented techniques extended with association management. Among these techniques and conventions, some are directly adopted from previous object oriented work, e.g., [3-5], while others have been self developed for specific process modeling support.

This work is about software process model template specification. The E^3 specification method has been used to specify the software process model of FIAT Iveco, an Italian organization. An excerpt of this work is presented as example.

The structure of this work is as follows: section 2 introduces the E^3 specification method by explaining the specification language and a presentation technique. An example is presented in section 3. Conclusions are given in section 4.

2. THE E^3 SPECIFICATION METHOD

The E^3 method for template specification offers:

---

* This work has been partially supported by MURST 40% project "Metodologie e Strumenti di V&V di Funzionalità, Prestazioni e Affidabilità di Processi e Prodotti nell'Ingegneria del Software" and by EERP project IT-023.

[1] The prefix "software" may be omitted in the following.

[2] E^3: Environment for Experimenting and Evolving software processes.

M. Lee et al. (eds.), *Software Quality and Productivity*

The E^3 method for template specification offers:

- an object oriented language with graphic representation. The PML offers a set of predefined classes and associations;
- a presentation technique based on views and paths.

The graphic representation gives support for understanding, communicating, browsing, and editing process model fragments as it is difficult to conceptually grasp the nature of the process models in textual forms. The presentation technique allows process specifications to be kept readable independently of their scale.

An E^3 specification consists of a set of classes and associations denoting a process model template. A specification classifies process model elements into sub-classes of predefined classes and poses constraints by means of associations. For example, there are predefined associations to express synchronization constraints among tasks and others to express data flow between tasks. User defined classes and associations are constructed by specialization from the predefined ones.

2.1. Modeling levels

The following three modeling levels are needed:

model level represents a given process model. A process model is represented as an object net, i.e., a set of objects and links.

template level expresses process model templates. A class is a template for a set of objects, while an association is a template for a set of links. Moreover, a cluster (or aggregate) of classes and associations is a template for a set of nets, i.e., process model fragments. Classes and associations are organized in inheritance hierarchies. The template level enables reuse and polymorphism.

meta-model level represents classes of templates. Each class is also an object of class meta-class, as in Smalltalk [5] and each association is also a link of association meta-association. Meta-classes and meta-associations are implicitly declared at the time the respective classes and associations are defined.

2.2. The specification language

The E^3 language for process model template specification is based on the framework depicted in Figure 1. Predefined classes and associations are organized in inheritance hierarchies. The E^3 specification language permits both formal descriptions (by means of classes, associations, and inheritance), and not formal descriptions (by means of comments).

Predefined classes are abstract (i.e., they are never instantiated) and are used to define new classes by means of inheritance; each class must be derived from a predefined class. Classes and associations within a given model must respect the following constraints.

Class inheritance Classes are organized in an inheritance hierarchy whose root is `Object`. Usual semantics of class inheritance is that a class inherits all the attributes and methods of its super-class and in case overrides them. Here, we extend this inheritance definition by saying that a class inherits also the associations to which its super-class participates.

Association inheritance Associations are organized in an inheritance hierarchy rooted by association. The following rule holds for association inheritance.

$$\text{if } R(A_1,\ldots, A_n) \wedge S(B_1,\ldots, B_n) \wedge Inh(S, R) \Rightarrow (m \leq n) \wedge (\forall i,\ 1 \leq i \leq m,\ B_i = A_i \vee Inh(B_i, A_i)).$$

Where R(A,B) denotes an association R defined between classes A and B; Inh(X,Y) denotes that X, being it either a class or an association, inherits from Y. Each class participating to the sub-association S, still participates to the super-association R.

Association overriding An association can be overridden, provided that each class participating to the overridden sub-association, does not participate to the original association. The following rule must be respected:

if $R(A_1,\ldots, A_n) \wedge S(B_1,\ldots, B_n) \Rightarrow \exists j, Inh(B_j, A_j) \wedge (\forall i, 1 \leq i \leq n, i \neq j, B_i = A_i \vee Inh(B_i, A_i))$.

These constraints impact specification of process models, reuse of process model fragments and limit change.

The graphic representation provides a different icon for each predefined class; user defined classes are depicted by means of their own predefined superclass icon. According to the graphical convention introduced in Figure 1(a), each predefined association is uniquely identified by its graphical representation (i.e., it is not necessary to write its name). Even though the same symbol is employed to depict more than one association, the connected classes give context for symbol interpretation.

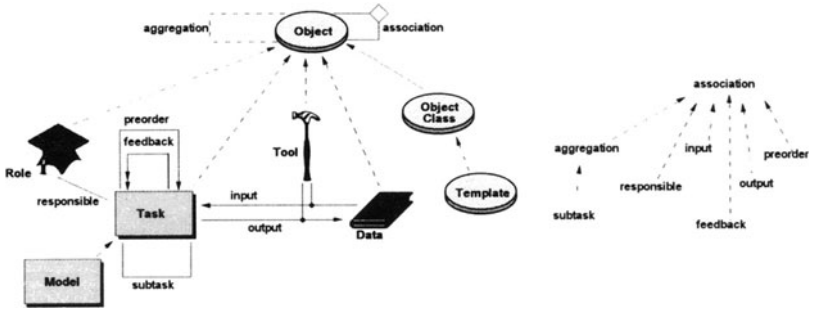

Figure 1. (a) Predefined classes and associations (b) Inheritance hierarchy of predefined associations.

In the following we give the meaning of each predefined class and association.

Object denotes whichever process model element. It is the inheritance root for each class and as each other class it is also an instance of class `ObjectClass`.

association both declares a class of links and constitutes itself a link (arity N). The inheritance hierarchy (rooted by `association`) for associations is displayed in Figure 1(b).

aggregation relates the whole (aggregate) and its parts. When an aggregate is destroyed, its parts are also destroyed, should they not participate in another `aggregation`. It inherits from `association` and its roles[3] are `aggregate` and `parts`.

Task denotes a software development activity. It starts execution, terminates either with success or with failure. In case, it re-starts execution. A software development activity, e.g., design, is represented as a `Task` sub-class.

subtask inherits from `aggregation`. Role names are `father` and `children`[4] . When a `Task` sub-class is connected by `subtask` to a set of children `Task` sub-classes, then father execution is the composition of children executions. Further, when the father is re-started all its children are re-

[3] Here, "role" denotes the role played by a class in the context of an association.

[4] Father and children must not be confused with inheritance role names, as sometimes used in object orientation literature.

started as well. Association `subtask` gives a tree structure, here called *subtask tree*, to a template. Each node of this tree is a `Task` sub-class and each edge is a `subtask` association. In the context of a given `Task` sub-class, association `subtask` declares a synchronization scope, enabling the definition of dependency constraints among sibling activities.

preorder role names are `predecessors` and `successors`. A task must start its execution when all its `predecessors` have terminated.

feedback role names are `giving_feedback` and `waiting_feedback`. A task `T` restarts when a task connected by feedback in the role `giving_feedback` terminates with failure and all `T`'s predecessors are terminated. While `preorder` can exist without `feedback`, `feedback` semantics depends on `preorder` semantics.

Role[5] defines a responsibility. Each sub-class represents a set of skills for a performer.

responsible connects a `Task` sub-class to the `Role` sub-class modeling the role of the activity responsible.

Data has a content that can be read and written. `Data` sub-classes are templates for software process products.

Tool sub-classes model automatic tools or techniques.

input and **output** express data flow among activities and define the manipulating tool.

Model is a particular `Task` sub-class modeling an entire process model, e.g., `Iveco_model` is the model that will be used as an example throughout this paper. `Each` subtask tree is rooted by a `Model` sub-class.

ObjectClass is a meta-class. It is the super-class of each meta-class. When a class is declared, its meta-class is implicitly declared.

Template is `Model` class. A process model template, i.e., a `Template` instance, is a `Model` sub-class, say `M`. It consists of a set of classes that can be calculated as the sum of: (1) all the `Task` classes in the subtask tree with root `M`; (2) for each of the `Task` sub-classes in the tree, all `Data`, `Tool`, and `Role` classes connected by predefined associations; (3) for each of these classes, all the user defined classes and associations connected to it.

2.3. The presentation technique

Four kinds of views are used in E3 to organize template specifications: Inheritance, Task, Task Synchronization, and User View. Further, each Task and Task Synchronization view for a given model is labeled by a path, that gives the position of the model fragment in the model subtask tree.

Inheritance View Both class and association *Inheritance Views* (IV's) can be build. The class IV shows the standard object orientation inheritance relation among classes. For a given class, there are two IV's: the *super* IV, that is the inheritance hierarchy from `Object` to the given class, and the *sub* IV, that is the inheritance tree rooted by the given class. Figure 3(a) shows an example of class sub IV. Figure 1(b) gives the association IV for the predefined associations.

Task View The *Task View* (TV) gives information about a single `Task` sub-class, by giving the `Data` taken as input and produced as output, and the responsible role. For each `Data` it gives the manipulating tools, i.e., the tools used to process the data. Figure 4(a) is an example of TV.

Task Synchronization View A process model subtask tree models many levels of activity decompositions that are not shown in a single view. Each *Task Synchronization View* (TSV) depicts the synchronization scenario for a single level decomposition of the activity modeled by the

[5] Not to be confused with "role" of associations.

`Task` sub-class given by the path. The TSV specifies the dependency constraints among `Task` sub-classes by means of the predefined associations `preorder` and `feedback`. Figure 2 shows a sample TSV (Association `subtask` is never shown pictorially, but it is deduced from the path).

User View The *User View* (UV) enables to connect classes by user defined associations. While the three views above impose strict constraints on what can be shown, the UV imposes the only constraint that associations and classes belong to some IV. Figure 4(b) shows a simple product configuration scheme defined in a UV.

Path The position of a given task in the activity breakdown is traced by means of the *path*, starting from the task (`Model` sub-class) modeling the whole process. As the same task can be subtask of different composite activities, it can have more than one path. Abstract classes do not participate to any subtask tree and root their own path.

3. AN EXAMPLE

In the following, we will refer to the informal process model template that has served as input to our specification as the Iveco process manual. The E^3 specification of the Iveco process manual is referred as the `Iveco_model`, a `Model` sub-class.

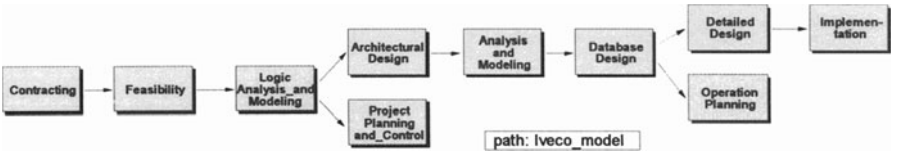

Figure 2. Iveco_model TSV.

`Iveco_model` is connected by `subtask` to 10 `Phase` sub-classes, each denoting one of the 10 main phases stated in the Iveco process manual. Figure 2 gives the `TSV` of the entire model. The sequencing among the different phases is strict and feedbacks are not allowed. Each of these phases is further decomposed in a production and a V&V sub-phases as inherited from abstract class `Phase`. Figure 3 shows both the IV and the TSV for class `Phase`.

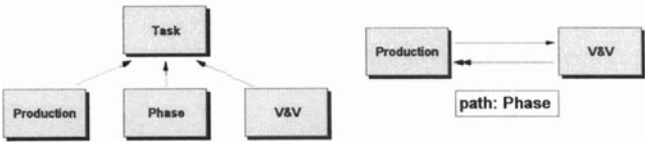

Figure 3. (a) `Phase` IV. (b) `Phase` TSV.

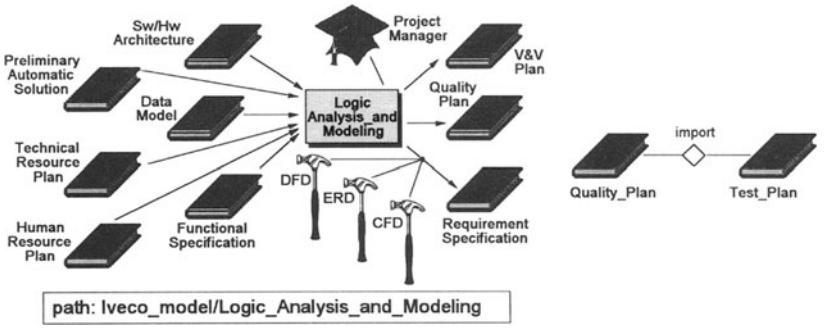

Figure 4. (a) `Logic_Analysis_and_Modeling` TV. (b) `Quality_Plan` UV.

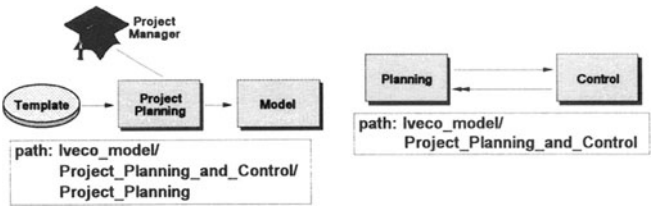

Figure 5. (a) Project_Planning_and_Control TSV. (b) Project_Planning TV.

Figure 4(a) gives the TV of class `Logic_Analysis_and_Modeling`. Class `Quality_Plan` is further specified in the UV depicted in Figure 4(b). Figure 5 shows both the TSV for `Project_Planning_and_Control` and the TV for `Project_Planning`. Project planning and control consists of two main phases, i.e., planning and control. The latter phase is supposed to transform a general process model description, i.e., a template, into a concrete process model, or plan.

4. CONCLUSIONS

In the process modeling arena [6-9], the efforts have been devoted toward definition of an universal process model language that combines different paradigms and that is powerful enough to specify, design, and implement process models. In E^3 , the main efforts are concentrated toward specification, i.e., toward understanding and communicating process models before trying to automate poor process models. A graphical editor that automates the E^3 specification method is under implementation. The E^3 specification language enables classification of process model elements into classes and associations. This helps the discovery of static inconsistencies in a process model definition, i.e., classes and associations play the same role that types play in programming languages. Meta-classes and meta-associations have been introduced to express constraints on process model templates. The meta-model level also enables to model explicitly the meta-process. `Project_Planning_and_Control` is an example of meta-phase specification. These aspects were confirmed by our experiment of specifying the template of Fiat Iveco. A monolithic textual document of 175 pages was transformed into about 200 classes and associations.

References

1. Conradi, R., Fernstrom, C., Fuggetta, A., and Snowdon, R.: "Towards a Reference Framework for Process Concepts", in J.-C. Derniame ed.: *Proc. from EWSPT'92*, Trondheim, Norway, Springer Verlag LNCS, September 1992, pages 3-18.
2. Feiler, P. H. and Humphrey, Watts: "Process Development and Enactment: Concepts and Definitions", in Leon Osterweil ed.: *Proc. from 2nd Int'l Conference on Software Process (ICSP'2)*, Berlin. IEEE-CS Press, March 1993, pages 28-40.
3. Booch, G.: "Object Oriented Design with Application", Benjamin Cummings Publishing Company, Inc., Redwood City, California, 1991.
4. Rumbaugh et al., J.: "Object-Oriented Modeling and Design", Prentice Hall, 500 pages, 1991.
5. Goldberg, A. and Robson, D.: "SmallTalk-80: The Language and its Implementation", Addison Wesley, 1983.
6. Estublier, J., and Melo, W.: "Software Process Model and Work Space Control in the Adele System", in L.Osterweil ed.: *Proc. of the 2nd IEEE International Conference on Software Process (ICSP'2)*, Berlin, Germany, February 1993.
7. Jaccheri, M. L. and Conradi, R.: "Techniques for Process Model Evolution in EPOS", IEEE Trans. on Software Engineering, December 1993, 19(12):1145-1156.
8. Fernström, C.: "Process WEAVER: Adding Process Support to UNIX", In L. Osterweil ed.: *Proc. of the 2nd IEEE International Conference on Software Process (ICSP'2)*, Berlin, Germany, February 1993.
9. Barghouti, N. S. and Kaiser, G. E.: "Scaling up rule-based development environments", *International Journal on Software Engineering and Knowledge*, Engineering, World Scientific, March 1992, 2(1): 59-78.