Evaluating the requirements for software process modeling languages and systems

M.Letizia Jaccheri
Norwegian University of Science and Technology (NTNU)
Trondheim, N-7034, Norway

Mario Baldi
Dipartimento di Automatica e Informatica, Politecnico di Torino Corso Duca degli Abruzzi, 24
Torino, I-10129, Italy

Monica Divitini
Norwegian University of Science and Technology (NTNU)
Trondheim, N-7034, Norway

ABSTRACT

We have defined a set of requirements for a process modeling language which are: multiple representation levels to describe general and instantiated processes; inheritance to factor and reuse common knowledge; process specific constructs with well defined syntax and semantics to describe process elements; and associations to relate and constraints such entities. A process modeling system must enable its users to inspect and analyze process models according to different perspectives. In this paper we discuss how we have set, implemented, and evaluated these requirements. The discussion is centered around a working example.

Keywords: software process modeling, object-orientation, associations.

1. INTRODUCTION

Research in software process modeling has led to process modeling languages, process centered software engineering environments (PSEE's), and methods to create, improve, enact, and reuse software process models [12]. Part of the resulting technologies has evolved into commercial products and there are documented approaches that have applied software process modeling to industrial cases [2] [4] [5] [16] [17] .

In this paper we will discuss a modeling work which was done in collaboration with the information system department of FIAT/Iveco. The first iteration has been our first attempt at producing an object oriented model of a real-world software process. During the first iteration we have identified a set of requirements for a Process Modeling Language (PML) and associated tool support. Then, we have designed and implemented E3 version 1 (v1) on the basis of these requirements. The main characteristics of the E3 PML are easiness of use also by non expert users, and support for abstraction and reuse, both in the context of the same model and across different models. In E3 we distinguish between instantiated process models and process templates. The term «process model» denotes whichever process description. A «template» is a general process model from which instantiated models can be obtained. A instantiated model is at the same abstraction level of a project plan, i.e., it includes time and resources information, while a template is at the same abstraction level of a quality manual.

During the second iteration, we have modeled the same process by using E3 v1 1. This iteration has produced: a high level process template, in the E3 PML, that is easily understandable and reusable for new case studies; a Smalltalk translation of this template that enables interactive simulation; recommendations for improvement of the input quality manual; a set of requirements for improving the E3 system. Then, we have developed a set of requirements for the E3 system version 2 (v2) and these requirements have guided the design and implementation of the second version of the system [19]. These requirements mainly concern the tool support part. The PML satisfies its requirements, i.e., it provides support for model understanding, abstraction, and reuse.

We have finally exploited this second version of the system to obtain a final model of the Iveco process.

The structure of this paper is as follows. Section 2 introduces the requirements and summarizes the main features of the E3 PML and the associated tool E3 p-draw following its evolution path from its conceiving to its latest version. Section 3 introduces the Iveco process, the respective E3 software process model. Conclusions are given in Section 4.

2. THE E3 SYSTEM

Here, we present the evolution of the E3 system from its initial requirements to its first and second implementation and we show how the modeling of the Iveco process has had a central role in this evolution. For more detailed explanations about E3, refer to [15].

This first modeling phase has showed that a special purpose PML is required. Thus, we have implemented the first version of the E3 PML and E3 p-draw and further we have exploited them for modeling the Iveco process. The second modeling phase has taken almost 1 month. We have manually translated the obtained process model into a Smalltalk program and we have simulated it.

This second tentative has resulted in a positive evaluation of the PML, and has given a set of extra requirements for the support tool. These requirements have guided the design and the implementation of the second version of E3 p-draw, described in Section 2.4. We have used one day to model the Iveco process with the second version of E3 p-draw.

Motivation

E3 has its roots in Object-Oriented (OO) techniques, and conveys two basic ideas. (1) Object-orientation can be successfully used for modeling software processes, as a software process consists of a net of interacting objects. (2) the description of a software process must be understood, analyzed, and improved before it is implemented and executed.

These two main ideas has guided us to our first process modeling experience during which we have modeled the Iveco process using the Coad and Yourdon [7] OO analysis method and a supporting tool. One process modeller has studied the quality manual and has produced three outputs:

1. a preliminary OO description of the Iveco process in the form of an OO model consisting of classes and relations;

2. a set of questions for the process owner;

3. a set of problems and requirements for a PML and associated tool support.

This first experience has demonstrated that process models can be obtained as the result of an OO analysis process. The effectiveness of an OO modeling approach to the description of software processes has set the following requirements for an OO PML.

Multiple representation levels: common knowledge about organization level processes can be represented in the form of classes and associations, and instantiated for each actual process. Also, common characteristics of classes can be derived by objects, which are instances of those classes. Project level information is often represented as plans while general knowledge about the organizational process is represented in the quality manual. A common framework is required to keep this kind of information related.

Inheritance: the models can be organized according to an OO inheritance hierarchy that enables to factor common knowledge in single classes and reuse it in specialized classes. The original Iveco quality manual describes the same information in several places, and these repetitions lead to inconsistencies. A typical example is the description of the document format: all document types share a common structure that should be represented in only one place.

On the other hand, when using Coad and Yourdon OO analysis method for software process modeling, we have faced the following problems:

Process specific constructs: typically, an OO software process model will contain a set of classes and associations that are general for several process models: a class document that describes the generic software item (including documentation), a class task that describes the generic activity, associations input and output that describe the constraints about data flow, etc.. Predefined process-dedicated syntax constructs are needed as non-trivial process templates can consist of hundreds of classes and associations that look as a flat web of identical boxes (classes) and arrows (associations). Hence, process specific constructs are needed to represent the main process components, like activities, products, tools, roles, together with the relations among them. These constructs must have an associate semantics that must be clear and intuitive. Since the language needs not to be executable, operational semantics definition is not mandatory.

Inspection and analysis: specific mechanisms designed for the software process domain are needed to help the user to inspect and analyze a model from different process perspectives. As it will be detailed in Section 3, the Iveco model encompasses 161 classes and 585 associations. Since it does not make sense to present more than circa 10 classes in a single page, one needs policies to section the model for presentation purposes. When inspecting an OO process model, the data and control flow perspective are of primary importance. In our context, data flow means that for a given task class, one is interested in seeing which are the input and output classes, etc. In addition to the classical data and control flow, for a given task, it is useful to find out which are its responsible agents, and which tools it uses. After a model has been produced, it is crucial to check for its correctness. This can be done by manual inspection of the model, also in cooperation with the process owners, but some automatic analysis mechanisms are needed. Analysis can be achieved either statically, i.e., by automatic inspection, or compilation of the model representation or automatically, by examining the behavior of the execution of the model. One way to obtain such behavior is model simulation.

Associations: process models must represent not only the entities involved in the process but also the relationships that relate and constrain such entities. Existing OO analysis and design methodologies and the related notations introduce associations like aggregation, use, or creation, and user-definable ones, but they do not provide a formal characterization. While classes are a means to express knowledge about local structure and behavior, associations express how classes can be related in building the global system. We state the requirement that associations must be first-class elements, like classes. If a class participates in an association, this knowledge has to be inherited by its subclasses. A subclass must have a means to redefine such inherited knowledge. This is crucial to allow a class to reuse not only local knowledge of its super class, but also the associations its super class participates in. The relationships that are common to every software process have to be represented in the associated template to facilitate template inspection, understanding, and evolution. The same association, e.g., precedence among activities, can appear several times in a template and the knowledge about a given association must be expressed by a single reusable item.

Syntax and semantics are not formally defined, thus preventing automatic analysis or simulation.

In the following description of the E3 system we will refer to examples taken from the representation of the Iveco process obtained using the language itself. The graphic notation showed in the figures of this paper is the one provided by the second version of E3 p-draw.

The E3 PML

We propose show how the E3 PML implements the requirements above. We will show how we have handled the implementation of requirement inspection and analysis in the different versions of the system.

Multiple representation levels: The E3 PML is an OO language augmented with associations. Attributes can be declared for each class and association. Methods can be declared for each class. Each class inherits methods and attributes from its superclasses. Each attribute has a name, a type and a value. Inherited attributes can be redefined and new ones can be added. A default value can be assigned to each attribute and it can be modified later. Each method has a name. We do not provide support for methods definition as this is a feature required for design and not for analysis support methods and tools [10]. The E3 PML offers three conceptual levels that represent information about a software process at different generalization levels. The first is the Instance level that is at the same abstraction level of a real process, the second is the Template level which represents characteristics common to various processes, the third is the MetaTemplate level which expresses general knowledge about how to manipulate Templates. The E3 PML offers objects and links at Instance level; classes and associations at Template level; metaclasses and metaassociations at MetaTemplate level. Each object is the instance of a class, and each link is the instance of an association. Moreover, each class is the instance of a metaclass and each association is the instance of a metaassociation. Thus, objects and links denote an instantiated (real) process, classes and associations denote a general process template, metaclasses and metaassociations express general knowledge about how to manipulate general templates.

Inheritance: Classes, metaclasses, and metaassociations are organized in three single inheritance hierarchies. A class inherits from its super class both attributes and methods, in the OO way. In addition, we have designed a special mechanism for association inheritance.

Process specific constructs: The E3 PML offers a kernel that consists of basic metaclasses, classes and metaassociations. Each process template consists of primary elements, e.g., tasks, roles, humans, tools, and software products, and the relationships among them. Thus, the PML offers these primary process elements as first order elements, in the same way as characters, strings, and integers are offered by a programming language. The kernel is immutable and it will be the same for each process template. Kernel metaclasses, are: E3object_mc, Task_mc, Data_mc, Tool_mc, and Role_mc. E3Object_mc is the super metaclass of the other metaclasses. Kernel classes are: E3object, Task, Data, Tool, and Role. E3Object is the super class of the other classes. Informally, Task denotes software process activities, Data denotes software products, Tool denotes tools or techniques, and Role denotes responsibilities. Kernel metaassociations are: E3assoc_ma, Aggregation_ma, Subtask_ma, Preorder_ma, Feedback_ma, Input_ma, Output_ma, Responsible_ma, Use_ma.

Inspection and analysis: this will be discussed later.

Associations: Local knowledge, i.e., attributes and methods of a class, is inherited in the OO standard way. A class also inherits the associations in which its super class participates and it can re-define them under well defined constraints. In this way, global knowledge, i.e., constraints about template fragments, or subnets, can be inherited as well. Inheritance and re-definition are the basis for abstracting and reusing knowledge in the context of a template. At the Template level, there is not inheritance among associations. Each association is the instance of a metaassociation. In addition, some associations are re-definitions of other associations. Each association takes its name from the name of its metaassociation, e.g., each instance of Aggregation_ma, will be named Aggregation. Instances of E3Assoc_ma will denote a whichever association between two whichever classes. Associations declare constraints about link instantiation. Only binary associations are provided as they are easy to be understood by not computer scientists. A binary association, say A(C1,C2) can be instantiated into several links, between *instances of C1 and instances of C2.*

Syntax and semantics: The language provides both a textual and a graphic notation. The PML is given a formal syntax and static semantics. The textual notation can be mapped onto the graphic one, and vice versa. The graphic notation should be exploited by users to develop, understand, and communicate software process templates, while the textual notation is manipulated by automated process modeling tools, such as parsers and analyzers. E3 PML is not an executable process modeling language. It is neither interpreted nor compiled, but it is analyzable. This has been a precise choice to privilege understanding, and analysis with respect to enaction.

The meaning of kernel classes is given in the following by expressing the semantics of the respective objects:

- Each Task instance denotes a software process activity. A Task instance will start when its predecessors have finished and its resources (input and roles) are ready.
- Role instances model actual resources with responsibilities and skills.
- A Data instance models a software process artifact or product component, like source code and anomaly forms.
- Tool sub classes define methods of work. A Tool instance represents a precise version of an automated tool or a written procedure.

The meaning of associations is given in the following by explaining the semantics of the corresponding links.

- link Aggregation(o1,o2) denotes that object o2 is part of o1.
- link Subtask(t1,t2) denotes that object t2 of class Task is a subprocess of t1.
- link Preorder(t1,t2) denotes that the t2 can start only after each task connected to it by link Preorder has finished.
- link Feedback(t1,t2) denotes that t2 must re-start when t1 finishes with no approval.
- link Responsible(t1,r1) denotes that the human resource r1 is responsible for carrying out task t1.
- link Input(t1, d1) denotes that t1 takes as input d1.
- link Output(t1, d1) denotes that t1 produces as output d1.
- link Use(t1, tool1) denotes that t1 exploits tool tool1.

E3 p-draw version 1

The first version of the E3 system (E3 v1) is based on Interviews, C++, and the OODBMS Objectivity. The associated tool E3 p-draw enables its users to develop, communicate, and browse software process templates. It has the goal to render intuitive the development of software process templates written in the E3 PML providing the graphical representation of classes and associations slightly different from the E3v2 one shown in this work. It offers operations to create, change, browse, and delete items. An operation can have side effects, when it would violate the constraints set by the language. For example, the deletion of a metaassociation triggers the deletion of its associations. These side effects never take place without the consensus of the user.

The graphical interface of the first version of E3 p-draw is based on Interviews; many E3 p-draw instances can run concurrently in a distributed environment and act as clients of the OODBMS Objectivity. A process modeling module, or simply module, is the working unit of the tool. A module can be saved, inspected, and retrieved.

E3 p-draw offers four kinds of views: Inheritance, Task, Task Synchronization, and User View, that implement respectively OO (Inheritance view), functional (Task View), control (Task Synchronization View), and informational (User View) perspectives.

- There are two kinds of Inheritance views:The SubInheritance View for C is the set of all classes that inherit from C.
- The SuperInheritance View for C is the transitive closure of all classes from which C inherits.

2. The Task View for T contains: T; all definitions (including the inherited ones) and re-definitions of associations input(T, Di), output(T, Dk), and responsible(T, Rm); all Di, Dk, and Rm.

3. The Task Synchronization View (TSV) of T contains: T; all definitions of association subtask(T, Xi); all definitions of associations preorder(Xi,Xj)and feedback(Xi,Xj), with Xi,Xj belong to Xseq, where Xseq is the sequence of the above Xi that are subtask of T.

4. The User View allows the modeller to define whichever association. The modeller must provide each User View with a name (or label) to enable the system to store and retrieve the view for subsequent browsing.

Task Views and Task Synchronization Views are labeled by the path through the associations subtask from the class network origin (the Task subclass that represents the whole template) to the given Task.

The PML imposes a set of constraints, e.g., it is forbidden to connect two Data subclasses by a preorder association. Other properties can be checked by means of queries on the underlying DBMS, but are not enforced. For example, one can check that there is at least a responsible role for each task. Examples of constraints that one can impose on a process template are: each Task class must produce at least an output Data class; a Task class cannot be connected by a feedback association to another Task class if the second class is not a predecessor of the first one, according to the relation defined by association preorder. A Task class C1 is predecessor of Task class C2 if there is an association preorder(C1,C2) between the two, or if it exists Task class Cx such that Cx is predecessor of C2 and there is an association preorder(C1,Cx).

Smalltalk Simualtion

We have implemented and simulated the Iveco process template using Smalltalk [13]. Simulation works according to the following general rules: each father task dynamically creates its subtasks which are executed according to synchronization constraints given by associations preorder and feedback. During the simulation objects are dynamically created and destroyed. Simulation is centralized: a single user interacts with the simulating process in order to provide run-time parameters that emulate choices made by people involved in the process. E.g., the user interfaces of the process participants are displayed; each interface provides the user with the list of activities the process participant is responsible for and the user makes decisions concerning the execution time and outcome of activities. A log of the simulation is recorded to allow off-line analysis of the results. We have developed some scripts that correspond to different scenarios. However, the simulation needs the interaction with the user.

We believe that the simulation is useful to test synchronization constraints among tasks; moreover, given a certain number of process participants each playing one or more roles, the simulation allows possible work overloads on some of the participants to be detected.

Lessons learnt, requirements for E3 version 2

The result of this work is a set of recommendations for improvement of the manual under study, an evaluation of the E3 PML and the related first version of E3 p-draw, and the implementation of v2 of the E3 system.

The strengths of the E3 system that have been revealed by this work are:

Inheritance: It is not immediate for users to understand Inheritance hierarchies. However, after a short explanation users are able to understand concepts like inheritance that factorizes structure and behavior that are common to all production phases in the template; the process owners admitted the importance of having the same concept represented in only one part of the manual. Standard OO reuse mechanisms that make possible for a class to inherit attributes and methods declared for its super class are not so useful at modeling level. In fact, at this level, a class (and respectively an association) is not characterized by its attributes and methods, but by the associations it participates in. Thus, our definition and re-definition mechanisms that enable reuse by inheritance of the structure of clusters of classes, was heavily used. The definition and re-definition mechanism was useful to declare global structural constraints that otherwise were not easy to declare with an OO language.

Process specific constructs: kernel classes and associations and their associated semantics facilitates model understanding and communication. They constitute the first classification axis of a process model, i.e., each item will be classified as a Task, or Tool, etc.. This increases accuracy in process models.

Associations: The original textual process description sometimes does not distinguish between activity, product, and tool description, i.e., products and tools are described within the activity description, thus the same description is repeated if the same kind of products and tools are manipulated by different activities. Associations help in producing declarative not redundant models.

Syntax and semantics: The graphic features of E^3 p-draw help process understanding. On the other hand, a well defined syntax and static semantics helps in producing consistent models.

The main weaknesses of E3v1 revealed by this case study are:

Multiple representation levels:

- Lack of instance level facilities: E^3 p-draw v1 does not support the Instance level. While a template is an abstract description for a set of model, a model is a description of a single process, including time and resource binding. If a template has to be understood and used, it must be possible to generate (either automatically or manually) instantiated models.

Inspection and analysis:

- Lack of a flexible view mechanism: the User View was conceived to increase flexibility since it allows general relationships among process elements to be expressed. These relationships cannot be included in the other views which constraint the usable associations. Nevertheless, User Views as defined here are difficult to manage and not easy and friendly to use. Flexible views must be defined. Also, the views provided by E^3 p-draw v1 are Task oriented and do not enable the user to browse a model from a perspective that is different from the Task one. On the contrary it can be useful, for a given product to see which tasks that consume it, or which that produces it, etc. Analogously for tools and roles.
- Problems in the simulation support: the Smalltalk simulation showed absence of trivial errors, e.g., deadlocks. However, it cannot be regarded as a true simulation in which probabilistic parameters are assigned to activities and resources as was suggested by the process owners. Also, the manual translation from E3 PML to Smalltalk can introduce errors. We have then abandoned this research path and we have decided to focus on static analysis instead. Static analysis is more suitable than dynamic simulation if the purpose of the models is understanding by humans and not execution. This assumption is supported by the fact that it was difficult for the users to understand and appreciate the Smalltalk simulation.

E3 p-draw version 2

The second version of the E3 system (E3v2) is written in Java, therefore it is portable on many platforms. E3v2 does not exploit a DBMS, rather, for portability reasons, the file manipulation functionality provided by the Java language.

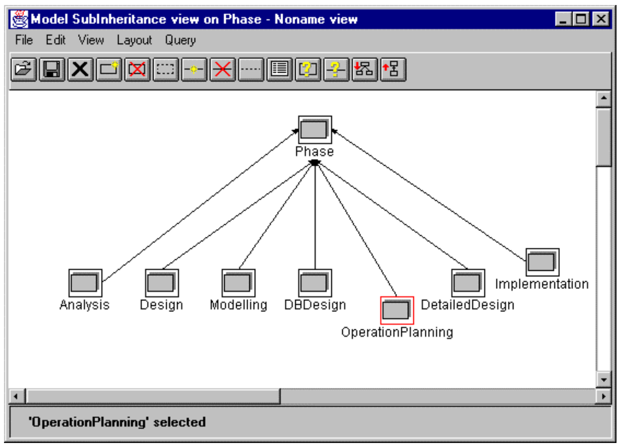
In addition to the obvious advantage of portability, the second version of the E3 system extends the first one in the following directions: we have added the Instance level. We have extended the view mechanism to enable the user to design his own views while maintaining compatibility with the data flow and control flow kinds of views provided by E3v1. E3 v2 offers static analysis of process models by means of queries.

Views

The tool supports three kinds of views: editable views, inheritance views, and derived views.

Editable views are available for each of the three PML levels. Inheritance views are not available for the Instance level. Some kinds of derived views do not make sense for the MetaTemplate level. An editable view is like a blackboard in which the user can create, modify, and delete PML items.

There are two kinds of inheritance views as in the first version of the E3 system: the superinheritance view, that for a given class displays its superclasses and the subinheritance view, that for a given class displays its subclasses, i.e., the inheritance tree rooted by the given class. Figure 1 shows an example of subinheritance view.

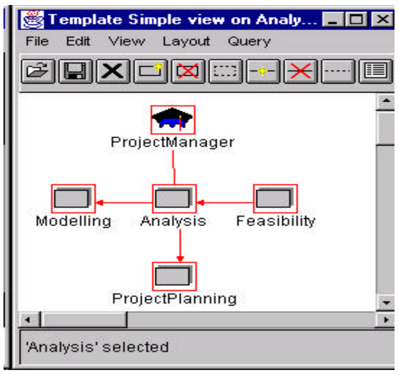


'OperationPlanning' selected

Figure 1: The subinheritance view for class Phase.

The derived views, or only views, enable the user to browse a process model in a selective way. There are four kinds of derived views: simple, simple recursive, composite, composite recursive. In addition, one can customize derived views by making invisible one or more metaassociations (if one operates at the Template level), or respectively associations (if one operates at the Model level). If a metaassociation is made invisible, its instance associations will not appear in derived views. If an association is made invisible, its instance links will not appear.

The simple view refers to a class (respectively object) and shows all associations (respectively links) which involve the class (respectively object). For example, in the case that metaassociations input_ma and output_ma are the only visible ones, the simple view for a Task sub class will show the classes connected to it by input and output associations (this is equivalent to a data flow oriented view). If metaassociation preorder is the only visible one, the simple view for a Task sub class will show its predecessor and successor classes (this is equivalent to a control flow oriented view, i.e., the Task view for E3 p-draw v1). Figure 2 shows an example of a simple view (of class Analysis) where only responsible_ma and preorder_ma metaassociations are visible. Note that the simple view extends the Task view as it can be applied to whichever class and object while the Task view was applicable only to Task classes.



'Analysis' selected

Figure 2: The Simple View for class Analysis.

The simple recursive view takes as input a class and it recursively applies the simple view to all classes related to the selected one. The composite view takes as input a class and shows all the associations that are instances of Aggregation_ma and in which the input class participates as destination class. Also, for each class that participates to the above associations as source class, it shows its simple view.

The recursive composite view refers to a class and it recursively applies the composite view on the class and its components. Figure 3 shows the recursive composite view for class Analysis, visible metaassociations subtask_ma and responsible_ma, i.e., it shows the task breakdown structure together with the assigned Role class, for each Task class.

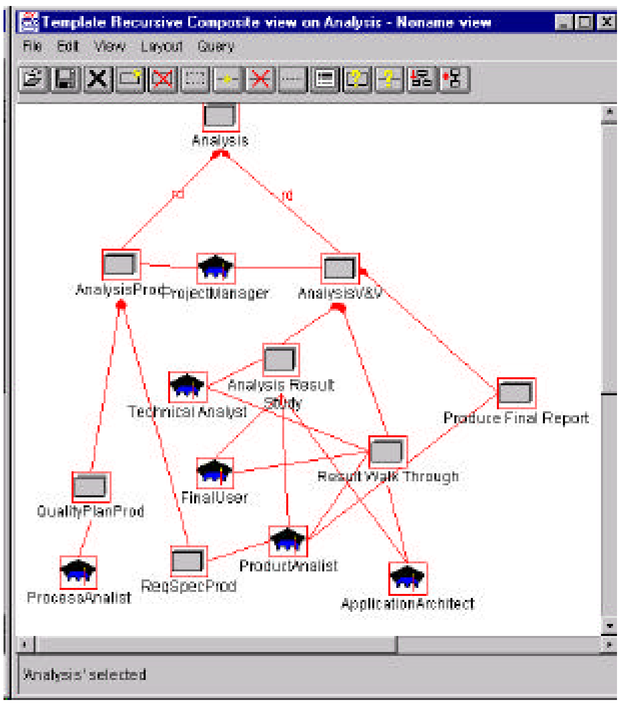


Figure 3: Recursive composite view for of class Analysis (visible metaassociations subtask_ma and responsible_ma).

Not all the views are provided at each level. In particular, at MetaTemplate level, the tool offers inheritance and simple views; at Template level all views are offered; at Instance level, inheritance views do not make sense, thus they are not provided.

Queries

The tool supports two kinds of analysis mechanisms that can be used at either Template or Instance level: check property presence and check property absence. Check property presence takes as input a MetaAssociation and shows those class pairs related by those associations that are instances of the given MetaAssociation. The user can choose to check this property both on the current view and on the whole module.

Check property absence takes as input a MetaAssociation and shows those classes that could participate to instances of the given MetaAssociation but they do not. Again, the user can choose to check this property on either a view or the whole template.

For example one can ask for absence of input; this corresponds to find those Task sub classes that are not directly connected by associations input. Note that the same result could be achieved by opening a recursive composite view on Analysis with input_ma visible. The query mechanism is preferable to the view one when a large portion of the template has to be analyzed.

Instance Level

E3 v2 offers support for the Instance (also called Model) level. An instantiated model can be automatically instantiated from a template. Further it can be manually customized. A part of a Template, denoted by a view, can be instantiated.

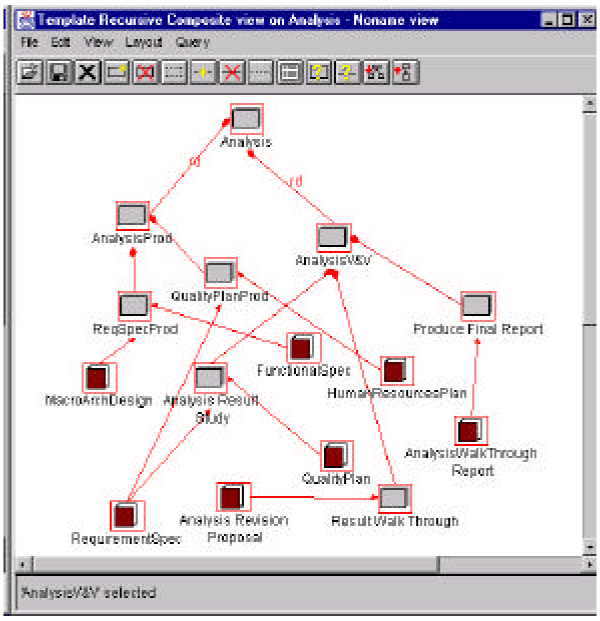


Figure 4: Recursive composite view for class Analysis (visible metaassociations subtask_ma and input_ma).

3. MODELING THE IVECO PROCESS IN E3

This section introduces the input quality manual, it outlines the E3 template, and finally discusses encountered problems and lessons learnt.

Original description of the Iveco Quality manual

The input was a manual of 175 pages of text and pictures that describes the software process of FIAT/Iveco. The software process is a very general one, that covers all software life cycles. FIAT/Iveco is a truck manufacturer that has an information system department with circa 200 persons developing administrative software.

In this section, we give a summary of the Iveco quality manual. First, we describe the main phases and their interaction, second, the document format, then, verification and validation issues. Finally, we describe in more detail one of the phases.

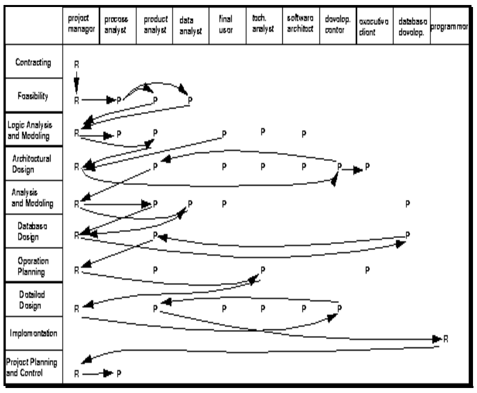
The main phases

The manual defines 10 main phases and for each phase it identifies:

- the role the responsible person must play;
- the main output documents and their semantic and syntactic standards;
- goals, concepts, and rules;
- description of the respective verification and validation (V&V) phase in terms of activities;
- techniques, tools, and responsibility.

The Iveco manual describes activities and responsibilities. Input documents for phases are not defined.

Figure 5 reproduces a table from the original quality manual. It displays process roles (on columns) and process phases (on rows). An «R» at the cross of column rolei with row phasej denotes that rolei is responsible for phase phasej while a «P» denotes participation. Arrows denote flow of information from one role to another.



	project manager	process analyst	product analyst	data analyst	final user	tech. analyst	software architect	develop. center	executive client	database develop.	programmer
Contracting	R										
Feasibility	R	P	P	P							
Logic Analysis and Modeling	R	P	P		P	P	P				
Architectural Design	R		P		P	P	P	P	P		
Analysis and Modeling	R		P	P	P					P	
Database Design	R		P							P	
Operation Planning	R		P			P			P		
Detailed Design	R		P		P	P	P	P			
Implementation											R
Project Planning and Control	R	P									

Figure 5: Responsibility table.

The roles of the persons responsible for the V&V activities are spread in activity descriptions. Finally, the manual describes a number of tools, along with some instructions and advises about their usage, and a few techniques, along with their suitability for being used in various activities.

Documents

The format of textual documents is described as displayed below. The quality manual is written following this standard template that is described in the manual itself.

Title

authors:

reviewers:

document code:

language:

version:

reference project:

class:

date:

Table of contents

X Name of Chapter

.X Name of SubChapter

.XX Name of Paragraph

.XXX Name of SubParagraph

Verification and Validation

An entire section of the manual is devoted to the description of a general V&V methodology that must be applied to the products of any activity. For each production phase, there must be a V&V phase. Each V&V phase is described with its:

- goals (quality attributes);
- respective production phase;
- tools.

Phase Logic Analysis and Modeling

A summary of the description of phase Logic Analysis and Modeling, as it is written in the Iveco quality manual, is summarized below. Here we call this phase Analysis. The manual contains a detailed description of both the phase outputs and exit conditions.

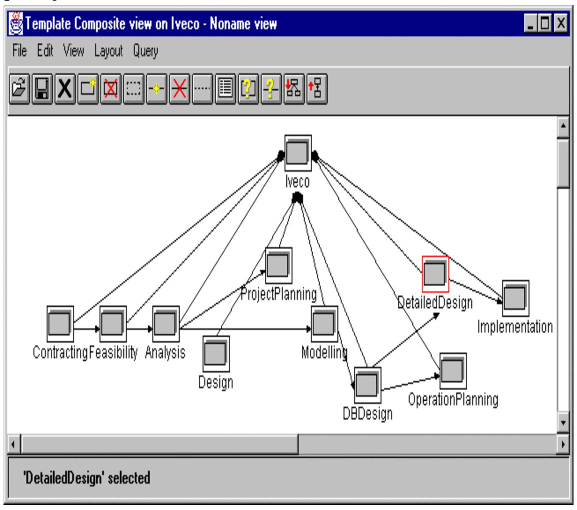
- Goal: detailed description and validation of system functional and non-functional requirements.
- Method: structured analysis.
- Techniques: DFD (Data Flow Diagram), ERD (Entity Relationship Diagram), CFD (Control Flow Diagram).
- Concepts: Abstraction, information hiding, process decomposition, event list, data dictionary, temporal constraints, hardware architecture.
- Results: Requirements specification, Quality Plan.

Generally, the decomposition of the phases in subtasks is not explicitly described in the manual; we inferred it from the final and interim outputs of the phase.

The Iveco Process Template

The E3 process template of the Iveco quality manual encompasses 161 user defined classes and 585 associations. No user-defined metaassociations were needed as no other connections other than those modeled by the E3 metaassociations were found in the manual. On the basis of our experiences with other case studies [18] we believe that the associations provided by E3 are sufficient to model almost each connection in the context of software processes.

One way to present our Template is to start from Figure 6, that describes task break down. Class Iveco in Figure 6 is connected by associations subtask to 10 Task subclasses, each denoting one of the 10 main phases identified in the Iveco quality manual.



'DetailedDesign' selected

Figure 6: Composite view for class Iveco.

In Figure 6, the sequencing among the different phases is strict and feedback's are not allowed.

After inspecting the task break down, let us examine Figure 4 that gives the subinheritance view for class Phase. Each class that inherits from Phase, e.g., Design, also inherits the association involving Phase.

Figure 7 and Figure 8 show the composite views for the classes AnalysisProd and AnalysisV&V which are the subactivities constituting the task Analysis.

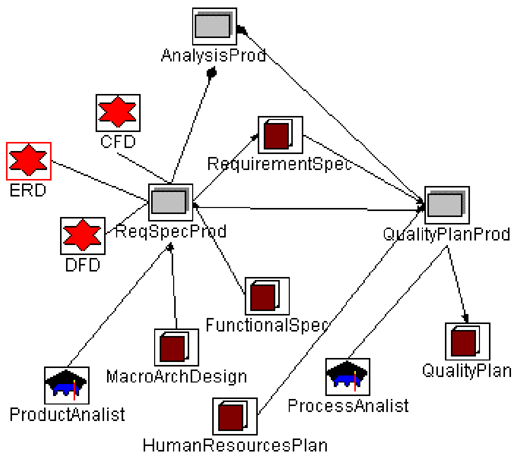


Figure 7: The Composite view for class AnalysisProd.

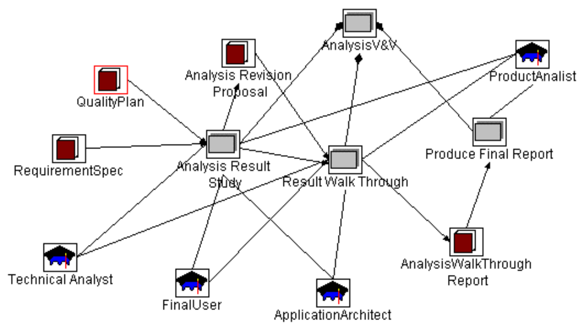


Figure 8: The Composite view for class AnalysisV&V.

Figure 9 gives the recursive composite view that defines the structure of documents according to the description given in the Iveco quality manual. Class Document is related by means of definitions of associations aggregation to classes Title and Chapter. The E3 PML does not impose any syntactic constraints on attribute and method definitions. Therefore, they can be either defined informally, or coded with an OO language, e.g., Smalltalk, for subsequent simulation, or implemented with an enactable OO PML, e.g., EPOS [14].

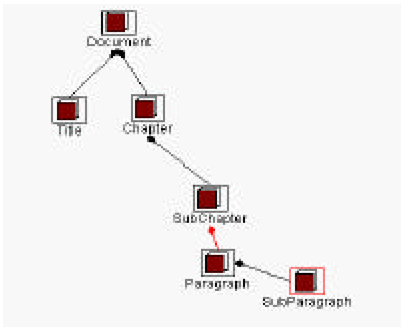


Figure 9: The Recursive composite view for class Document (visible metaassociation aggregation_ma).

Consistency management
The storage, view, and query mechanisms of E3 p-draw enable the modeller to query and browse a template. The modeller can either define his own queries or use the set of predefined queries. Examples of predefined queries are: check that each task is assigned to a responsible role; that each product is associated with a given quality attribute, etc..

This has helped in validating the template and implementing consistency checks, such as find out which output are produced but never used, which tasks do not have a well defined responsible person, etc..

If we compare the E3 Iveco process template specification with the informal description given in the Iveco quality manual, the former has several advantages: 1) task sequencing is explicit; 2) task input/output are explicit; the description granularity level is finer; 3) inconsistency have been detected and fixed. The main source of inconsistencies in a quality manual is the continuous change and adaptation of the manual itself that is performed by different persons at different times. It is a well known configuration management problem that a change to an item may render non-consistent several other items that depend, in some way, on the changed item. An explicit representation of the dependencies among items, by means of associations, facilitates the process of both removing inconsistencies and maintaining a consistent definition. Also, the different associations model different kinds of dependencies among items, thus providing a basis for maintaining the quality manual.

Some high level consistency constraints are enforced by the PML, e.g., each activity must be connected to some higher level ones, or must be the most high level one. Mentioned input and output products, associated tools, and roles, must have been declared before they are connected to an activity.

4. Conclusions

E3 is a research prototype which has been designed and implemented to evaluate two main ideas: first, process models are important as communication vehicles thought not executable, second, modeling notations can be regarded as a starting point for process modeling. From these initial ideas we have developed a set of requirements and implemented two versions of the E3 system.

Experience with the system validates our initial ideas and enables us to range our requirements. A PML must offer simple process specific constructs will well defined syntax. Such constructs must also include associations, such as relationships between activities (preorder and subtask) and document flow. The meaning of such constructs must be defined and clear in the context of the organization that use the PML. The PML must enable a clear distinction among general process models and project specific ones. A supporting process modeling tool must provide support to create, retrieve, and modify process models according to different perspectives.

These kinds of requirements can be offered by a simple process modeling tool such as E3, or they can be integrated in a more complex system which offers also other process support facilities such as workflow management.

Ongoing and future work include further validation of the E3 system both in industrial and in academic settings.

5. References

[1] Aoyama, M. Concurrent Development Process Model, IEEE Software, Vol. 10, No. 4, July 1993, pages 46-56.

[2] Aumaitre, J. and Dowson, M. and Harjani, D. Lessons learned from formalizing and implementing a large Process Model, Proc. of the third European Workshop on Software Process Technology, February 1994, Grenoble (France), pages 227-240.

[3] Baldi, M. and Jaccheri, M.L. Software Process Model Specification, Proc. of the first IFIP/SQI International Conference on Software Quality and Productivity: Theory, Practice, Education and Training, December 1994, City Polytechnic of Hong Kong, Hong Kong, pages 149-155.

[4] Bandinelli, S. and Fuggetta, A. and Lavazza, L. and Loi, M. and Picco, G.P. Modeling and Improving an Industrial Software Process, IEEE Trans. on Software Engineering, Vol. 21, No. 5, May 1995, pages 440-454.

[5] Barghouti, N.S. and Rosenblum D.S. and Belanger D.G., Two Case Studies in Modeling Real, Corporate Processes, Software Process Improvement and Practice Journal, Vol. 1, Num. 1, August 1995, pages 17-32.

[6] Cain, B. G. and Coplien, J.O. A Role-Based Empirical Process Modeling Environment, 2nd International Conference on the Software Process, February 1993, pages 125-135.

[7] Coad, P. and Yourdon, E., Object-Oriented Analysis, Prentice Hall, Englewood Cliffs, 1991.

[8] Coad, P. and Yourdon, E., Object-Oriented Design, Prentice Hall, Englewood Cliffs, 1991.

[9] Dion, R. Process Improvement and the Corporate Balance Sheet, IEEE Software, Vol. 10, No. 4, July 1993, pages 28-36.

[10] D. W. Embley and R.B. Jackson and S.N. Woodfield, OO Systems Analysis: Is It or Isn't It?, IEEE Software, Vol. 12, No. 4, July 1995, pages 18-33.

[11] E3 Web page, http://www.idi.ntnu.no/~letizia/E3DIR/e3.html.

[12] Finkelstein, A. and Kramer, J. and Nuseibeh, B. Ed. Software Process Modelling and Technology, Research Studies Press Ltd., Tauton, Somerset, England, 1994.

[13] A. Goldberg and D. Robson, Smalltalk-80: The language and its implementation. Addison Wesley, 1983.

[14] Jaccheri, M.L. and Conradi, R. Techniques for Process Model Evolution in EPOS, IEEE Trans. on Software Engineering, December 1993, Vol. 12, No. 19, pages 1145-1156.

[15] Jaccheri, M.L. and Picco, G.P., and Lago, P., and Eliciting Process Models in E3, ACM Transactions on Software Engineering and Methodology, 7:4, October 1988, pages 368-410.

[16] Kellner, M.I. and Hansen, G.A. Software Process Modeling: A Case Study, Proc. of twenty-Second Annual Hawaii International Conference on System Sciences, Vol. 2 - Software Track, edited by B.D. Shiver, IEEE, January 1989, pages "175-188.

[17] Mc Gowan, C.L. and Bohner, S.A, Model based Process Assessment, Proc. of 15th IEEE Iternational Conference on Software Engineering, Baltimora, MA, 1993, pages 202-211.

[18] Milano A., Modellazione di un processo industriale, Master Thesis, Politecnico di Torino, in Italian.

[19] Project E3, associated documentation, and source code, http://www.polito.it/~letizia.

[20] Rumbaugh, J. et al., Object-Oriented Modeling and Design, Prentice Hall, 1991, 500 pages.