

UPGRADE is the European Journal for the Informatics Professional, published bimonthly at <<http://www.upgrade-cepis.org>>

UPGRADE is the anchor point for UPENET (UPGRADE European NETWORK), the network of CEPIs member societies' publications, that currently includes the following ones:

- Mondo Digitale, digital journal from the Italian CEPIs society AICA
- Novática, journal from the Spanish CEPIs society ATI
- Piroforiki, journal from the Cyprus CEPIs society CCS
- Pro Dialog, journal from the Polish CEPIs society PTI-PIPS

Publisher

UPGRADE is published on behalf of CEPIs (Council of European Professional Informatics Societies, <<http://www.cepis.org/>>) by Novática <<http://www.ati.es/novatica/>>, journal of the Spanish CEPIs society ATI (Asociación de Técnicos de Informática <<http://www.ati.es/>>).

UPGRADE is also published in Spanish (full issue printed, some articles online) by Novática, and in Italian (abstracts and some articles online) by the Italian CEPIs society ALSI <<http://www.alsi.it/>> and the Italian IT portal Tecnoteca <<http://www.tecnoteca.it/>>.

UPGRADE was created in October 2000 by CEPIs and was first published by Novática and INFORMATIK/INFORMATIQUE, bimonthly journal of SVI/FSI (Swiss Federation of Professional Informatics Societies, <<http://www.svifsi.ch/>>).

Editorial Team

Chief Editor: Rafael Fernández Calvo, Spain, <rfoalvo@ati.es>

Associate Editors:

- François Louis Nicolet, Switzerland, <nicolet@acm.org>
- Roberto Carniel, Italy, <carniel@dgt.uniud.it>
- Zakaria Maamar, Arab Emirates, <Zakaria.Maamar@zu.ac.ae>
- Soraya Kouadri Mostéfaoui, Switzerland, <soraya.kouadrimostefaoui@unifr.ch>

Editorial Board

Prof. Wolfgang Stucky, CEPIs Past President
Prof. Nello Scarabottolo, CEPIs Vice President
Fernando Piera Gómez and Rafael Fernández Calvo, ATI (Spain)
François Louis Nicolet, SI (Switzerland)
Roberto Carniel, ALSI – Tecnoteca (Italy)

UPENET Advisory Board

Franco Filippazzi (Mondo Digitale, Italy)
Rafael Fernández Calvo (Novática, Spain)
Panicos Masouras (Piroforiki, Cyprus)
Andrzej Marciniak (Pro Dialog, Poland)

English Editors: Mike Andersson, Richard Butchart, David Cash, Arthur Cook, Tracey Darch, Laura Davies, Nick Dunn, Rodney Fennemore, Hilary Green, Roger Harris, Michael Hird, Jim Holder, Alasdair MacLeod, Pat Moody, Adam David Moss, Phil Parkin, Brian Robson.

Cover page designed by Antonio Crespo Foix, © ATI 2004

Layout: Pascale Schürmann

Editorial correspondence: see "Editorial Team" above

Advertising correspondence: <novatica@ati.es>

Upgrade Newsletter available at <<http://www.upgrade-cepis.org/pages/editinfo.html#newsletter>>

Copyright

© Novática 2004 (for the monograph and the cover page)

© CEPIs 2004 (for the sections MOSAIC and UPENET)

All rights reserved. Abstracting is permitted with credit to the source. For copying, reprint, or republication permission, contact the Editorial Team.

The opinions expressed by the authors are their exclusive responsibility.

ISSN 1684-5285

Next issue (December 2004):
"Cryptography"

(The full schedule of UPGRADE is available at our website)

- 2 Editorial
Four Years of UPGRADE

SPT, Software Process Technology

Guest Editors: Francisco Ruiz-González and Gerardo Canfora

Joint monograph with Novática*

- 3 Presentation
Software Process Technology: Improving Software Project Management and Product Quality – *Francisco Ruiz-González and Gerardo Canfora*
- 6 Software Process: Characteristics, Technology and Environments – *Francisco Ruiz-González and Gerardo Canfora*
- 11 Key Issues and New Challenges in Software Process Technology – *Jean-Claude Derniame and Flavio Oquendo*
- 17 A Taxonomy of Software Engineering Environment Services: The Upcoming ISO/IEC Standard 15940 – *Dan Hyung Lee and Juan Garbajosa-Sopeña*
- 22 Open Source and Free Software: A New Model for The Software Development Process? – *Alfonso Fuggetta*
- 27 Applying The Basic Principles of Model Engineering to The Field of Process Engineering – *Jean Bézivin and Erwan Breton*
- 34 Software Process Modelling Languages Based on UML – *Pere Botella i López, Xavier Franch-Gutiérrez, and Josep M. Ribó-Balust*
- 40 Supporting the Software Process in A Process-centred Software Engineering Environment – *Hans-Ulrich Kobialka*
- 47 Managing Distributed Projects in GENESIS – *Lerina Aversano, Andrea De Lucia, Matteo Gaeta, Pierluigi Ritrovato, and Maria-Luisa Villani*
- 53 Software Process Measurement – *Félix García-Rubio, Francisco Ruiz-González, and Mario Piattini-Velthuis*
- 59 Process Diversity and how Practitioners Can Manage It – *Danilo Caivano and Corrado Aaron Visaggio*

MOSAIC

- 67 Data Architecture
A Disquisition on The Performance Behaviour of Binary Search Tree Data Structures – *Dominique A. Heger*
- 75 News & Events: News from CEPIs and EUCIP; SCI 2005 (Call for Papers)

UPENET (UPGRADE European NETWORK)

- 77 From **Novática** (Spain):
IT and Disabilities
Braille and The Pleasure of Reading: We Blind People Want to Continue Reading with Our Fingers – *Carmen Bonet-Borrás*
- 84 From **Piroforiki** (Cyprus):
Information Technology in Today's Organizations
Is the IT Productivity Paradox Resolved? – *Kyriakos E. Georgiou*

* This monograph will be also published in Spanish (full issue printed; summary, abstracts and some articles online) by **Novática**, journal of the Spanish CEPIs society ATI (*Asociación de Técnicos de Informática*) at <<http://www.ati.es/novatica/>>, and in Italian (online edition only, containing summary abstracts and some articles) by the Italian CEPIs society ALSI (*Associazione nazionale Laureati in Scienze dell'informazione e Informatica*) and the Italian IT portal Tecnoteca at <<http://www.tecnoteca.it/>>.

Software Process Modelling Languages Based on UML

Pere Botella i López, Xavier Franch-Gutiérrez, and Josep M. Ribó-Balust

A Software Process Model (SPM) is a description of the structural and behavioural aspects of a process in the field of software development using some Process Modelling Language (PML) as description formalism. In the last 15 years, process modelling and, particularly, software process modelling, has gained a growing importance as a mechanism which allows, on the one hand, a better understanding of the process, which facilitates its assessment and improvement; and, on the other hand, the ability to automate to a certain extent its enactment, as it is usual in other engineering fields. A fundamental challenge concerning SPMs is how to find a standard PML to describe them. For this reason, in the last few years, an important research effort has been made in order to adapt UML (Unified Modelling Language) to the specific requirements of SPMs and, as a result, some UML profiles and metamodels (like SPEM or PROMENADE), which propose software process modelling formalisms based on UML, have emerged. In this article we outline the state of the art in the subject of software process modelling, we present its challenges and we focus specially in the use of UML as PML.

Keywords: Process Modelling Language, Software Process Model, UML, UML Extension.

1 Introduction: The Issue of Standardization in SPM

Software Process Technology has emerged in the late eighties, but in two fronts. One of them, starting from the seminal work of Leo Osterweil (*Software processes are software too*) [7] has been named Software Process Modelling (SPM), and has been very active from the research point of view, with a lot of contributions in general or specialised conferences and magazines, but with a relative small impact in the industrial world. The main goal of SPM has been the design of Process Modelling Languages (PML) able to describe software processes in a formal way. The other front refers to Software Process Assessment (SPA) and Software Process Improvement (SPI). It comes from the pioneering works by Watts Humphrey [4] developed at the SEI (Software Engineering Institute) and which has led to the definition of the CMM (Capability Maturity Model).

With a strong impact in the industry, this front has evolved with new models, as the CMMi (Capability Maturity Model integration) family, the ISO standard SPICE and other models. It has become an established industrial practice. Theoretically speaking, SPM is a good basis for SPA and SPI, since a formally described process will be easier to assess and, then, to improve. But in real life, all the SPA&SPI models do not need a formal description as a starting point. This fact, together with the proliferation of languages and notations that have emerged to describe software processes [2][1], are probably the reasons for the low impact of the SPM research. Probably, the first step to be taken so that the software industry may benefit from the SPM research is to standardize PMLs.

Pere Botella i López is Full Professor at the UPC (*Universitat Politècnica de Catalunya*), Barcelona, Spain. He has been active in the software engineering field from more than 25 years. Has been Dean of the Faculty of Informatics (1992-1998) and Vice-rector of the UPC (1982-1986, 1998-2002). From 1989 to now, Director of the Master program on Software Engineering at the UPC. Author of more than 40 publications. Program committee member of several international conferences, including ESEC, ICSE, RE, etc., being executive chair and co-editor of ESEC'95. Member of the Steering Committees for ESEC and JISBD (the spanish main event on software engineering). He has been coordinator in Spain for RENOIR (European Network of Excellence in Requirements Engineering). <botella@lsi.upc.es>

Xavier Franch-Gutiérrez is Associate Professor in the Software Department (LSI) at the UPC (*Universitat Politècnica de Catalunya*), Barcelona, Spain. He received his BSc and PhD in Software from the UPC. He is and has been a principal and co-investigator of several funded research projects. He is currently leading the research group in Software Engineering for Information Systems (GESSI) at the LSI, <<http://www.lsi.upc.es/~gessi/>>, compound of more than 10 full-time researchers. His current lines of research include requirements engineering, selection of COTS components, quality model construction and software process modelling. He has over 40 refereed publications in conferences, journals and books. <franch@lsi.upc.es>

José M. Ribó-Balust holds a BSc degree (1990) and a PhD (2002) in Computer Science, both from UPC (*Universitat Politècnica de Catalunya*), Barcelona, Spain. His thesis work defined PROMENADE as a UML-based software process modelling language. Since 1997 he is a member of the research group in Software Engineering for Information Systems (GESSI) at the UPC. He also works as a Lecturer at the Computer Science Dept. of the *Universitat de Lleida*, Spain. Currently, his research interests address the design of methodologies for the correct definition and extension of metamodels. <josepma@eps.udl.es>

One of the main requirements for a PML regarding standardization is that it should rely on a notation and semantics which are standard in the software industry, so that the chosen PML can be used without the burden of having to learn yet another notation and another software tool and also with the possibility to make it easily understandable to other software engineers.

UML (Unified Modeling Language) seems to be a natural candidate for such a standard process modelling formalism since it has become a standard *de facto* in the modelling of object-oriented systems and an important trend among the researchers of the field is to consider a software process itself as a piece of software [7].

The question that arises at this point is whether UML is able to deal with the specific requirements of SPM. The research on this topic began in the late nineties, in which several research groups tested the capabilities of UML as a modelling formalism in the related fields of software and workflow processes. A preliminary result of this research is that, although UML seems to be powerful enough to address the structural aspects of a process, it lacks some degree of expressiveness and flexibility in order to model its behavioural part (specially in the case of software processes in which the flexibility requirements are usually more important).

Therefore, UML should be adapted somehow in order to model software processes: not only should we add new constructs to deal with some behavioural issues but we should also introduce in the language the vocabulary of the field (*activities, artifacts, agents, tools, roles, etc.*) and its specific features. This language adaptation is usually referred to as *UML extension*.

In this article we introduce which mechanisms are provided by UML in order to extend the language and how an extended UML may be used in order to model the structural and behavioural aspects of a software process. Finally, we will outline some examples of proposals for using UML as a PML for software processes.

2 UML Extension Mechanisms

We have already introduced the necessity to extend the UML language in order to model software processes. There are two different ways in which we may do this:

2.1 By Means of An Explicit Extension of The UML Metamodel (Heavyweight Extension)

The UML metamodel [11] contains the definition of the elements that are used in a UML model (e.g., *class, association, dependency, generalization, etc.*). For instance, Figure 1 shows two UML classes (*Company* and *Employee*) and an association (*works-for*) between them. The precise definition of what a class and an association are is stated in the UML metamodel. In particular, the definition of the concept of *class* is given by means of the element of the UML metamodel called *Class*, whereas the notion of *association* is described by means of the element *Association* of the metamodel. The elements of the metamodel are called *metaelements* (e.g., metaclasses and metaassociations). Any UML model is an instance of the UML metamodel, which means that any element that comes up in a

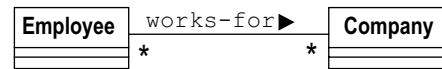


Figure 1: A Simple UML Model.

UML model is an instance of some metaelement (e.g., *Company* is an instance of *Class* and *works-for* is an instance of *Association*).

The ability of UML to model software processes may be enhanced by adding to the UML metamodel new metaclasses and metaassociations between them. For instance, if we need the concept of *Activity* in order to model specific tasks to be carried out during software development, we may add to the UML metamodel a metaclass for that purpose. In the same way, we may add other metaclasses to model the notions of *Role* (a specific responsibility within the process, e.g., *test engineer*) and *Document* (a product which is developed in an activity and which may be used in other ones) and some metaassociations to describe the relationships between these metaclasses (e.g., an activity generates a document; a role is responsible for an activity and also for a document). Figure 2 shows a fragment of this UML metamodel extension.

As a result, a software process model may be created with elements which are instances of the metaclasses *Document, Activity* and *Role*.

The main advantage of this alternative is that it constitutes a very elegant and powerful extension approach. A SPM built in this way is a proper instance of a well-formed UML extended metamodel. However, this approach does not result in UML-compliant models (i.e., they are not instances of the UML metamodel, but of an extended metamodel). This issue challenges standardization. For instance, how do we represent in a SPM instances of the new metaclasses in a UML-compliant way? How do we represent in the model the new features introduced by the metamodel (e.g., the role which is responsible for a specific activity)?

2.2 By Means of UML Profiles (Lightweight Extension)

A UML profile is an adaptation of the existing UML metaclasses by means of the built-in extension mechanism of UML so that they can fit in a specific domain (such as SPM). It is important to notice that UML profiles do not provide a first-class extension mechanism in the sense that they do not modify

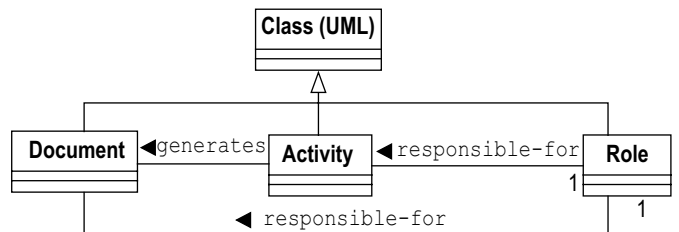


Figure 2: A Fragment of An Extension of The UML Metamodel.

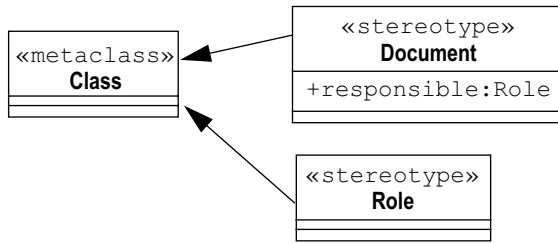


Figure 3: Stereotype Definition.

the UML metamodel. This specific extension mechanism provided by UML is based on the notion of *stereotype*. A stereotype defines how an existing metaclass may be adapted/extended. This extension may involve some of the following: a new name for that extended metaclass, new properties for that metaclass and/or new constraints.

UML supplies a notation to depict the extension provided by stereotypes in a model. For instance, Figure 3 shows the definition of the stereotypes *Document* and *Role* which extend the metaclass *Class* (the notation introduced by UML 2.0 has been used). The stereotype *Document* adds the property *responsible* of kind *Role*. As a result, it is possible to define in a specific SPM a class *SpecDoc*, with stereotype *Document* (which means that *SpecDocument* is a kind of *Document*) which has *SpecEngineer* as value of the property *responsible* (which means that the role class that is responsible for the documents of class *SpecDocument* is *SpecEngineer*). Figure 4 shows how this situation can be represented in a UML class diagram.

Since UML profiles do not modify the UML metamodel and follow the UML built-in extension mechanisms, they provide a full UML compliance. However, stereotypes do not have the same semantics as actual metaclasses and their expressiveness is poorer. Finally, this approach does not supply a representation for the extension at the metamodel level, which impairs the comprehensibility of the resulting model and of the profile itself. The notation introduced by UML 2.0 to define profiles contributes to reduce this comprehension problem but it does not eliminate it.

As suggested in the UML 2.0 infrastructure document, it is an ongoing matter of discussion whether to use the *heavyweight* or the *lightweight* approaches in order to extend the UML metamodel to fit it in a specific domain.

3 Modelling The Structural Part of A SPM

UML is a language intended to model object-oriented systems. The expressiveness of the object-oriented paradigm has proven appropriate to model the structural aspects of software processes [5, RF99]. Hence, the constructs offered by UML to model structural aspects of systems are globally suitable for this purpose. In particular, UML *class diagrams* may be used for modelling both software process elements and also relationships of different kinds among them.



Figure 4: A UML Model That Uses Stereotypes.

Software process elements may be defined as stereotyped classes. A different stereotype may be defined for each kind of software process element (documents, activities, roles, etc.). If necessary, some constraints and class properties may also be added to the stereotypes. Indeed, a complete UML profile can be defined. Figures 3 and 4 above show examples of the definition of some stereotypes that could take part in a profile intended to model software processes and also the use of one of those stereotypes in order to define a couple of classes of a specific SPM.

UML associations, generalizations and dependencies may be used to model the required relationships among classes. For the sake of an example, we mention here one specific structural requirement of software processes: *composition* of documents and activities.

Activities (also documents) may be atomic or composite. Composite activities are constituted of *smaller* (less complex) subactivities. In order to complete a composite activity, its subactivities should be carried out in a specific manner (which is stated in the behavioural description of the process; see Section 4). In their turn, these subactivities may be further decomposed in even smaller ones. This decomposition process can take place as many times as necessary. This can be modelled in UML by means of (possibly stereotyped) composite aggrega-

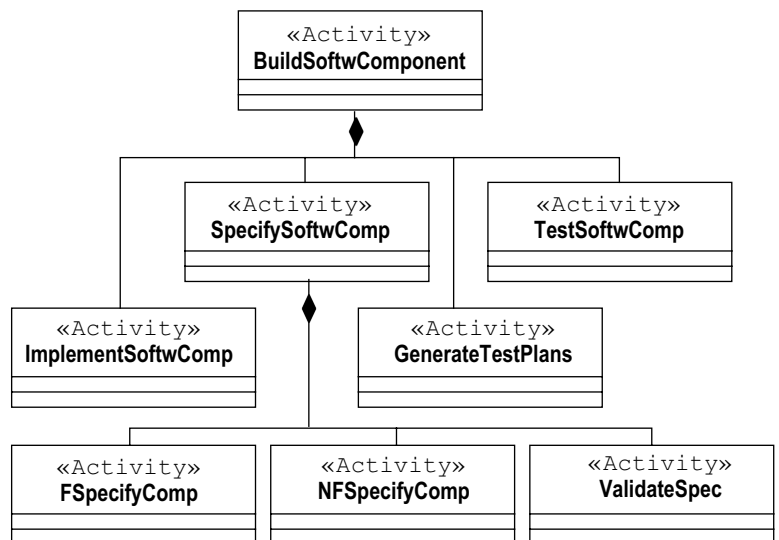


Figure 5: Activity Composition.

tions. Figure 5 shows how to model the process in which an activity class *BuildSoftwComponent* may be decomposed into the subtasks of *ImplementSoftwComp*, *SpecifySoftwComp*, *GenerateTestPlans* and *TestSoftwComp*. In its turn, *SpecifySoftwComp* is further decomposed.

Another interesting relationship between documents or activities is *refinement*, which can be modelled in UML through inheritance. Other relationships between software process classes may be modelled by stereotyping the general-purpose *dependency* UML relationship.

4 Modelling The Behavioural Part of A SPM

A SPM should describe, in addition to the software process structural aspects, the behaviour of such a process. This behaviour may be described using two different paradigms, namely *proactive control* and *reactive control*. Proactive control states the enactment of process activities according to a pre-established plan. Reactive control, in its turn, is based on describing the enactment of some actions as a response to events due to the occurrence of some condition.

PMLs should be expressive enough to offer both types of behaviour description. Let us show how UML may deal with both controls.

4.1 Proactive Control

There exist different approaches that a PML may follow in order to support proactive control. Probably, the most popular one consists in providing imperative descriptions of which activity is to be enacted after the end of a specific one. This approach can be modelled in UML by means of activity diagrams, in which a *transition* from activity *a* to activity *b* is triggered whenever activity *a* finishes its enactment. This transition grants the permission to start *b*.

Although this approach is possible, it often leads to a control flow description in terms of a set of activities which are enacted either following a strict sequence (possibly with conditions and loops, which have been incorporated to UML 2.0) or in a concurrent way (no other policy in between is allowed).

Usually, modelling software processes requires more flexible and expressive constructs: software developers tend to perform several activities at the same time and move from one to another depending on their interactions with the rest of the team or on some specific requirements (for instance, a deadline).

This reflection leads to other approaches to model the proactive behaviour of a software process. A popular one is the representation of different forms of precedence requirements between activities [8, 9]. For instance, instead of modelling something of the sort: *the implementation of a component will start once its specification has finished*, it seems better to overlap both activities to a certain extent: *the implementation of a component should begin some time after the starting point of its specification and should finish after the end of its specification*.

The second form of modelling uses *precedence relationships*, instead of pure transitions in order to describe behavioural relationships between tasks. A precedence relationship is stated between a set of source activities and another set of target ones and establishes in a *declarative* way which require-

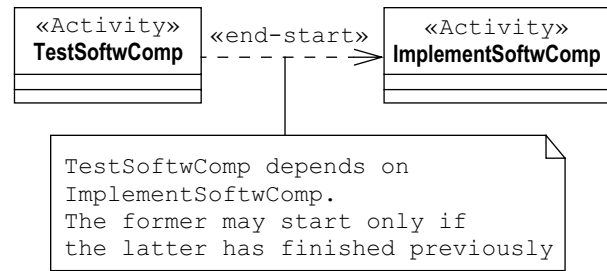


Figure 6: A Precedence As A Dependency.

ments concerning the state of the source activities are needed in order to start/finish the enactment of the target ones. Usually, the second modelling approach (using *precedence relationships*) provides not only a higher degree of freedom when enacting it, but also a more expressive, realistic, and less strict model.

The problem with precedence relationships is that UML does not support them in a direct way. Hence, a UML extension is necessary if such relationships are to be incorporated into the PML. A precedence relationship may be modelled in UML as a special kind of dependency between activities (see Figure 6).

4.2 Reactive Control

An expressive model for a software process should describe, not only the proactive plan that the process is supposed to follow during its enactment, but also its reaction to certain events that may occur during such enactment (e.g., a notification is received from the supervisor to abort the process or to start at once a specific activity).

The *reactive control* of a PML is responsible for expressing the behaviour of activities as a response to events. UML offers several powerful reactive elements that may be used for this purpose (i.e., *state machines*, *events*, *signals*, *actions*, etc.). However, it is quite usual that current PMLs in the field of software processes use some elements to model reactive control which are not explicitly defined in UML. This is the case of Event-condition-action (ECA) rules, which have become very popular in the context of PMLs in order to model reactive behaviour [2]. These rules establish the execution of an action as a response to an event in the case that a certain condition holds. ECA rules may be associated to model entities (e.g., activities). The introduction of ECA rules in a UML-based PML, or other reactive elements, may lead to an extension of

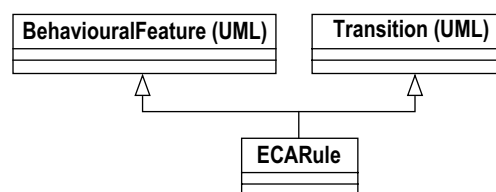


Figure 7: Definition of ECA-rules in PROMENADE (Fragment).

the UML metamodel. For instance, a UML-based PML called PROMENADE (see Section 5 for an outline of it) provides a definition of ECA rule both as a (UML) behavioural feature (so that it can be associated to an entity as one of its features) and as a (UML) transition (hence, it inherits the reactive behaviour modelled by transitions); see Figure 7.

5 Some Proposals To Model Software Processes with UML

Although most of the PMLs in the field of SPM that can be found in the literature are not based on UML, in the late nineties some approaches investigated the use of UML as a modelling formalism in the related fields of software and workflow processes. These approaches did not make an entire proposal of a PML but just focused on some of its elements. They shown some drawbacks of activity diagrams and proposed some alternatives to describe process behaviour. Some of them were quite unnatural (e.g., by means of class diagrams with stereotyped associations for showing the control and data flow [5]).

The next step was to propose an extension of UML to describe software processes. As we have seen, this can be achieved either by defining a UML profile or by means of a heavyweight extension.

The approach of defining a UML profile has been followed in the own UML definition (in versions prior to 2.0 [11]), which define a profile for software development processes. However, this profile is quite basic, since it only defines some stereotypes, no properties and almost no additional constraints. On the other hand, the defined stereotypes do not seem to be sufficient to deal with the terminology introduced by SPMs. Even more, no behavioural aspects are mentioned in this definition.

In the last few years, two new UML-based proposals (namely, SPEM [10] and PROMENADE [8, FR03]) have come up sharing virtually the same innovative definition approach: they are defined as heavyweight extensions of the UML metamodel but they overcome the limitations suffered by these kinds of extensions (namely, not full UML-compliance and, thus, no standard languages) by offering a transformation of the extended metamodel into a UML profile. This seems to be a good solution to keep the best of both worlds: while the heavyweight extension ensures a proper language definition at the metamodel level, its transformation to UML profile guarantees conformance with UML. Apart from this coincidence in the modelling approach, both proposals turn out to be quite different.

SPEM is an adopted specification of the OMG done by several companies with the purpose of modelling a family of software development processes (namely, RUP, SI Method, OPEN, etc.). A process in SPEM is defined by means of *iterations*, *phases*, *activities* and *steps*. Activities are not normally decomposed beyond steps, which are atomic (therefore, only two-levels of activity-subactivity decomposition are normally considered). Control-flow in SPEM is based on dependencies between activities (however, only *finish-start* and *finish-finish* dependencies have been defined). SPEM does not define reactive control-flow constructs. As a consequence of these features, SPEM does not seem to be designed to describe detailed processes.

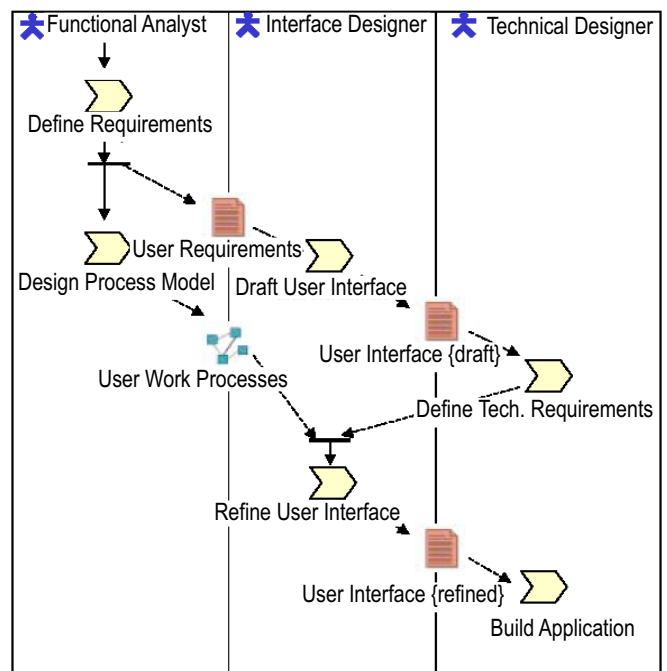


Figure 8: Use of Activity Diagrams in SPEM.

When the extended metamodel is transformed into an UML profile, some stereotypes like *Step*, *Guidance*, *Activity*, etc. do come up. Each of these stereotypes are given a suggested notation and may be used within usual UML diagrams in order to model the various aspects of a process. For example, Figure 8 shows an activity diagram to describe the behaviour of a process fragment.

PROMENADE is a PML which has been defined with the objectives of improving expressiveness, flexibility, standardization and modularity in SPM construction. Remarkably, it allows the modelling of complex and/or detailed software processes in an expressive way by providing both proactive and reactive ways to express model behaviour. Proactive control relies on the notion of *precedence relationship*, which has been defined in the framework of UML as a special kind of dependency. Different families of such precedence relationships have been defined. Furthermore, the modeller may define his/her own new kinds of precedence relationships. In its turn, reactive control is based on ECA rules. Figure 6, above, shows a very simple example of a UML-compliant representation of a PROMENADE precedence relationship between two activities.

6 Conclusion

One important challenge for the software process modelling research community is the development of standard notations to describe software processes. In all probability, such standardization will contribute both to disseminate the research results in software process technology all over the software engineering community (including, software industry) and to incorporate the formalization of software processes into the

SPA&SPI initiatives. UML seems a good choice for such a standard notation since it has become a standard *de facto* in the modelling of object-oriented systems. While UML offers enough expressiveness to model structural aspects of software processes, some authors argue that more constructs should be added to the language in order to model behavioural ones (e.g., proactive and reactive aspects). For this reason (and also to define properly the specific concepts of software processes in this language) UML-based PMLs should extend UML. Two possibilities do exist to do that: the *lightweight* and the *heavyweight* extensions. The former defines a UML profile and the latter, performs an explicit extension of the UML metamodel.

Several proposals have been defined in the last three or four years in this sense: from quite basic UML profiles to more sophisticated approaches like SPEM or PROMENADE.

It is worth mentioning that the use of UML as the basis to construct PMLs will lead to standardization in the sense that the defined PML will rely on a well-known syntax and semantics, and it will be possible to use widespread UML-based tools in order to model software processes. However, it is also true that different PMLs may use UML in different ways (e.g., they may use different constructs or the same constructs to model different aspects). In any case, the use of UML will constitute a qualitative improvement regarding SPM standardization.

References

- [1] J. - C. Demiamé, B. A. Kaba, D. Wastell (eds.). Software Process: Principles, Methodology and Technology. Lecture Notes in Computer Science, Vol. 1500. Springer-Verlag, Berlin Heidelberg New York, 1999.
- [2] A. Finkelstein, J. Kramer, B. Nuseibeh. Software Process Modelling and Technology. Advanced Software Development Series, Vol. 3. John Wiley & Sons Inc., New York Chichester Toronto Brisbane Singapore, 1994.
- [3] X. Franch, J. M. Ribó. Using UML for Modelling the Static Part of a Software Process. In Proceedings of UML '99, Forth Collins CO (USA). Lecture Notes in Computer Science (LNCS), Vol. 1723, pp. 292–307. Springer-Verlag, 1999.
- [4] W. S. Humphrey. Managing the Software Process. SEI Series in Software Engineering. Addison Wesley, 1989
- [5] D. Jäger, A. Schleicher, B. Westfechtel. Object-Oriented Software Process Modelling. Proceedings of the 7th European Software Engineering Conference (ESEC), LNCS 1687 Toulouse (France), September 1999.
- [6] X. Franch, J. M. Ribó. A UML-based Approach to Enhance Reuse within Process Technology 9th International Workshop, EWSPT 2003, Helsinki, Finland. Lecture Notes in Computer Science (LNCS), Vol.2786. Springer-Verlag, 2003.
- [7] L. Osterweil. Software Processes are Software Too. In Procs. of the Intl. Conf. on Software Engineering (ICSE-9), 1987.
- [8] J. M. Ribó, X. Franch. A Precedence-based Approach for Proactive Control in Software Process Modelling. In Proceedings of the SEKE-2002 conference (Software Engineering and Knowledge Engineering). ACM Press. Ischia (Italy). September, 2002.
- [9] C. Schlenoff, M. Gruninger et al. The Process Specification Language (PSL) Overview and Version 1.0 Specification. NIST Internal Report (NISTIR) 6459, 1999.
- [10] Software Process Engineering Metamodel, version 1.0. Adopted specification formal/2002-11-14 of the OMG (accessible at <http://www.omg.org>).
- [11] Unified Modelling Language, version 1.5. Adopted specification formal/03-03-01 of the OMG (accessible at <http://www.omg.org>). Also, UML2 Infrastructure Final Adopted Specification and UML 2 Superstructure Final Adopted Specification (accessible at <http://www.omg.org>).