Towards an Integration of Process Modeling and Project Planning

Michael Gnatz, Martin Deubler, Michael Meisinger
Technische Universität München
Institut für Informatik
Boltzmannstr. 3, 85748 Garching
(gnatzm|deubler|meisinge)@in.tum.de

Andreas Rausch
Technische Universität Kaiserslautern
Gottlieb-Damiler-Straße
D-67653 Kaiserslautern
rausch@informatik.uni-kl.de

Abstract

Most development projects have very complex dependencies regarding the tasks to accomplish. Process models offer the chance to incorporate the knowledge of many project managers into active projects. Bridging the gap between process models and project plans by defining such models precisely seems to be beneficial.

In this paper, we show the benefits and highlight some of the interesting problems of integrating process modeling and project planning. We introduce metamodeling techniques to constrain the instantiation of the process model, so that structural aspects of the process plan can be derived. A small, consistent example is used throughout the paper to illustrate our approach.

1. Introduction

Despite great progress in the field of software engineering, many IT projects are not regarded as successful. Either they are canceled before completion or they overrun budget, are late, or deliver fewer features than originally promised [10]. Reasons are manifold, but certainly most of them are non-technical.

In particular, project management and project planning are crucial to a project. Most development projects have very complex dependencies regarding the tasks to accomplish, that less experienced project managers often underestimate. Well-defined and repeatable development processes are one building block of a successful project. Such processes ease project planning by providing a model of clear milestones, descriptions of activities to perform and document templates for writing all kinds of specifications needed in a development project. Defined and therefore repeatable processes offer the chance to incorporate the knowledge and the lessons learned of many seasoned project managers into active projects [3].

Since defined processes have to be reusable in different project contexts to be profitable, processes are defined as process models, which are abstractions of concrete processes. Despite this abstraction, we expect process models to be easy to use, adaptable to the needs of the specific project, and not demanding too much effort to learn. In practice however, many standardized process models have great weaknesses regarding usability [2]. An example is the German V-Modell 97 ([12], [13]), which is currently updated ([14]).

Adapting and using process models is often regarded as overhead and tedious work. Process models like the V-Model are regarded as a piece of inspiring literature, which is read once by the project leader at the beginning of the project and then is usually forgotten. Improving this situation requires some kind of automation support. “Using” a process model can mean to derive an automated workflow process from a process model description. However, we believe that the benefit of enacting development processes this way is limited. Development processes have not much in common with industrial manufacturing processes, but are unique and demand creativity.

In contrast, our focus is on an iteratively adapted project plan as a process model’s outcome, which is carried out “manually” by people. According to its nature, project planning is an iterative task. Effective planning requires that process models are present during the whole life cycle of the project. Every adaptation of the project plans should consider the process model. In order to increase the benefits of current process models, people involved in a project must immediately realize how a process model influences the project. Consequently, a process model must include descriptions or even formal definitions that make the coherence of the process model with a project plan obvious and straightforward.

Bridging the gap between process models and project plans by defining such models precisely seems to be beneficial. This paper is investigating the idea of deriving project plans from process models, given a specific project. Our approach focuses on describing how to derive a so-called structural project plan from a process model. A structural project plan contains instances of a process model’s activities and products, and the logical dependencies between them. Instantiation can mean multiple instantiation as for example in the case of an activity “Implement Component”. The project manager’s task is to

tailor the process model (by selection of process model elements) as well as to plan multiple occurrences of the process model's activities and products (by instantiation of the chosen elements), as shown in Figure 1. Time and resource planning of activities or the determination of critical path tasks for example, are not in our scope.

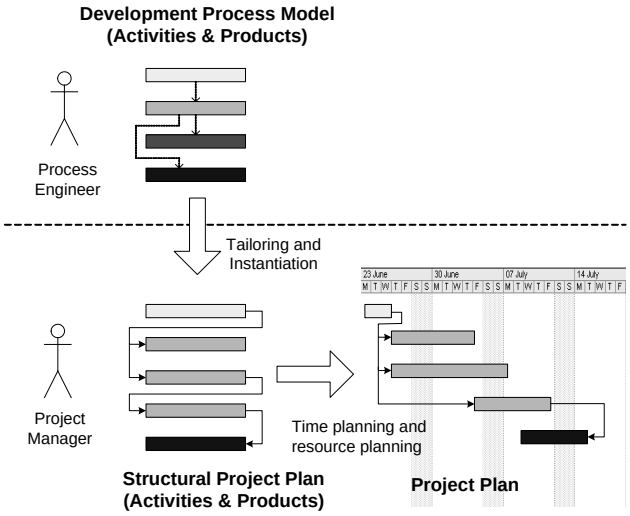

Figure 1. Deriving Project Plans from Process Models

The type of process models considered here is intended to facilitate planning in order to make projects more predictable, as for example the German V-Modell 200x [14] is supposed to do. Such process models provide more like a management view of the development process than a developers view, as opposed to others like the Rational Unified Process ([4]), for example.

In this paper, we show the benefits and highlight some of the interesting problems of integrating process modeling and project planning. We introduce metamodeling techniques to constrain the instantiation of a process model. On this basis, we are able to introduce techniques for deriving the structural aspects of a project plan from a process model, which conforms to the process metamodel. A small, consistent example is used throughout the paper to illustrate our approach. The focus of the paper is to show the idea and its potential in general, thus providing a basis for further research.

The organization of the rest of the paper is as follows: Section 2 illustrates the problem by presenting an exemplary process model and an exemplary structural project plan. Section 3 shows our modeling approach intended to solve the problem in general. Section 4 gives examples for concrete models according to our modeling approach. Section 5 provides an overview of related work. A conclusion and outlook on further work in section 6 ends the paper.

2. Problem statement

Consider an insurance company that has started a development project for a new insurance management system – code-name "eInsurance System 200x". Like almost all companies our insurance company provides a standardized software development process which has to be tailored for the specific needs of the project "eInsurance System 200x".

Figure 2 shows a small cutout of the already tailored process model, where the project manager has already chosen the products and activities that are of interest in the context of his project. The process model consists of three activities, namely *Design*, *Implementation* and *Integration Test* and the involved work products, respectively.

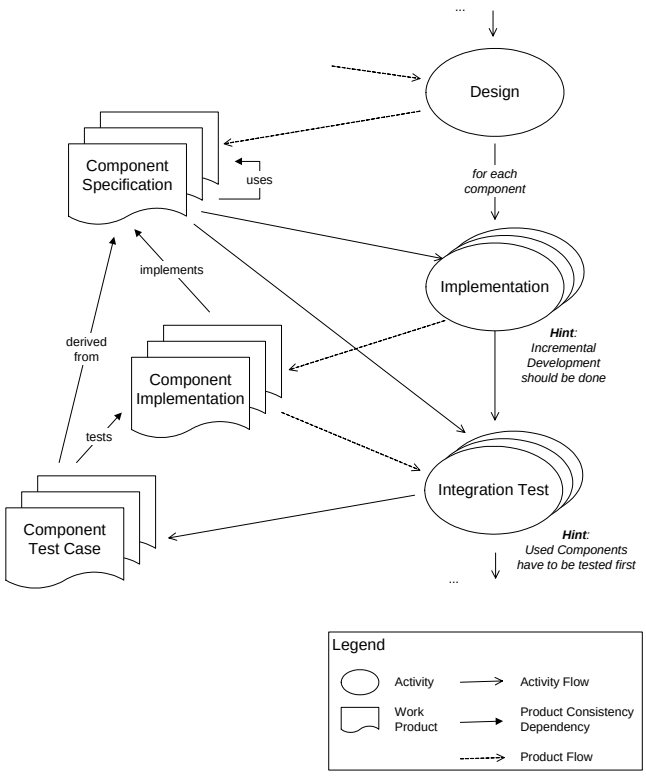

Figure 2. Cutout of the Company's Software Development Process Model [1]

Whereas, as shown in Figure 2, the *Design* activity should be performed once, *Implementation* and *Integration Test* are intended to be executed several times, that is for each *Component Specification* provided by the *Design*. The model additionally states that *Implementation* should be done incrementally, meaning that the imple-

[1] For this and the following figures, references to elements of the figure in the explaining text are set in *italics*.

mentation (and the delivery as well) of chosen components should be done one after the other regarding the activity flow. A constraint provided in form of a hint is that the *Integration Tests* of a particular component cannot be done before the components used by this component have been tested. Please note, that consistency dependencies between work products, as for example the *uses* relationship between *Component Specifications* or the *derived from* dependency between *Component Specification* and *Test Case* are included in the process model as well.

Since the detailed deliverables evolve while the project progresses, project planning is an iterative task. Having a first rough idea of the system to develop in terms of components and their dependencies the project manager can start with the planning of implementation and test of these components. An exemplary component architecture for the "eInsurance System 200x" is shown in Figure 3.

According to the component architecture of the system, the process model suggests a corresponding set of deliverables (instances of the process model's work products) as shown in the structural project plan in Figure 4 (a). The *Business Layer Specification* is directly associated to the corresponding *Business Layer Implementation* and to several *Business Layer Test Cases*, whereas the *Business Layer Implementation* and the *Business Layer Test Cases* are further associated to each other.

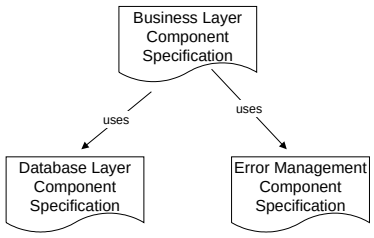

Figure 3. Exemplary Component Architecture

Figure 4 (b) shows an alternative instantiation of products in a structural project plan that is not a valid instantiation of the process model in Figure 2. Please note, that from a formal point of view it is not clear that Figure 4 (a) is a valid instantiation while Figure 4 (b) is an invalid instantiation.

Next to determining the structure of the work products, the exemplary component architecture has consequences on the flow of activities according to the process model. Of course, corresponding activities have to be included in the structural project plan for each identified product, as shown in Figure 5 (a).

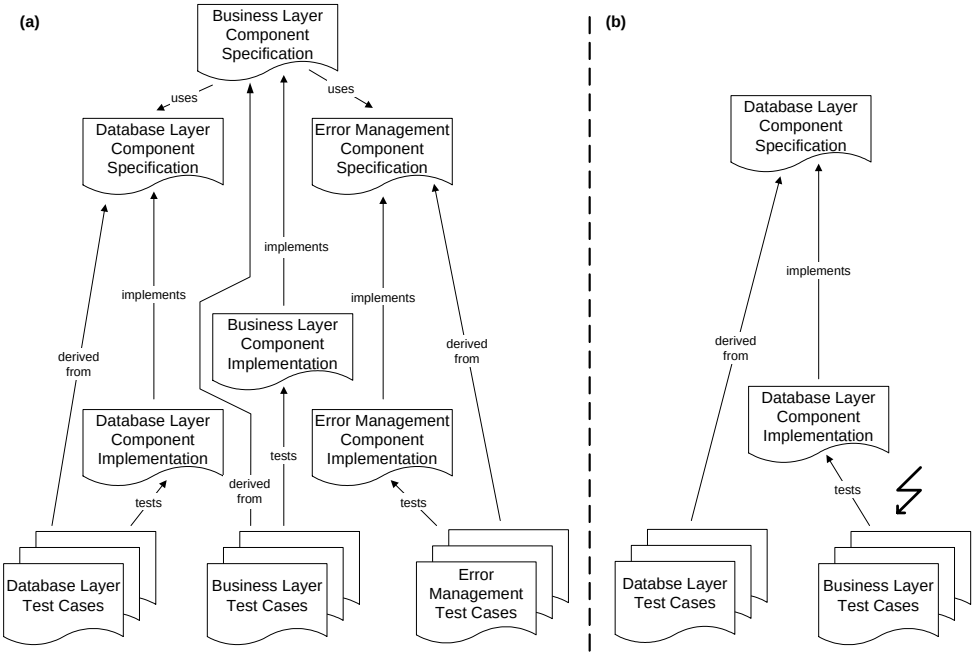

Figure 4. Structural Project Plan Showing Work Products: (a) Valid Instantiation, (b) Invalid Instantiation

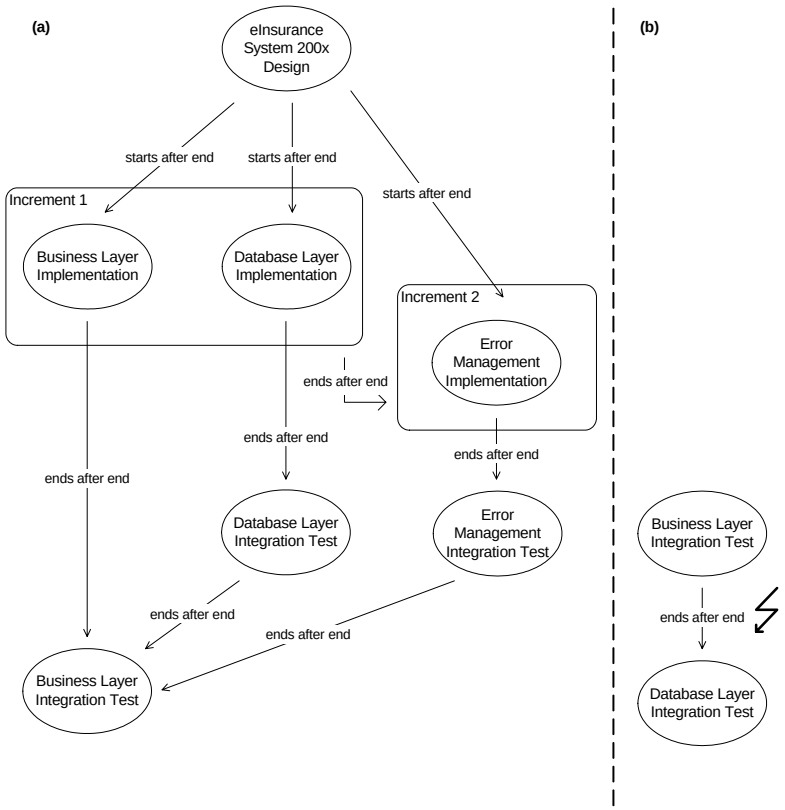

Figure 5. Structural Project Plan Showing Activities: (a) Valid Instantiation, (b) Invalid Instantiation

According to the process model in Figure 2, in the structural project plan in Figure 5 (a) the *Business Layer Implementation* has to start after the end of the *eInsurance System 200x Design*. However, the fact, that the *Business Layer Integration Test* has to end after the *Database Layer Integration Test* is just mentioned in form of a hint in the process model in Figure 2. This ordering is nevertheless required, because the *Business Layer Implementation* has a *uses* dependency to the *Database Layer Implementation*, as defined by the component architecture in Figure 3. In contrast, Figure 5 (b) does not provide a valid instantiation of the process model from Figure 2.

Let us also assume that the project manager decides to build the "eInsurance System 200x" in two increments. The first increment contains business and database layer while the second increment contains the error management. As shown in Figure 5 (a), the *Error Management Implementation* is starting after the first increment has been finished (which in practice is not the best idea but suits us here).

The problem is that the process model does not contain all necessary information in terms of a formal description. This means that a derivation of a structural project plan cannot be done automatically. In our example, the required inputs seem to be the system's component architecture and the assignment of the components to increments. For a clever project manager it is easy to keep an overview of the "eInsurance System 200x" project example. However, for projects more realistic in size, as well as for process models more realistic in size, building a project plan, which is consistent to the company's standardized development process model, will get very complex and tedious.

3. A Layered Modeling Approach

Standardized software development process models are quite different in practice and usually contain specific solutions for certain types of projects and organizations. Therefore, providing a single process model, which allows for the consistent instantiation of a project plan, would to be a little contribution. Furthermore, process models such as the V-Modell 97 [12] for example, which can be adapted to almost every project, are so generic that their usefulness as a guideline for a concrete project might be doubted.

Our modeling approach, depicted in Figure 6, is to provide a *Process Metamodel*, which establishes a com-

mon language for describing *Process Models*. We use a layered modeling approach according to the metamodel structure provided in [8]. A process engineer can use the language defined by the *Process Metamodel* to describe a company's specific software development process model (see also [3]). The *Process Metamodel* offers clear definitions for terms like activity or work product and their possible relations.

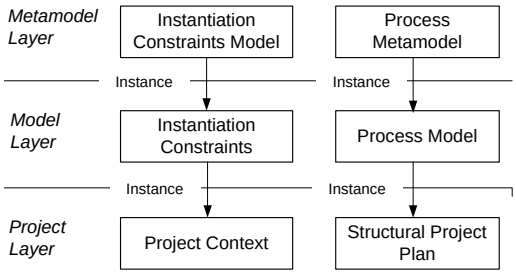

Figure 6. Overview of the Proposed Layered Modeling Approach

A *Process Model* is an instance of a *Process Metamodel*, providing for example an "implementation" activity and an "implementation" work product. The *Process Model* serves as a model for the *Structural Project Plan*, since according to our approach the *Structural Project Plan* itself is an instance of the *Process Model*. Hence, the *Process Metamodel* has to specify the elements and their relationships in the *Structural Project Plan* as well, since these are instances of instances of the *Process Metamodel*. For example, the *Structural Project Plan* might contain an activity "business layer implementation" as well as an activity "database layer implementation", which are both instances of the "implementation activity" of the *Process Model* (, which in turn are instances of the activity in the *Process Metamodel*).

The instantiation of the *Process Model* involves multiple instantiations of certain elements, according to the *Project Context*, which contains for example the concrete project's component architecture. On the model layer, the process engineer has to provide *Instantiation Constraints* for the *Process Model*. The *Project Context* is an instance of these *Instantiation Constraints*. On the metamodel layer, there is an *Instantiation Constraints Model* to provide a model (or language constructs) for the process engineer of how to constrain the instantiation of the *Process Model*.

Integrating process modeling and project planning is the basis for a tool-supported derivation of a structural project plan from a process model. In our approach, the coherence of process model and structural project plan is made explicit by specifying the language as well as the instantiation constraints for process models in form of a metamodel layer. Process engineers using this metamodel layer have a common language to take care of the applicability of their process models for automatically deriving a structural project plan.

In order to be able to refer to instances of instances according to our modeling approach in the following text, we use the term "instances" for entities in the model layer, and the term "occurrences" for entities in the project layer in the rest of the paper.

4. Process Models and Project Plans

This section provides examples according to our layered modeling approach. Section 4.1 shows our exemplary metamodel layer; section 4.2 shows a corresponding process model layer and section 4.3 a possible project layer. The examples provided here are the same in content as the examples introduced in section 2.

4.1 Exemplary Process Metamodel Layer

Figure 7 shows according to our modeling approach an *instantiation constraints model* and a *process metamodel*. The upper part of Figure 7 contains an abstraction (depicted as classes with UML stereotype <<*abstract*>>) from the concrete metamodel in the lower part of the figure.

From an abstract point of view a *process metamodel* consists of *model classes* that are related by binary, directed *model associations*. *Model associations* can be of certain types, establishing particular consistency constraints for valid occurrences of *model associations* in the plan. For example, an association of type *one-all* states that an occurrence of this *model association* in the project plan relates one occurrence of a *model class* with all occurrences of associated *model classes* in the plan.

Planning units in the *instantiation constraints model* are the inputs needed for deriving a structural project plan from the process model automatically. They determine the multiple instantiation of *Model Concepts* in the structural project plan. Each *planning unit* has only one instance in the process model (depicted as classes with stereotype <<*singleton*>> in Figure 7), but several occurrences in the structural project plan. *Planning units* are *composed of model classes* and *model associations*. The intended semantics is that for one occurrence of a *planning unit* in the project plan there has to be at least one occurrence of all *model classes* and *associations* the *planning unit is composed of.*

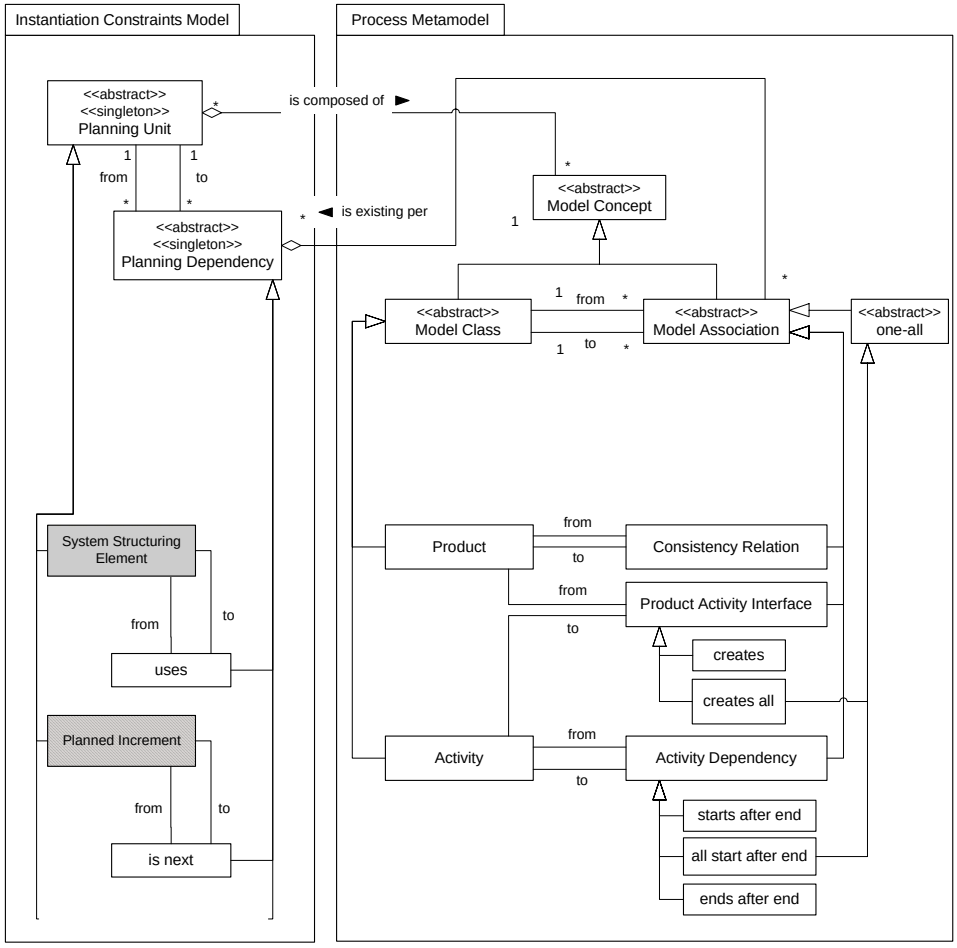

Figure 7. Exemplary Instantiation Constraints Model and Process Metamodel

Planning units are related by *planning dependencies*, which in turn can influence the occurrence of certain *model associations* in the plan, as expressed by the *is existing per* association in Figure 7. A simple semantics is that for every occurrence of a *planning dependency* there is one occurrence of a *model association* in the structural project plan. In general, the semantics has to be more complex, since for example for a *planning dependency*, there might be a *model association* in the reverse direction as well.

A concrete *process metamodel* is shown in the lower part of Figure 7 as a specialization of the abstract classes in the upper part of the figure. In our exemplary metamodel concrete *model classes* are *products*[2] and *activities*. *Products* have *consistency relations* among each other, which are concrete *model associations*. *Activities* have *activity dependencies*, like for example *starts after end*. *Activities* can *create* a *product, which is* determined by the *product activity interface*. Please note, that the *Process Metamodel* shown here provides just a very simple language.

Examples for concrete *planning units* in the *instantiation constraints model* are the *system structuring element* and the *planned increment*. *System structuring elements* are architectural elements of the system like software components. Occurrences of these have to be identified by the system designer and communicated to the project manager, since the (possibly multiple) occurrence of certain *products* and *activities* in the plan is determined by the architecture of the system. Similarly, *planned increments* and their order are an input for the automatic derivation of the plan from the process model. Other useful *planning units* not shown in Figure 7 might be "subprojects", for example, since certain project management activities occur per subproject. Another example are "reporting intervals", since some products as status reports to the customer, for example, occur on a regular temporal basis.

[2] *Product* is an abbreviation for "work products".

Planning units provide a language construct for the process engineer to specify the multiplicity of *products*, *activities* and their relationships according to a fixed set of multipliers. Please note that leaving the decision of multiple instantiation of certain process model elements totally open to the process engineer would just lead to providing cardinality constraints for the associations contained in the process model. In contrast, in our approach the concept of *planning units* is intended to serve the process engineer as a methodological guideline when building process models.

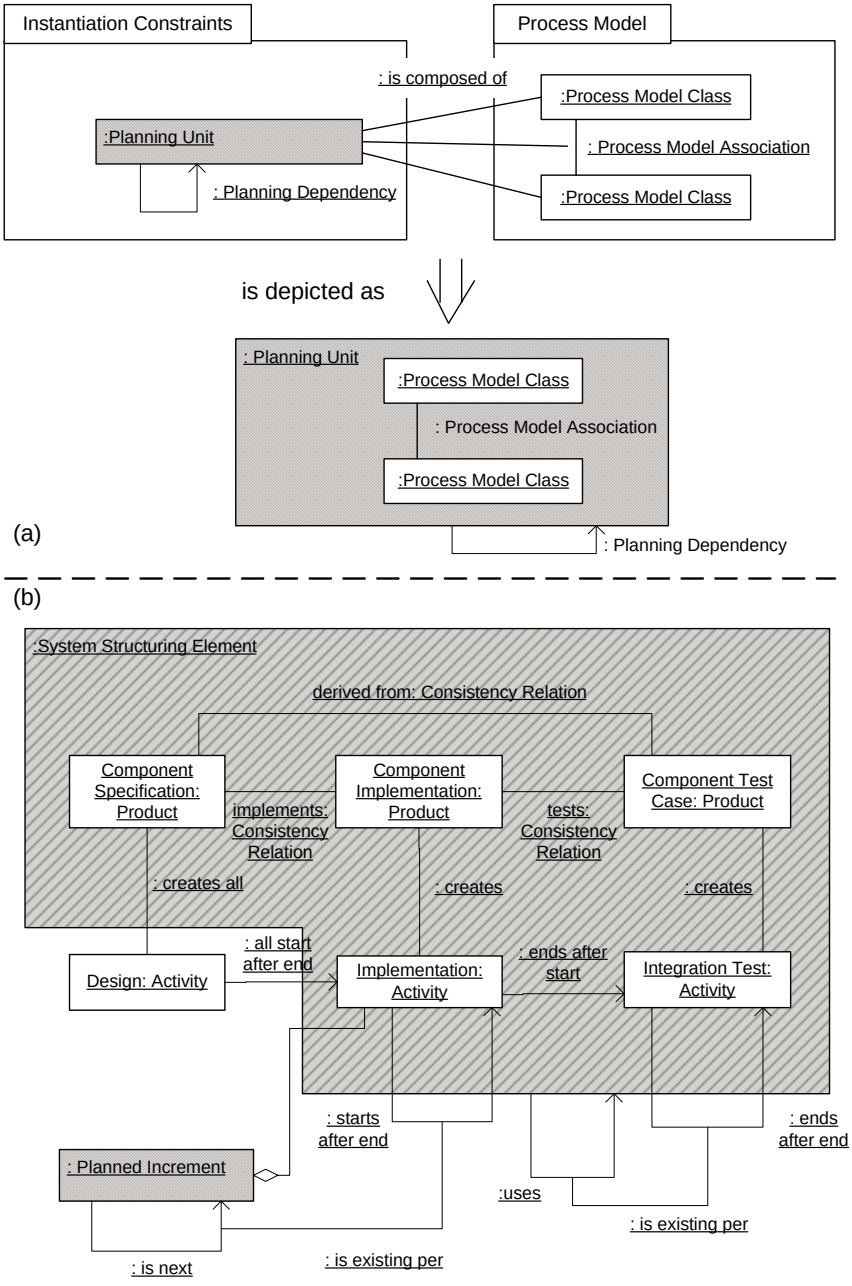

Figure 8. (a) Integrated Representation of Instantiation Constraints and Process Model, (b) Exemplary Instantiation Constraints and Process Model

4.2 Exemplary Process Model Layer

This section shows an exemplary process model and exemplary instantiation constraints. Figure 8 (a) explains the integrated representation for both models, which is used to represent both models in Figure 8 (b). *Planning units* being *composed of process model classes* and *associations* are represented as surrounding boxes. This notation has the same meaning as an UML composition association while making the diagram in Figure 8 (b) more legible.

Please note that the process model in this section is intended to be the same in content as the process model in Figure 2, but its notation is according to our modeling approach more formal. The *process model* is an instance of the process metamodel in Figure 7, the *instantiation constraints* are an instance of the instantiation constraints model.

The exemplary process model contains products and corresponding activities for specification, implementation, and testing. Products are related by consistency relations, not different from those in Figure 2, as for example the *derived from* relation between *component test case* and *component specification*. In Figure 8 (b), all *products* are included in the *system structuring element*, that is the *system structuring element* is *composed of* them.

In contrast to that, occurrences of the *ends after end* dependency exist between occurrences of *integration test* activities associated to different occurrences of *system structuring elements* in the plan. In Figure 8 (b), this is expressed by an *existing per* association between the planning dependency *uses* and the activity dependency *ends after end*. The meaning is the following: For each occurrence of *uses* between two *system structuring elements*, the *integration test* activities contained in the *system structuring elements* are related by *ends after end*.

Similarly, *planned increments*, which are *composed of implementation* activities together with their *is next* associations determine the *starts after end* dependencies between the *implementation* activities contained in these *increments*.

The *design* activity is the only activity not included in a *planning unit*, because according to our exemplary process model there can be just one occurrence of the *design* activity in a plan. This occurrence of the *design* activity *creates all* occurrences of *component specifications*. All occurrences of *implementation* activities start after the *design* activity ends. This is expressed by the *creates all* and *all start after end* associations, respectively, which are both instances of one-all model associations in the process metamodel.

4.3 Exemplary Structural Project Plan Layer

Figure 9 shows an integrated representation of a structural project plan and a project context. Both are instances of the process model and the instantiation constraints of the process model layer in Figure 8 (b). Please note, that occurrences of products and consistency relations are part of a structural project plan as well, but not shown here.

The surrounding boxes in Figure 9 are part of the project context, that is they represent occurrences of system structuring elements (*Business Layer*, *Database Layer* and *Error Management*) and planned increments (*Stage 1* and *Stage 2*), respectively. The structural project plan here is the same in content as the one presented in section 2.

In our example, the occurrences of *system structuring elements* and their *uses* relationship, as well as *planned increments* and the *implementation* activities included in them are the only input needed for the derivation of the structural project plan from the process model.

Since in our modeling approach the process metamodel layer does constrain the process model layer and the project layer as well, the metamodel does not only have to specify UML cardinalities for the model, but also for the project layer. For example, consider the *is composed of* association between *planning unit* and *model concept* in the metamodel in Figure 7. In the metamodel, the specified cardinality is many-to-many, because a *planning unit is composed of* many *model concepts* and vice versa. All instances of the *is composed of* association in the process model must be of cardinality one-to-at-least-one, that is for every occurrence of a *planning unit* there has to be at least one occurrence of a *model class* in the plan and for every occurrence of a *model class* there is exactly one occurrence of a *planning unit*. Similar consistency constraints have to be provided for the *existing per* association and the *all start after end* association in the metamodel, for example.

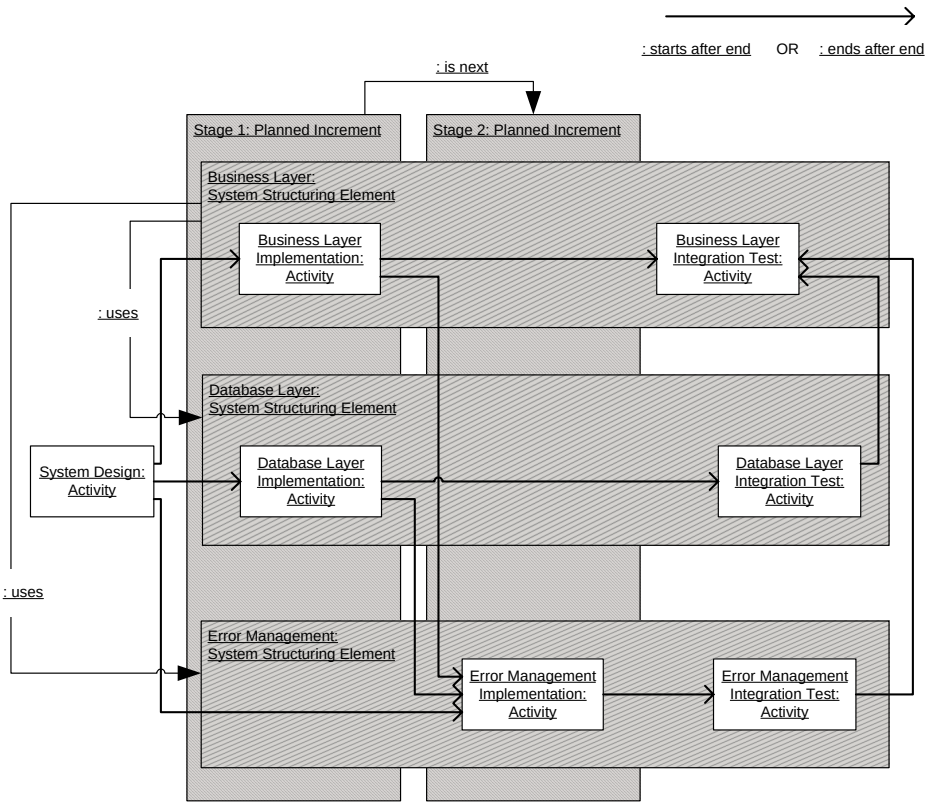

Figure 9. Exemplary Project Context and Structural Project Plan showing Activities

5. Related Work

Meta-case tools approaches as for example Maestro [7] tried to support the development process by providing a kind of workflow support based on tracking document-processing states. Similarly, the approach of process programming in [9] or "process sensitive engineering environments" as for example [1] based on Petri nets tried to automate as much of the development process as possible. More recent approaches like [6] or [15] for example are aiming at flexible workflow support tools for development processes as well.

In contrast, we believe that the benefit of enacting development processes is very limited. Development processes have not much in common with industrial manufacturing processes, but are unique and demand creativity. Our focus therefore is on an iteratively adapted project plan as a process model's outcome, which is carried out "manually" by people. The granularity of planned activities is much more abstract and less constraining than single steps to be carried out in a workflow engine.

Other approaches to process metamodeling like [3], [11] or [10] do not provide precise semantics in terms of a mapping of process model elements to a projects' planning level. Furthermore, commercial tools like [5] for deriving a project plan from a process model are based on the implicit meta-model of the [12]. With [5] multiple instantiation of process model elements has to be done manually by the project manager.

6. Conclusion and Further Work

In this paper, we showed the usefulness and some of the problems of integrating process modeling and project planning. We presented our layered modeling approach and provided an exemplary process metamodel together with an instantiation constraints model, which seem to be appropriate to solve the problems according to our example. By illustrating the usefulness of deriving project plans and the problems arising, we tried to lay the foundations for some further research. The solutions presented in this paper serve as a useful first step.

Interesting questions for the appropriateness of the approach are, whether a process model suited for deriving a structural project plan might be too complicated to elaborate. Of course, on the other hand, the question arises, if planning according to the restrictions that a process model provides might be too rigid. To provide an answer to the first question, the details of a process metamodel suitable for planning have to be elaborated. To give a positive

answer for the second question the concept of tailoring of the process model to the current needs of a project has to be integrated in the approach.

Next to the question of tailoring, several indispensable features have not been considered yet. Additional planning units need to be elaborated, as for example for subprojects. Not only simple incremental development, but also life cycle models like iterative development have to be looked at in detail. Furthermore, process models usually contain “cross-sectional” themes as quality assurance or configuration management, for example. Quality assurance activities like “assess product” are usually not written several times (for each product) in a process model, but just once in order to avoid redundancy in the process model. Nevertheless, these “generic” activities have to be considered when doing project planning, since quality assurance for example needs to be planned properly.

7. References

[1] S. Bandinelli, A. Fugetta, C. Ghezzi. Software Process Model Evolution in the SPADE Environment. IEEE Transactions on Software Engineering. 1993.

[2] M. Deubler, M. Gnatz, M. Meisinger, A. Rausch. Analyse-Workshop Ergebnisse. Workshops mit V-Modell 97 Anwendern. http://www.v-modell-200x.de. 2003.

[3] M. Gnatz, F. Marschall, G. Popp, A. Rausch, W. Schwerin. The Living Software Development Process. Software Quality Professional, Volume 5, Issue 3, June 2003.

[4] Philippe Kruchten. The Rational Unified Process, An Introduction, Second Edition. Addison Wesley Longman Inc. 2000.

[5] microTool: in-Step - The Workflow Management System for IT projects. http://www.microtool.de/instep/en/

[6] F. Maurer, B. Dellen, F. Bendeck, S. Goldmann, H. Holz, B. Kötting, M. Schaaf. Merging Project Planning and Web-Enabled Dynamic Workflow Technologies. IEEE Internet Computing. Vol. 4, No. 3 2000.

[7] G. Merbeth. Maestro II – das integrierte CASE-System von Softlab. In: CASE – Systeme und Werkzeuge (Hrsg. H. Balzert). 1993.

[8] Object Management Group (OMG). 1999. Meta Object Facility (MOF) Specification. http://www.omg.org, document number: 99-06-05.pdf.

[9] L. Osterweil. Software processes are software too. Proceedings of the 9th international conference on Software Engineering. 1987.

[10] Josep M. Ribo, Xavier Franch. PROMENADE: A PML Intended to Enhance Standardization, Expressiveness and Modularity in Software Process Modelling. Research Report. Departament de Llenguatges i Sistemes Informàtics. 2000.

[10] Standish Group International, Inc. 2001. Collaborating on Project Success. Software Magazine, February/March 2001. Wiesner Publishing. 2001.

[11] Software Process Engineering Meta-model (SPEM), Version 1.0. Object Management Group (OMG). http://www.omg.org/technology/documents/formal/-spem.htm. 2002.

[12] AU 250, Entwicklungsstandard für IT-Systeme des Bundes, Vorgehensmodell. Juni 1997.

[13] Entwicklungsstandard für IT-Systeme des Bundes, Vorgehensmodell. Teil 3: Handbuchsammlung: Tailoring und projektspezifisches V Modell. Juni 1997.

[14] Projekt WEIT - Weiterentwicklung des Entwicklungsstandards für IT-Systeme des Bundes auf Basis des V-Modell-97. http://www.v-modell-200x.de. 2003.

[15] B. Westfechtel. Ein graphbasiertes Managementsystem für dynamische Entwicklungsprozesse. Informatik Forschung und Entwicklung, Vol. 16. Springer-Verlag. 2001.