METHOD MANUAL BASED PROCESS GENERATION AND VALIDATION

Peter Killisperger
Competence Center Information Systems, University of Applied Sciences - München, Germany
Advanced Computing Research Centre, University of South Australia, Adelaide, Australia

Georg Peters
Department of Computer Science and Mathematics, University of Applied Sciences - München, Germany

Markus Stumptner
Advanced Computing Research Centre, University of South Australia, Adelaide, Australia

Thomas Stückl
System and Software Processes, Siemens Corporate Technology, München, Germany

Keywords: Software process, Instantiation, Automation.

Abstract: In order to use software processes for a spectrum of projects they are described in a generic way. Due to the uniqueness of software development, processes have to be adapted to project specific needs to be effectively applicable in projects. Siemens AG has started research projects aiming to improve this instantiation of processes. A system supporting project managers in instantiation of software processes at Siemens AG is being developed. It aims not only to execute instantiation decision made by humans but to automatically restore correctness of the resulting process.

1 INTRODUCTION

Explicitly defined software processes for the development of software are used by most large organizations. At Siemens, business units define software processes within a company-wide Siemens Process Framework (SPF) (Schmelzer and Sesselmann, 2004). Because of their size and complexity, they are not defined for projects individually but in a generic way as reference processes for application in any software project of the particular business unit.

Due to the individuality of software development, reference processes have to be instantiated to be applicable in projects. That is, the generic description of the process is specialized and adapted to the needs of a particular project. Until now, reference processes are used as general guideline and are instantiated only minimally. A more far reaching instantiation is desirable, because manual instantiation is error-prone, time consuming and thus expensive due to the complexity of processes and due to constraints of the SPF. Siemens defined an improved instantiation to comprise tailoring, resource allocation and customization of artifacts. The latter is the individualization of general artifacts for a project and their association with files implementing them.

In order to reduce effort for instantiation considerably, tool support has to be extended. We have developed a flexible architecture for systems that execute instantiation decisions made by humans and automatically restore correctness of the resulting process. We define a process to be correct when it complies with the restrictions on the process defined in a method manual. A method manual is a meta model defining permitted constructs in a process, derived from organizational restrictions (e.g. SPF).

In this paper we describe the developed architecture for instantiation systems executing decisions

Killisperger P., Peters G., Stumptner M. and Stückl T. (2009).
METHOD MANUAL BASED PROCESS GENERATION AND VALIDATION.
In *Proceedings of the 11th International Conference on Enterprise Information Systems - Information Systems Analysis and Specification*, pages 256-261
DOI: 10.5220/0002014102560261

made by humans and restoring correctness of the resulting process. We show the implementation of the architecture for instantiation of a reference process of a particular business unit of Siemens AG.

The paper is structured as follows: Section 2 discusses related work. Section 3 describes the developed instantiation approach and its architecture. A case study describing the implementation of the approach at Siemens AG follows in Section 4. In Section 5 we evaluate our approach and draw some conclusions in Section 6.

2 RELATED WORK

Instantiation of processes to project specific needs has been subject to intensive research in recent years. In early software process approaches it was thought that a perfect process can be developed which fits all software developing organizations and all types of projects (Boehm and Belz, 1990). It was soon recognized that no such process exists (Basili and Rombach, 1991), (Osterweil, 1987). Due to the dynamics of software development, every project has individual needs. Therefore, the description of a general process (i.e. the reference process) has to be adapted.

Early approaches to overcome this problem have been developed e.g. by (Boehm and Belz, 1990) and (Alexander and Davis, 1991). Many different approaches have been proposed since then. For example, the situational method engineering approach (ME) (Brinkkemper, 1996) emphasizes bottom-up assembly of project specific methods from fragments, requiring very detailed fragment specifications. In contrary to ME, approaches like SLANG (Bandinelli and Fuggetta, 1993) regard processes as programs, enacted by machines (Feiler and Humphrey, 1993).

Although much effort has been put into improving the adaption of software processes to project specific needs, none has become accepted in industry as the standard so far. An important reason is the variety of processes used. For instance, (Yoon et al., 2001) developed an approach for adapting processes in the form of Activity-Artifact-Graphs. Since the process is composed of activities and artifacts, only the operations ”addition” and ”deletion” of activities and artifacts are supported as well as ”split” and ”merge” of activities. Another example is the V-Model (BMI, 2004), a process model developed for the German public sector. It offers a toolbox of process modules and execution strategies. The approach for developing a project specific software process is to select required process modules and an execution strategy. Due to these dependencies on the Meta models, none of the existing approaches offers a complete and semi-automated method for instantiating Siemens processes.

Since the main purpose of Siemens software processes is to offer high level guidance for humans, adaptation approaches for business processes are also of interest. Approaches for processes and workflows of higher complexity are often restricted to only a subset of adaptation operations. For instance, configurable EPCs (C-EPCs) (Rosemann and van der Aalst, 2007) enable customization of reference processes. However, the approach only allows activities to be switched on/off, the replacement of gateways and the definition of dependencies of adaptation decisions.

(Armbrust et al., 2008) developed an approach for the management of process variants. A process is split up in stable parts and variant parts. The process is adapted by choosing one variant at the start of a project. Although the need for further adaptations during the execution of the process has been identified, no standardization or tool support is provided.

(Allerbach et al., 2008) developed an approach called Provop. Processes are adapted by using change operations which are grouped in Options. Options have to be predefined and can be used to adapt processes, but they do not guarantee correctness.

Existing tools (e.g. Rational’s Method Composer (RMC) (IBM, 2008)) provide only minimal support in instantiation. Decisions made by humans have to be executed mostly manually. Approaches have been developed making the user aware of inconsistencies (Kabbaj et al., 2008), but the actual correction of the process is still left to humans.

3 META MODEL BASED ARCHITECTURE

In order to improve current situation, we propose an architecture for instantiation systems that execute instantiation decisions made by humans and automatically restore correctness of the resulting process.

The actual changes in the process are executed by elemental operations on the process. Examples are ”Deleting an Activity” and ”Association of a Resource with an Activity”. Existing tools allow to perform those changes but they do not restore correctness of the resulting process.

For example: An activity a_1 is connect by an control flow to an activity a_2 which in turn is connected by an control flow to an activity a_3. If a project manager wants to delete a_2 in most existing tools he removes a_2 from the process . Additionally, he has to take care of restoring correctness e.g. establish the

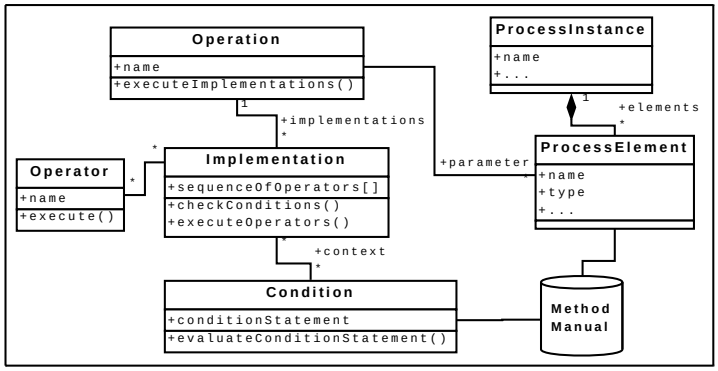

Figure 1: Architecture for Instantiation Systems.

broken control flow between a_1 and a_3, take care of affected information flows and resource connections. This task is further complicated by additional restrictions on process modeling, e.g. 'an *artifact* can only be created by one *activity*'.

In order to execute instantiation decision and restore correctness, the framework must consider the environment in which a change is performed. For every change of a specific process step instance *s* the context of that change is the set of instances and relationships connected to *s*. The abstract description of these entities and relationships for a particular type of process step is what we call the scope description of an operation. The scope description can be applied to the method manual of the process which generates all contexts an operations can be executed in.

Example: If milestones have restrictions on the type of entity they can succeed and precede, correctness depends on the preceding and successive element of the milestone. Thus, the scope description is: *Type of predecessor and type of successor.*

Entities and relationships influencing the way an operation has to be executed depend on the definition of the operation (i.e. changes on a process instance in order to instantiate it). Therefore the scope description of each operation has to be defined by an human expert. We propose the following procedure:

1. Identify restrictions on classes of entities of the method manual directly affected by the operation.
2. Identify classes of entities allowed in the process having a relationship with the classes of entities directly affected by the operation.
3. Identify restrictions on related classes of entities which might be breached by execution of the operation.

From the scope description, all individual contexts can be automatically generated. Every context has to be associated with an particular implementation which executes an operation in this explicit context. However, not every context requires an individual implementation that is only applicable in this particular context. From this follows that a number of contexts can be associated with one implementation.

The approach described above is a general framework for instantiation of processes and is not restricted to Siemens processes or a particular process definition language. The way the framework is implemented in detail depends on the requirements of the applying organization and on the method manual of the process which led to the developed of architecture described in Figure 1.

The class *ProcessInstance* corresponds to a project-specific software process. A process instance is created by copying the reference process for a particular project.

Instances of the class *ProcessEntity* are the elemental building blocks of a *ProcessInstance*. They can be of different types. Examples of the type "Activity" are "Develop Design Specification" or "Implement Design". Entities and thus process instances are adapted to project specific needs by running instances of *Operation* on them.

The class *Operation* correspond to instantiation operations. How an operation is executed on entities depends on the context in which the affected entities are nested in the process instance. Therefore instances of *Operation* can be associated with more than one instance of *Implementation*. When an *Operation* is executed, the *Implementation* which relates to the context at hand is executed.

For finding an *Implementation* that relates to a specific context *Operation.executeImplementations()* calls *Implementation.checkConditions()* for each of its implementations. If "true" is returned, the correct implementation has been found.

The actual process adaptations are carried out by a sequence of operators (*squenceOfOperators[]*) which

execute the actual changes in the process instance. Operators are elemental actions which create, update and remove entities of a process instance. The entirety of operators is stored in a repository. Their number is not fixed but can be further extended if necessary.

A context consists of one or several elemental *Condition*-instances. An examples of a condition is <self.source.isType() = 'Activity'> (type of source element is an activity). In order for a context to be true, all conditions associated with the context have to be evaluated *true*. They do not have to be defined by humans and their number is limited. Conditions are elemental statements about states and relationships of elements. The method manual defines the set of allowed statements for elements of a process instance. Conditions are therefore automatically derived from the method manual. In the next section, the proposed architecture is implemented for a reference process of a business unit at Siemens AG.

4 CASE STUDY

The reference process of the particular Siemens business unit covers the whole lifecycle of a software product. It is similar to a workflow but its focus is to be read and understood by human.

The process used in this article is simplified because of space limitations. Figure 2 gives an overview of the allowed classes of elements.

Entities are of the classes StartEvent, ManualActivity, AutomaticActivity, Milestone, Artifact, Human, System and splits and joins of the types And, Or and Xor. They are connected by ControlFlow, InformationFlow and ResourceConnection. A method manual has been defined on the basis of the Siemens Process Framework (SPF) and on specific regulations of the business unit. It describes restrictions on classes and their relationship by using OCL (OMG, 2006).

Operations required for instantiation of the reference process have been defined by experts of the business unit. In the following we focus on one example by describing how the scope description of the operation *Creating a Resource Connection* is defined.

The operation *Creating a Resource Connection* is defined as follows: *An existing Resource (res) is associated with an existing Activity (act) by a ResourceConnection (rc) of the type ExecutesRC, ResponsibleRC or ContributesRC. The user has to select the Resource (res), Activity (act) and the type of the ResourceConnection (rc).*

For defining the scope description of the operation the procedure described in section 3 was used:

1. **Identify restrictions on classes of entities of the method manual directly affected by the operation.**

Creating a ResourceConnection can affect restrictions on Flow, ResourceConnection, ExecuteRC, ResponsibleRC and ContributeRC since ResourceConnection has a superclass and subclasses. The method manual contains the following restrictions for these classes:

```
context ExecuteRC
  inv: (self.source.oclIsKindOf(Human)
     and self.target.oclIsKindOf(ManualActivity))
  or (self.source.oclIsKindOf(System)
     and self.target.OclIsKindOf(AutomaticActivity))
context ResponsibleRC
  inv: self.source.oclIsKindOf(Human)
  inv: self.target.oclIsKindOf(Activity)
context ContributesRC
  inv: self.source.oclIsKindOf(Human)
  inv: self.target.oclIsKindOf(ManualActivity)
```

In case of ExecuteRC, Human can only be associated with ManualActivity and System can only be associated with AutomaticActivity. In case of ResponsibleRC and ContributeRC, only Human can be associated with ManualActivity or AutomaticActivity.

From this follows that how a new ResourceConnection has to be created depends on the kind of ResourceConnection to be created, the kind of Activity and the kind of Resource. Thus a first part of the Scope Description is:

```
rc.isType()
res.isType()
act.isType()
```

2. **Identify classes of entities allowed in the process having a relationship with the classes of entities directly affected by the operation.**

By analyzing the statements extracted in step 1 it is identified that the classes Activity, AutomaticActivity, ManualActivity, Human and System have a relationship with the classes identified in 1. (Activity has no explicit restrictions defined in the method manual).

```
context AutomaticActivity
   inv: self.incomingCF->size()=1
   inv: self.outgoingCF->size()=1
   inv: self.executesRC->size()=1
   inv: self.responsibleRC->size()=1
   inv: self.contributesRC->empty()
context ManualActivity
   inv: self.incomingCF->size()=1
   inv: self.outgoingCF->size()=1
   inv: self.executesRC->notEmpty()
   inv: self.responsibleRC->notEmpty()
   inv: self.responsibleRC->size()<=1
context Human
   inv: self.incomingCF->empty()
```

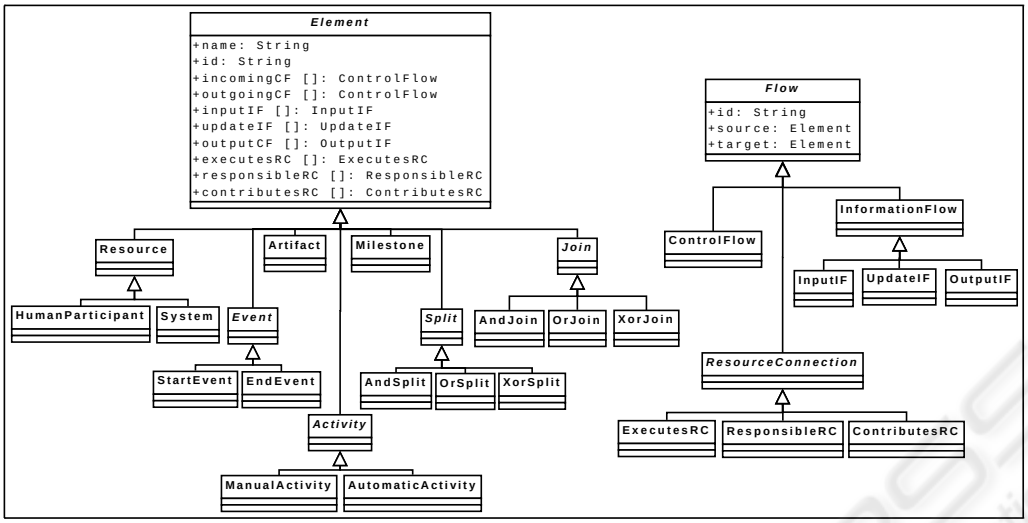

Figure 2: Classes of Process Entities and their Relationship.

```
   inv: self.outgoingCF->empty()
   inv: self.inputIF->empty()
   inv: self.updateIF->empty()
   inv: self.outputIF->empty()
context System
   inv: self.incomingCF->empty()
   inv: self.outgoingCF->empty()
   inv: self.inputIF->empty()
   inv: self.updateIF->empty()
   inv: self.outputIF->empty()
   inv: self.responsibleRC->empty()
   inv: self.contributesRC->empty()
```

The actually affected properties of the classes are determined through the types of attributes of class Element (all four classes are subtypes of Element).

3. Identify restrictions on related classes of entities which might be breached by execution of the operation.

By examining the statements from step 2, we can ignore all which are not affected by the creation of a ResourceConnection.

The number of ExecutesRC and ResponsibleRC an AutomaticActivity can be associated with is restricted to 1 and no ContributesRC are allowed. Since these values are static, they do not have influence on the way a new ResourceConnection is to be created.

We can ignore restrictions on ExecutesRC, since ManualActivity does not have a maximum number. A ManualActivity is only allowed to have either zero or one ResponsibleRC. Therefore the number of existing ResponsibleRC has to be taken in consideration. Since the operation is defined to deal with an instance of Activity (*act*) and not ManualActivity, we abstract to the former.

For Human Resources there are no restrictions regarding *Creating a ResourceConnection*.

System is not allowed to have ResponsibleRC and ContributeRC associated with it. Since these restrictions are static, they can be neglected.

In conclusion the Scope Description of the operation *Creating a Resource Connection* is:

```
rc.isType()
res.isType()
act.isType()
act.responsibleRC.size()
```

From the Scope Description all possible contexts can be computed. In the case of the operation *Creating a Resource Connection* each combination of all types of *rc*, *res*, *act* and the number of *responsibleRC* of *act* is regarded as individual context.

Because of space restrictions we cannot detail all contexts and their implementation for *Creating a Resource Connection*, but make use of a use case:

A resource connection ResponsibleRC (rc) has to be created between the ManualActivity (act) and the Human (res). The Activity (act) has another ResourceConnection (rc2) of the same class and one of the class ExecutesRC.

The specific context for execution ”Creating a ResourceConnection” in this situation is therefore:

```
rc.isType() = ResponsibleRC
res.isType() = Human
act.isType() = ManualActivity
act.responsibleRC.size() = 1
```

The implementation of the Basic Instantiation Operation for this context is:

```
delete (rc2)
create ResponsibleRC (act, res)
```

Since ManualActivity is only allowed to have one ResponsibleRC associated, first, rc2 is deleted and then a new ResponsibleRC (rc) created.

5 EVALUATION

A prototype of a system for instantiating the reference process of a Siemens business unit available in an extended version of XPDL 2.0 (WFMC, 2005) has been developed. The operations ”Inserting an Activity” and ”Deleting an Activity” have been chosen for testing. The architecture has been implemented accordingly.

For the operation ”Inserting an Activity” the system computed 43 contexts from the method manual and created for each context an instance of *Implementation*. Experts defined a sequence of operators for executing the insertion of an activity. Each implementation was executed with this sequence. The resulting process instances were checked by a debugger-class regarding their conformity to the XML-based method manual. The tests showed no violations of the resulting process instances.

For the operation ”Deleting an Activity” the same procedure was chosen. The system identified 49 contexts and created the corresponding instances of *Implementation*. After executing all implementations with a first sequence of operators, 44 resulted in a correct process. Violations of the remaining were written to a file and solved by iteratively developing the correct sequences of operators for them.

6 CONCLUSIONS

Siemens is currently undertaking research efforts to improve their software process related activities. Part of these efforts is the development of a system that supports project managers in instantiation of reference processes. The system aims not only to execute decisions but to restore correctness of the resulting process. Since the implementation of such a system is organization-specific and depends on the permitted constructs in the process, a flexible architecture has been developed and described in this paper. The approach was applied to a reference process of a business unit at Siemens AG and its feasibility was verified by the implementation of a prototype.

ACKNOWLEDGEMENTS

This work was partially supported by a DAAD postgraduate scholarship.

REFERENCES

Alexander, L. and Davis, A. (1991). Criteria for selecting software process models. In Proceedings of the Fifteenth Annual International Computer Software and Applications Conference, pages 521528.

Allerbach, A., Bauer, T., and Reichert, M. (2008). Managing process variants in the process life cycle. In Proceedings of the Tenth International Conference on Enterprise Information Systems, volume ISAS-2, pages 154161.

Armbrust, O., Katahira, M., Miyamoto, Y., Munch, J., Nakao, H., and Ocampo, A. (2008). Scoping software process models - initial concepts and experience from defining space standards. In ICSP, pages 160172.

Bandinelli, S. and Fuggetta, A. (1993). Computational reflection in software process modeling: The slang approach. In *ICSE*, pages 144–154.

Basili, V. and Rombach, H. (1991). Support for comprehensive reuse. *Software Engineering Journal*, 6(5):303–316.

BMI (2004). The new v-modell xt - development standard for IT systems of the federal republic of germany. URL: http://www.v-modell-xt.de (accessed 01.12.2008).

Boehm, B. and Belz, F. (1990). Experiences with the spiral model as a process model generator. In *Proceedings of the 5th Intern. Software Process Workshop 'Experience with Software Process Models'*, pages 43–45.

Brinkkemper, S. (1996). Method engineering: engineering of information systems development methods and tools. *Information & Software Tech.*, 38(4):275–280.

Feiler, P. H. and Humphrey, W. S. (1993). Software process development and enactment: Concepts and definitions. In *ICSP*, pages 28–40.

IBM (2008). Rational method composer. URL: http://www-01.ibm.com/software/awdtools/rmc/ (accessed 26.11.2008).

Kabbaj, M., Lbath, R., and Coulette, B. (2008). A deviation management system for handling software process enactment evolution. In *ICSP*, pages 186–197.

OMG (2006). Object constraint language v 2.0. URL: http://www.omg.org/spec/OCL/2.0/ (accessed 15.11.2008).

Osterweil, L. J. (1987). Software processes are software too. In *ICSE*, pages 2–13.

Rosemann, M. and van der Aalst, W. (2007). A configurable reference modelling language. *Information Systems*, 32(1):1–23.

Schmelzer, H. and Sesselmann, W. (2004). *Geschäftsprozessmanagement in der Praxis: Produktivität steigern - Wert erhöhen - Kunden zufrieden stellen*. Hanser Verlag, Muenchen, 4 edition.

WFMC (2005). WFMC-TC-1025-03-10-05 Specification for XPDL v2.0. URL: http://www.wfmc.org (accessed: 28.11.2008).

Yoon, I.-C., Min, S.-Y., and Bae, D.-H. (2001). Tailoring and verifying software process. In APSEC, pages 202209.