

# Specifying OCL Constraints on Process Instantiations

Peter Killisperger<sup>1</sup>, Markus Stumptner<sup>1</sup>, Georg Peters<sup>2</sup>, and Thomas Stückl<sup>3</sup>

<sup>1</sup> Advanced Computing Research Centre, University of South Australia

<sup>2</sup> Department of Computer Science and Mathematics, University of Applied Sciences München

<sup>3</sup> System and Software Processes, Siemens Corporate Technology, München, Germany

Email: mst@cs.unisa.edu.au

**Abstract.** Due to the variety of concerns affecting software development in large organisations, processes have to be adapted to project specific needs to be effectively applicable in individual projects. We describe a project aiming to provide tool support for this individualised instantiation of reference processes, based on an OCL-based specification of instantiation operations. The aim is not only to execute instantiation decisions made by humans but to automatically ensure correctness of the resulting process, potentially resulting in followup actions being executed or suggested.

## 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). 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 specific projects[4]. 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.

While structured adaptation approaches have been proposed, they are usually restricted to only a subset of adaptation operations, as in the case of configurable EPCs (C-EPCs) [5], which merely allow activities to be switched on/off, gateways to be replaced, and dependencies of adaptation decisions to be defined. The Protop approach [1] uses change operations which are grouped in Options. Options have to be predefined and can be used to adapt processes, but do not guarantee correctness. Due to their dependencies on very limited metamodels and an essentially informal representation, none of the existing approaches was found to offer a complete and at least semi-automatable method for instantiating Siemens processes.

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. the SPF).

In this paper we describe the framework developed for instantiation of processes, specify process properties by OCL constraints, and discuss the implementation options and how the constraints are used to maintain (restore) correctness of instantiations.

## 2 Constraints in the Software Engineering Framework

On the basis of information collected in interviews with practitioners at Siemens AG, a *Software Engineering Framework* (SEF) is being developed (Figure 1) with the aim of improving the instantiation and application of software processes [2].

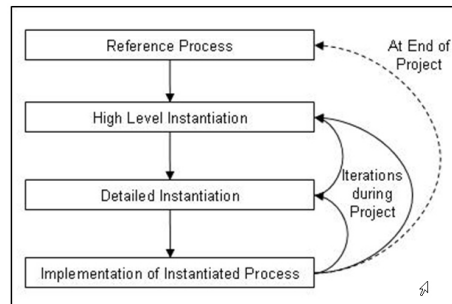


Fig. 1: Software Engineering Framework.

The SEF consists of a reference process, gradual instantiation by high level and detailed instantiation and an implementation of the instantiated process. High level instantiation is a first step towards a project specific software process by adapting the reference process on the basis of project characteristics and information that can already be defined at the start of a project and is unlikely to change. Such characteristics can be, e.g., the size of a project (a small project will only use a subset of the process) or high reliability of the software product (requiring certain activities to be added).

High level instantiation is followed by detailed instantiation which is run frequently during the project for the upcoming activities. A step by step approach is proposed, because it is often unrealistic to completely define a project specific process already at the start of a project.

The resulting instantiated process can be used in projects in different ways including visualization of the process and management of project artifacts.

Although instantiation in the SEF is split into two distinct stages, it is advantageous if both are based on the same principles. A set of elemental Basic Instantiation Operations (BIO) have been defined which are used for both stages. Examples are: "Deleting an activity" or "associating a resource with an activity". In high level instantiation, BIOs are executed on process elements as batch (i.e. predefined process adaptations depending on e.g. the project type) and in detailed instantiation individually. Using the same principles enables flexible definition and adaptation of predefined instantiation batches.

Existing tools (as mentioned in the introduction) allow performing those changes but they do not guarantee compliance with constraints. That is, current tools do not automatically correct a process when constraints on the process are violated due to instantiation decisions.

The reference processes of particular business units cover the whole lifecycle of a software product. They can be seen as being similar to workflows but their focus is on being read and understood by humans. (It is significantly harder to capture the details of design processes in a rigid workflow format than standard business processes.)

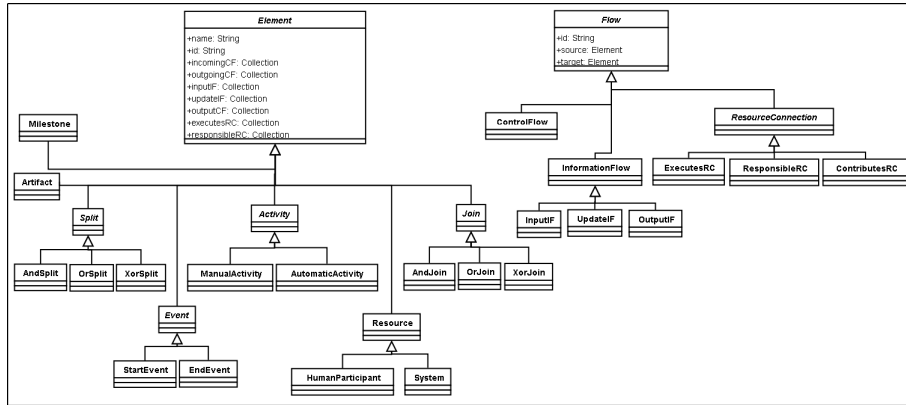


Fig. 2: Classes of Process Entities and their Relationship.

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

The constraints imposed on these processes are derived from process modeling languages and organizational modeling policies such as the SPF. Examples are: A manual activity has to have at least one human participant executing the activity or an artifact has to be created by an activity before it can be input of an activity. A method manual has been defined that can incorporate generic constraints on the reference process and specific regulations of the business units. It describes restrictions on classes and their relationship by using OCL [3]. The choice of OCL was made for several reasons: lowering of the hurdle for developers not fluent in formal notations who could still work with the intended OCL semantics; past experience by some of the authors in establishing formal semantics for OCL subsets; the integration with UML/MOF.

The operations required for instantiation of the reference process were defined by experts of the business unit. For example, 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). We give several example constraints taken from the the method manual that affect the relevant classes:*

```

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

```

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

Most of the constraints involved are of a very simple nature, typically referring to existence requirements for particular relationships. (They cannot be expressed as cardinalities in the process metamodel since they include conditions that are checked as preconditions for change operations.)

### 3 Architecture and Implementation

In order to reduce the effort of instantiation and to guarantee adherence of constraints we have developed an architecture for executing instantiation decisions made by humans which automatically restores correctness of the resulting process. The way a process is instantiated and what constraints are to be met by the process depends on the organization and on the process modeling language used. The architecture of an instantiation system must support differing method manuals since organizations impose different constraints their processes have to meet. Constraints may also change over time.

A process can be instantiated by running *Basic Instantiation Operations* (BIOs) which adapt the process in a predefined manner. If the resulting process violates constraints, the system is expected to provide a list of further changes that will restore consistency. Existing public domain OCL implementations turned out to be either too restricted in the types of constraints checked, or to produce the occasional runtime failure. Therefore it was decided to initially develop a purpose-built validation and repair module to show the viability of the approach.

How a violation is to be corrected depends on the environment in which the entities causing the violation are settled in the process. It might be possible to correct a violation not only in one particular way but in a variety of ways depending on properties and relationships of entities in the process. The way a violated constraint is corrected also affects what following violations occur and how they can be corrected. Consider an example where the project manager decides to adapt a purely sequential process by running the BIO inserting a parallel / alternative path which is defined as follows: *The user selects two control-flows from where the new path diverges / rejoins and specifies the type of split and join for diversion / rejoin. Both control-flows are deleted by the system and a split and a join of the chosen type is created.* By itself, this definition results in an insufficiently connected process, and the options of completing the missing control flow connections have to be examined. We implemented a systematic search mechanism that avoids potential live-lock situations, duplicated processes, and loops. The user is presented with a list of potential amended processes satisfying the constraints which he can accept or edit.

The system prototype, written in Java, was tested on process descriptions for a particular business unit. XPD L export functions were implemented as part of the tool for exchanging process definitions with the existing ARIS Toolset. The business unit uses a process comprising 23 types of entities including phases (composite activities consisting of a sub process), milestones and control flows. The method manual comprises furthermore several subtypes of activities, resources, artifacts, splits, joins, events, information-flows and associations of resources with activities (called resource connections). From the textual method manual, 135 constraints were identified on types of entities of the Meta model and classified in roughly ten different types of constraints for which repair methods were been defined. The reference process used for testing comprises about 3000 instances of entities. The 135 constraints defined on types of entities result in about 14000 constraints on instances of entities which have to be checked and, if necessary, corrected when violated during instantiation.

15 BIOs (defined by experts) necessary for instantiation of the reference process of the business unit were implemented. The operations are as elementary as possible in order to reduce complexity and avoid dependencies with other operations. Because of this simplicity, it might be necessary to execute more than one BIO to accomplish a complex adaptation step. The implementation was tested on representative editing sessions on the sample process and found viable for interactive use. To give an indication of performance, When running the BIO “inserting a parallel/alternative path” as described above (one of the least constrained operations with a resulting large potential search space), the unoptimised prototype provided the correct suggestion in about 2.5 seconds.

## 4 Conclusion

We have described a current research effort to improve software process related activities at Siemens. Part of these efforts is the development of a system that supports project managers in instantiation of reference processes subject to an explicit constraint representation. 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 and its feasibility was verified by the implementation of a prototype.

Further work is planned to include an implementation that replaces the dedicated repair-oriented algorithm with a generic constraint satisfaction implementation, as used in our service composition work [8], in which OCL is used as the specification language for pre-and post-conditions. This will both provide a more flexible computational architecture for reasoning about process instantiation and a formal semantics for the OCL constraints in our domain instead of the carefully crafted but essentially ad hoc semantics built into the repair operators of the current implementation. We also plan to adapt past work on behavior-based consistency criteria for refinement of object lifecycles to the process instantiation scenario [7,6], which will incorporate OCL consistency requirements.

## References

1. A. Allerbach, T. Bauer, and M. Reichert. Managing process variants in the process life cycle. In *Proceedings of the Tenth International Conference on Enterprise Information Systems*, volume ISAS-2, pages 154–161, 2008.
2. Peter Killisperger, Georg Peters, Markus Stumptner, and Thomas Stückl. Instantiating software processes. In *Proceedings 17th International Conference on Information Systems Development (ISD'08)*, 2008.
3. OMG. Object constraint language v 2.0. URL: <http://www.omg.org/spec/OCL/2.0/> (accessed 15.11.2008), 2006.
4. Leon J. Osterweil. Software processes are software too. In *ICSE*, pages 2–13, 1987.
5. M. Rosemann and W. van der Aalst. A configurable reference modelling language. *Information Systems*, 32(1):1–23, 2007.
6. Markus Stumptner and Michael Schrefl. Behavior Consistent Inheritance in UML. In *Proc. Int'l Conf. on Conceptual Modeling (ER)*, pages 527–542. Springer-Verlag, 2000.
7. Michael Schrefl and Markus Stumptner. Behavior-consistent Specialization of Object Life Cycles. *ACM TOSEM*, 11(1):92–148, 2002.
8. Rajesh Thiagarajan and Markus Stumptner. Service Composition With Consistency-based Matchmaking: A CSP-based Approach. In *Proc. IEEE European Conf. on Web Services (ECOWS)*, pages 23–32, 2007.