

# Automated Model Driven Development Processes

Aitor Aldazabal<sup>1</sup>, Terry Baily<sup>1</sup>, Felix Nanclares<sup>1</sup>,  
Andrey Sadovykh<sup>2</sup>, Christian Hein<sup>3</sup>, Martin Esser<sup>3</sup>, Tom Ritter<sup>3</sup>

<sup>1</sup>European Software Institute  
Parque Tecnológico de Bizkaia #204  
E-48170 Zamudio, Bizkaia-Spain  
{aitor.aldazabal|terry.bailey|felix.nanclares}@esi.es

<sup>2</sup>Softeam  
21 Avenue Victor Hugo  
75008 PARIS, France  
andrey.sadovykh@softeam.fr

<sup>3</sup>Fraunhofer Institute FOKUS  
Kaiserin-Augusta-Allee 31  
10589 Berlin, Germany  
{christian.hein|martin.esser|tom.ritter}@fokus.fraunhofer.de

**Abstract.** Development of large scale software systems requires well organized process structures for a fast, fault tolerant and cost-effective execution. A number of frameworks exist which assist developers in outlining and formalising processes, their workflows and work breakdown structures. The different activities which compose these processes need to be executed in an organized fashion. Each activity relies on certain development artefacts as inputs and in turn provides others as outputs. In this sense the individual activities involved in a development process can be thought of as services. Considering the typical scale of the development life-cycle for a complex piece of software, there are usually lots of services to be called in an orchestrated way to create a high-quality product in the end. This paper presents ModelBus, a novel integration framework, consisting of a service-oriented Middleware platform connecting arbitrary, model-based development tools, an orchestration tool allowing the definition and execution of service compositions and a management tool used to model and enact whole development processes. Its core objective is to introduce automation to software development processes on a large scale, ultimately resulting in an increase of productivity. It is available as Open Source Software and part of the MDDi Project [4] hosted by Eclipse.

**Keywords.** Process Enactment, Automation, Tool Integration, Modeling Services.

## 1 Introduction

In a model-driven development process naturally a lot of actions need to be performed on models. Among them are model transformations, model compositions, model verification, model checking, model storage and code generation. In the course of the development process contributors typically assume various roles with different interests in individual parts of the system at different stages of development. Each person engaged contributes at his own level of expertise and relies on different techniques for working with models. To facilitate the interaction of all parties involved and ultimately provide the basis for a smooth, fault tolerant development process, it is clearly desirable to create a homogeneous tool landscape which is able to satisfy most – if not all – of the developers' individual requirements.

So, obviously, there is a need to integrate different tools used by different parties at different steps of the process. Considering their diversity, this is by no means a simple task. Each of these tools offers distinct functionality and usually relies on special preconditions with respect to the expected input. For example, a model transformation tool expects input models of a specific type, realised using a specific modelling framework. Other tools, meant to be operated by human interaction offer a graphical user interface instead. This user interface visualizes functionality implemented in a single tool or even provided by a set of tools.

Altogether, based on [1] the following four types of integration can be identified:

- **Data Integration.** To integrate different tools which act on data (in this case on models) they need a common understanding of what kind of data they are supposed to handle and how this data is encoded. In the context of ModelBus this makes it necessary to exactly identify the types of models (e.g. a UML 2.1 model [6]) as well as the modelling framework (e.g. MOF[7]) and/or serialisation methods used.
- **Control Integration.** If tools offer certain functionality to be used by client tools, a description of the services exposed and the way they are meant to be invoked is absolutely vital for achieving smooth, reliable tool interaction. This typically requires a close examination of the corresponding access mechanisms and transport protocols (e.g. Web Services using SOAP over HTTP).
- **Process Integration.** Setting up a development process requires defining process phases and steps in which different activities are to be accomplished. The execution of the process relies on functionality provided by individual tools as well as interaction with human beings. Process integration is concerned with the orchestration of functionality, roles and data flows from a global perspective.
- **Presentation Integration.** Another type of integration arises from the need of an integrated user experience. The user of a set of tools naturally wants to have data presented to him in a way suitable for his specific working context as well as the role he is currently acting out within the development process. The challenge at this point lies in choosing the appropriate format and level of granularity for the presentation.

Faced with the task of integrating several arbitrary applications and considering Eclipse as one well-known 'integration platform', one may be inclined to think that a basic level of integration could be achieved by implementing all the needed tool

functionality in the form of a set of Eclipse plug-ins and then simply deploying those in a single Eclipse installation. On top of that a common perspective could be created, which in this case would serve the purpose of user presentation integration. As tempting as it may seem at first glance, this approach actually falls way short of creating a truly homogeneous and well integrated tool landscape, since it largely neglects the issue of control integration. Eclipse certainly offers support for local plug-in interaction, but in no way does it provide an out-of-the-box remote communication mechanism. All things considered, the Eclipse approach as outlined above rather resembles a common packaging than a truly integrated toolchain.

This paper focuses on presenting a solution for the first three types of integration needs; namely, data, control and process integration. Additionally, it discusses possible ways of achieving a satisfactory level of presentation integration.

The integration solution outlined below was created as part of two European projects – ModelWare [8] and ModelPlex [9] – both of them co-funded by the European Commission. The ModelWare project has already been successfully completed, whereas the ModelPlex project is currently still under development. Among other objectives, these projects put special emphasis on the evaluation of the product presented herein based on industrial case studies.

## 2 A world of Services

Simply combining a set of plug-ins into one Eclipse installation certainly yields a unified graphical interface to a tool chain but this can nonetheless hardly be called a properly integrated toolset with regard to the definition provided in section 1. Now to improve on this idea in the way of control integration something akin to the Eclipse Team Provider interface could be added to our setup. The ETP interface defines a uniform way of accessing similar services exposed by different providers. It is commonly used to plug several different CVS or SVN clients into a single Eclipse installation. However, following this course of action would only provide us with a common entry point to tools similar in functionality, when it is true standardized cooperation between possibly unrelated applications we are actually aiming at.

Another possible solution to our control integration problem lies in leveraging the functionality of the Eclipse Communication Framework (ECF) which offers a collection of extensible interfaces that applications can use to communicate with each other not only in a local but also in a remote fashion. However, this framework is targeted at Eclipse based tools only and does not sufficiently address the specific issues that arise when dealing with models.

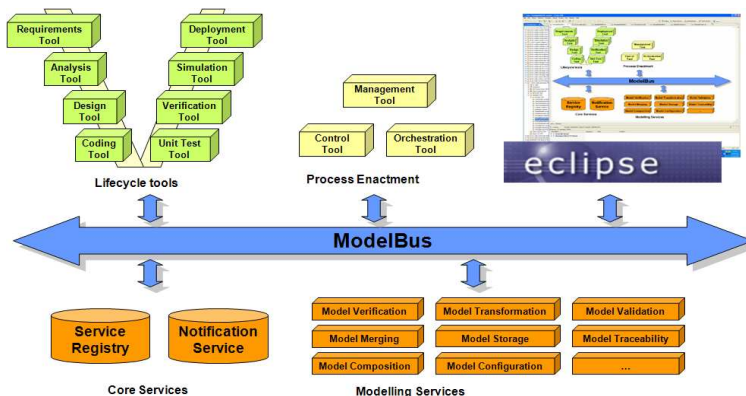
For this reason we finally decided to design an architecture of our own which allows full-fledged integration of arbitrary software tools (Eclipse based tools included), is strictly service oriented ('Service Oriented Architecture' – SOA) and relies on well established open standards. We call this platform ModelBus. One may think of ModelBus as a Middleware platform, not unlike CORBA [10], which uses a registry to record services exposed by tools connected to it and generally allows clients to invoke those services without having to worry about the distributed nature

4 **Aitor Aldazabal, Terry Baily, Felix Nanclares,  
Andrey Sadovkykh, Christian Hein, Martin Esser, Tom Ritter**

of the system. As an alternative, a SOA focused on software tool integration could certainly also be constructed using plain Web Services. Actually, ModelBus complies with this strategy to a certain point. The reason for not embracing the Web Service concept as a whole is that protocols like REST[11] or XML-RPC[12] do not care about the content of the messages they transport, which heaves the burden of validating the data solely on the receiver, and if this is neglected, results in an unstable platform prone to type violation errors.

In this context the issue of providing and requesting services which involve direct user interaction needs special attention. One of the solutions proposed, suggests implementing user interactions as services themselves: To request input from a user one service implementation simply invokes a second one that pops up some sort of message box to prompt the user for the necessary information and then routes it back to the caller. The latter processes it and eventually displays it again, possibly making use of yet another service to do so. This strategy may seem perfectly feasible for simple in- and output formats such as plain text, but is certainly not appropriate where data requires more complex graphical representations. For example, in order to properly render the output of a model simulator in a terminal window an application may very well need additional software currently nowhere available on the system. This scenario clearly overtaxes the capabilities of a pure SOA-based approach, and although we are well aware of this, we have so far not been able to find a completely satisfying answer to the issue, so that it will need to be addressed in future research activities by investigating appropriate presentation integration solutions. One possible approach may lie in leveraging the provision mechanisms offered by Eclipse (Myinstall, P2), which allow an easy distribution and remote installation of software bundles.

Ultimately, ModelBus in collaboration with the process enactment tools described later in this paper can be applied to create, orchestrate and monitor composite services provided by a variety of tools and integrate them into a single workflow. This can be presented to the user in the form of a high-level interface using an intuitive, business-oriented notation. Setting a development process into motion would then ideally be as simple as clicking a button, triggering all the necessary tools, services and user interaction events automatically.



**Fig. 1: ModelBus General Architecture**

The rest of this section attempts to provide a brief introduction to the key concepts of the ModelBus architecture illustrated in figure 1. For further details refer to [2] and [3].

A tool connected to ModelBus may consume as well as provide services. These services are generally called modeling services. A modeling service can offer arbitrary functionality, e.g. the transformation of an input model to an output model based on a specific set of rules. ModelBus uses the concept of an adapter to bridge differences in technology between its own infrastructure and applications willing to connect to it. To deploy a tool to ModelBus and publish its modeling services its adapter has to be successfully registered with the ModelBus Service Registry first. Afterwards, a tool in need of the service can simply consume it by issuing a corresponding method call via its own adapter. The adapter contacts the Service Registry, which then forwards the actual invocation to one of possibly several registered providers. Whether the invocation is handled locally or remotely remains completely transparent to the caller.

Besides the Core Services and Lifecycle Tools depicted in figure 1, ModelBus heavily relies on a number of Process Enactment tools further outlined in the following sections.

### 3 Orchestration

As already mentioned, Model Driven Engineering (MDE) and Model Driven Development (MDD) processes usually involve multiple heterogeneous model-based applications which are assembled into tool chains:

Working on large software projects almost inevitably imposes the use of code and model sharing solutions, such as CVS, SVN or other repositories, modelling suites providing different types of system views, such as goal and requirements models (BPDM), business models (BPMN [13]), conceptual, architectural and design models (UML2 with different types of profiles, DSLs) as well as a multitude of other tools used for model transformation, code generation, verification, simulation and testing. Developers usually touch several of them at each step of a project's life-cycle. Commonly, the modelling services those tools provide are executed manually. Developers copy models from one tool to another along with any execution parameters they require. This way of handling tool interaction is undoubtedly very inefficient, especially so, if the steps performed follow steadily recurrent patterns.

Provided such patterns can indeed be identified, the integration and orchestration capabilities of the ModelBus platform can be leveraged to replace the steps hitherto undertaken manually with an automated process and as a result noticeably facilitate development. For example, standard quality procedures may require a set of checks (naming convention, requirements description coverage, documentation coverage checks, etc.) to be run on a regular basis. Depending on the size of the models to be examined the whole procedure can be very time-consuming. Deploying the corresponding tools on ModelBus instead would allow automating the process,

6 Aitor Aldazabal, Terry Baily, Felix Nanclares,  
Andrey Sadovykh, Christian Hein, Martin Esser, Tom Ritter

thereby freeing lots of precious developer time, which could then for instance be used to improve quality management as a whole.

The Orchestration Tool developed in the context of the ModelWare and ModelPlex projects plays a key role in the automated execution of services provided by Model Driven Architecture (MDA) tools on ModelBus.

It relies on the Business Process Execution Language (BPEL) [14], a language originally designed for the purpose of orchestrating Web Services. BPEL offers a flexible notation for the description of business processes, including different types of service invocations, conditional and unconditional invocation transitions, loops, event handlers as well as language constructs for rollbacks and compensation activities.

Apart from its expressiveness and flexibility the use of BPEL comes with the additional benefit of allowing conventional MDA-tool based services and plain Web Services to be embedded into the same development process side by side. This is particularly useful in regard to human interactions, which under these circumstances can be included either as conventional Web Services like mail notifications or as modelling services compliant with the WS-Human Task and WS-BPEL4People standards.

To illustrate all this, let us take a more detailed look at the quality checking example mentioned earlier. The following Eclipse BPEL diagram outlines the process:

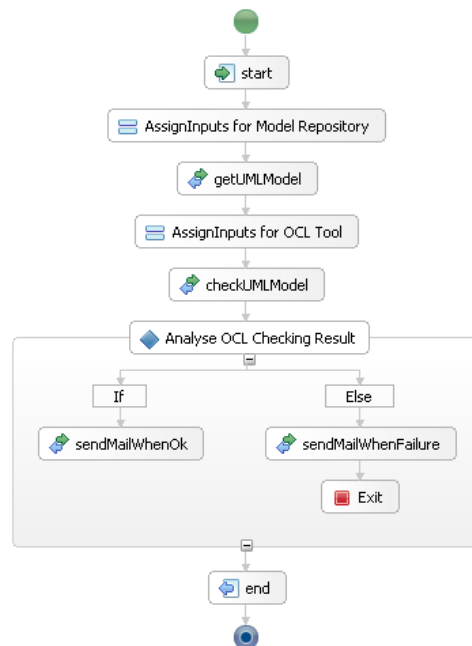


Fig. 2: Quality Constraints Checking BPEL Orchestration

The set of applications involved basically consists of a Model Repository and an OCL (Object Constraint Language [16]) Checking Engine. In the first step the OCL tool

retrieves a model from the repository. After that it validates the model according to the quality constraints it is assigned. Finally, a report on the outcome of the operation is mailed to all developers registered. Obviously, this process involves both classic ModelBus/MDA services – Model Repository and OCL verification tool – as well as conventional Web Services – the mail notification system.

At this point one may argue that integrating the checker functionality directly into a development environment and incrementally examining the constraints during composition of the model would be a lot more efficient. However, depending on the size of the model and the complexity of the given constraints the whole verification process may have to be applied to the completed model as a whole. In this case, said approach would result in blocking a whole workstation for a long time, which is clearly not desirable. Apart from that BPEL offers the additional benefit of reliably ensuring process execution persistency.

## 4 Process Enactment

### 4.1 Overview

Service orchestration is used to coordinate invocations of modelling services needed to complete specific activities. Once defined, it becomes the task of the project management to smoothly fit these activities into the software engineering process. Traditional project management and planning tools often do not offer much in the way of support for an effective arrangement of project activities and usually force project leaders to manually repeat a whole lot of tedious tasks, which by themselves do not add any value to the final product, but instead waste serious amounts of time, thus increasing overall costs. Sometimes these problems are palliated by using templates, provided the tools support those. However, in large organizations dealing with complex projects, each of the processes composing the life cycle of a project has to be examined at several levels of abstraction, rendering templates inefficient if not completely useless.

In this section we propose the use of MDE techniques usually found in Software Engineering domains to overcome the drawbacks of traditional process management tools. This approach is based on the assumption that traditional planning and monitoring tools more or less resemble plain programming languages in so far as they usually provide only a single low-level view of a project and as a result produce the same management problems a programmer faces when dealing with plain source code. MDE methods have already proven their usefulness in the programming area, and even granted that business oriented project planning and monitoring differ from the more technical domain of software development, the nature of the problems, which arise in both, is actually quite similar, and hence the solutions may be similar as well.

To be able to apply MDE methods to project management, appropriate means to model the processes and entities which make up the domain are needed. For this purpose several meta-process modelling standards, such as the Software Process

8 **Aitor Aldazabal, Terry Baily, Felix Nanclares,  
Andrey Sadovykh, Christian Hein, Martin Esser, Tom Ritter**

Engineering Meta-model (SPEM) [15] or the Business Process Modelling notation (BPMN) [13] – both of them initiated by the Object Management Group (OMG) – have already been conceived. These modelling notations can be seen as the equivalent of the Unified Modelling Language (UML) for the business process domain.

Being able to model all information concerned with the planning and layout of a project in a unified, structured fashion, makes it possible to subject this data to a wide range of transformations. Not only do transformations enable project managers to view business processes from different levels of abstraction, but also to automatically integrate data from various sources, thereby possibly inferring additional information, and to ensure high quality standards using existing model validation techniques.

The Software Process Engineering Meta-model (SPEM) is intended as a framework, which provides organizations with the concepts crucial to modelling, interchanging, documenting, managing and presenting their development methods and processes. It constitutes the very model underlying the Rational Unified Process (RUP) as well as many of the IBM Global Service Methods, even though to a lesser extent. Moreover it serves as the basis for the construction of a multitude of process authoring and management tools, including IBM's Rational Process Workbench, Osellus's IRIS, Softeam's Objecteering as well as various other developments in project planning and estimation, resource management, process simulation, etc.. The actual enactment of processes – that is, the planning and execution of a project – is, however, beyond its scope.

The Business Process Modeling Notation (BPMN) defines a standardized graphical notation used to represent workflows within a business process. BPMN was developed by the Business Process Management Initiative (BPMI), and is now being maintained by the OMG after the two organizations merged in 2005. The latest adopted version is 1.0, but a proposal for version 2.0 has already been submitted. The primary goal of BPMN is to provide a standard notation that is readily understandable by all business stakeholders including the business analysts who create and refine the processes, the technical developers responsible for implementing them and the business managers who monitor and direct them. In this way it serves as a common language to bridge the communication gap that frequently occurs between business process designers and implementers.

Both SPEM and BPMN are similar in their ability to describe workflows composed of multiple processes and tasks pertaining to arbitrary projects, products or activities. However, unlike SPEM, BPMN primarily focuses on the flow of activities, the relationships between individual artefacts and their associations with activities. All in all its vocabulary is comparable to that of a UML activity diagram. SPEM on the other hand does not allow such a fine workflow representation, but instead offers a very rich ontology, which enables process managers to describe aspects of a process which cannot be expressed in BPMN, such as roles, different states of artefacts, guidelines and work breakdown structures. Generally speaking, SPEM 2.0 is mainly used to formally specify software processes at a global level. The reasons prompting companies to do so are manifold. They range from quality model compliancy such as CMMI® (Capability Maturity Model Integration) to a desire for a better understanding of how an organization actually performs certain business tasks. Although specifically conceived for software development purposes, most of the SPEM specification nonetheless maintains a level of abstraction high enough to

potentially make it suitable for use with other vertical domains like banking and automotive engineering as well.

Considering their individual strengths and weaknesses, one may say that BPMN and SPEM naturally complement each other and are therefore best used together.

However, by themselves they are still clearly insufficient for the management of large enterprise projects and products, since neither do they provide specific, detailed execution plans required for actual process enactment nor do they offer any kind of mechanism to monitor and control business processes, not even mentioning the enforcement of specific standards.

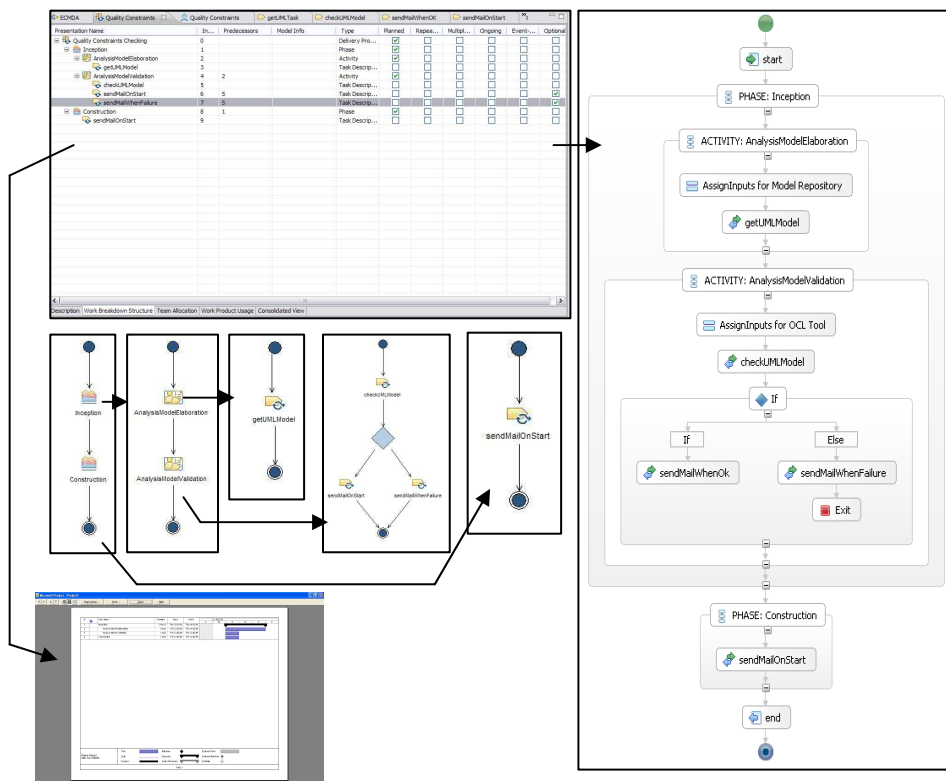
Fortunately, there already exist a couple of case-studies as well as mappings and tooling support for the transformation of BPMN models to BPEL, which can be executed inside of a SOA context, like the one provided by the ModelBus platform.

### 4.2 Fusing Enactment with Orchestration

In the course of the ModelPlex project the Eclipse Process Framework Composer was modified to serve as the core tool to model SPEM processes and to run transformations from SPEM to BPEL and MSProject. The transformations are based on Open Architecture Ware. Altogether, two types of transformations are performed: model-to-text transformations used to generate BPEL and MSProject files on the one hand and model-to-model transformations yielding enactment models on the other.

Figure 3 delineates how the key components of this mechanism interact. Unfortunately, none of the prototype's project planning and tracking abilities are displayed, since they are currently still under development.

Fig. 3: Process Enactment for Quality Constraints Checking



### 4.3 Example Use Case of MDE Techniques for Process Enactment

To illustrate the coupling of the software engineering process with orchestrations of MDD tool chains, let us consider the following simplified example of a RUP-like engineering process.

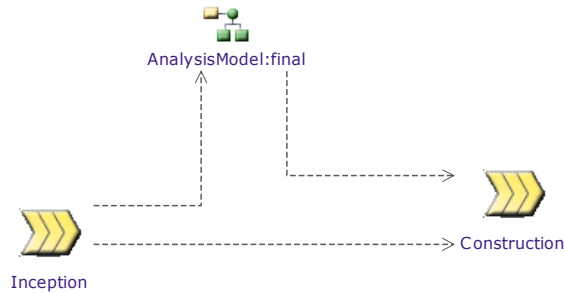


Fig. 2.: Simplified Software Engineering Process

The process consists of two phases – Inception and Construction. During the Inception phase an AnalysisModel is elaborated, validated and released. It is then transferred to the development team responsible for the Construction phase. Based on this engineering process, a project plan is created, which defines the number of iterations, dates, human resources allocated and the like.

From this an executable process plan can be generated by consecutively applying transformations from SPEM to BPMN and eventually to BPEL [5]. The resulting output defines the team interactions as well as the orchestration of all process patterns identified.

Figure 4 depicts a sample execution plan for our RUP-like process, building on the orchestration originally conceived for the quality constraints checking example presented in section 3.

It can be executed with common BPEL engines and monitored using well-known interfaces, making it possible to trace the execution of the project and synchronize the original project plan with its executable representation. All in all, this example demonstrates, how the ModelBus platform and its core tools can be applied to integrate a set of heterogeneous Model Driven Development tools into orchestrated tool chains and couple them with project execution environments using process enactment techniques.

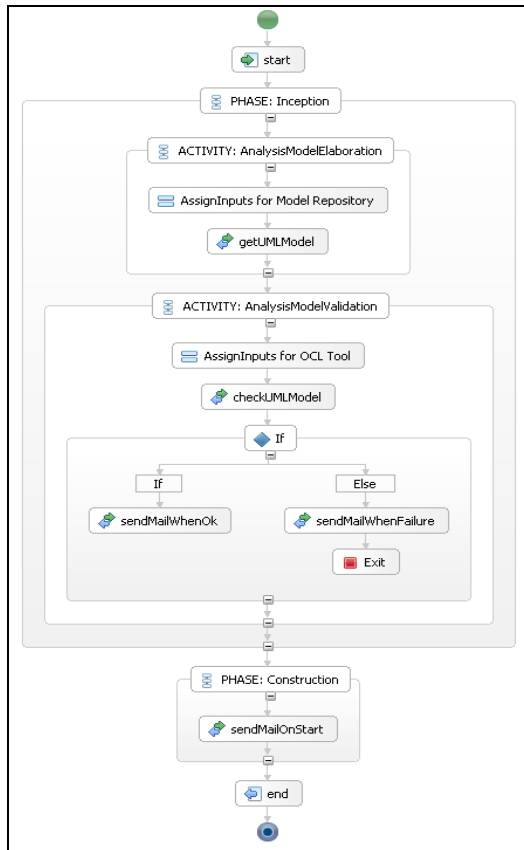


Fig. 3: Process Enactment for Quality Constraints Checking

## 5 Conclusion

In this paper we presented a general solution for the creation of integrated tool chains with the ultimate purpose of facilitating the management and automated execution of software development processes. Our approach is based on ModelBus, a service-oriented tool integration framework which allows applications to provide and consume arbitrary modelling services. Service invocations are automated with the help of an Orchestration Tool, which consists of a BPEL engine capable of orchestrating modelling services as well as plain Web Services. These service orchestrations constitute the fundamental building blocks of the enactment of a model-driven development process. The workflow within this process can be described using standard meta-process modelling languages such as SPEM and BPMN, which are subsequently transformed into executable BPEL plans.

12 **Aitor Aldazabal, Terry Baily, Felix Nanclares,  
Andrey Sadovykh, Christian Hein, Martin Esser, Tom Ritter**

Three different types of application integration, namely data, control and process integration could already be successfully realized. Pragmatic approaches to the issue of presentation integration have also been discussed, but need further investigation in upcoming research activities.

One of the major benefits of our solution certainly lies in the coupling of integrated tool chains with the MDD software engineering process, which in our eyes shows great potential in regard to increasing productivity. Apart from that, the platform offers extensive support for automation and contributes valuable post-execution capability, which can be used to evaluate completed tasks.

## References

- [1] I. Thomas, B.A. Najmeh. Definitions of tool integration for environments. IEEE Software, 9(2):29-35, March 1992
- [2] X. Blanc, M.-P. Gervais, P. Sriplakich, Model Bus. Towards the Interoperability of Modelling Tools, Proceeding of the European workshop on Model Driven Architecture: Foundations and Applications (MDAFA'2004), June 2004, Linköping University, Sweden, selected for : Lecture Notes in Computer Science (LNCS) « Model Driven Architecture: Revised Selected Papers », Volume 3599/2005, Springer
- [3] Prawee Sriplakich. ModelBus - An Open and Distributed Environment for Model Driven Engineering, Ph.D, September 2007
- [4] Model Driven Development integration – MDDi, [www.eclipse.org/mddi](http://www.eclipse.org/mddi)
- [5] Reda Bendraou, Andrey Sadovykh, Mari-Pierre Gervais and Xavier Blanc. Software Process Modeling and Execution: The UML4SPM to WS-BPEL Approach. Proceedings of IEEE EUROMICRO 2007, pp. 314-322, ISBN 0-7695-2977-1
- [6] OMG Document - UML – Unified Modelling Language (UML) Specification: Infrastructure. <http://www.omg.org/docs/formal/07-02-03.pdf>
- [7] OMG Document – MOF – Meta Object Facility formal/02-04-03.pdf
- [8] ModelWare Project <http://www.modelware-ist.org/>
- [9] ModelPlex Project. <http://www.modelplex.org>
- [10] OMG Document – CORBA – Common Object Request Broker Architecture [http://www.omg.org/technology/documents/corba\\_spec\\_catalog.htm](http://www.omg.org/technology/documents/corba_spec_catalog.htm)
- [11] Representational State Transfer <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- [12] Extensible Markup Language Remote Procedure Call -XML-RPC Specification <http://www.xmlrpc.com/spec>
- [13] OMG Document - Business Process Modeling Notation <http://www.omg.org/cgi-bin/apps/doc?dtc/06-02-01.pdf>
- [14] OASIS Document - Business Process Execution Language <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>
- [15] OMG Document - Software Process Engineering Metamodel <http://www.omg.org/technology/documents/formal/spem.htm>
- [16] OMG Document – OCL - Object Constraint Language Specification Version 2.0. <http://www.omg.org/docs/ptc/05-06-06.pdf>