Towards an MDA-based development methodology for distributed applications

Anastasius Gavras[1], Mariano Belaunde[2], Luís Ferreira Pires[3], João Paulo A. Almeida[3]

[1]Eurescom GmbH, [2]France Télécom R&D, [3]University of Twente

[1]gavras@eurescom.de, [2]Mariano.belaunde@rd.francetelecom.com, [3]{pires,alme}@ewi.utwente.nl

Abstract

This paper proposes a development methodology for distributed applications based on the principles and concepts of the Model-Driven Architecture (MDA). The paper identifies phases and activities of an MDA-based development trajectory, and defines the roles and products of each activity in accordance with the Software Process Engineering Metamodel (SPEM). The development methodology presented in this paper is being developed and applied in the European 5th Framework project MODA-TEL, which aims at assessing the applicability and potential of MDA in the context of telecom services and applications. This paper also discusses the application of the proposed methodology on a typical telecom service case study. The paper claims that the proposed methodology is general enough to be applicable to distributed applications in other domains as well.

1. Introduction

The Model-Driven Architecture (MDA) [6], which is being currently promoted by the Object Management Group (OMG), consists of a set of concepts and principles for the development of distributed applications. The MDA standards define technologies to support these concepts and principles, but they do not prescribe nor require any specific *development methodology*, by which we mean that MDA gives no guidelines in terms of the processes (activities and phases), roles and responsibilities that are involved in the development trajectory of a distributed application. Furthermore, the MDA technologies are not explicitly related to identifiable activities within software development processes, since these technologies are being developed to be generally applicable in combination with development processes that may already be anchored in organisations.

Since MDA does not prescribe a development methodology, each MDA-based development project has to define its own methodology or apply existing ones. This paper outlines the MDA-based development methodology that is being developed and applied in the MODA-TEL project [2]. MODA-TEL is an European IST 5th Framework project that aims at assessing the applicability and potential of MDA in the context of telecom services and applications. This paper identifies phases and activities in the development process, and defines the roles and products of each activity in accordance with the Software Process Engineering Metamodel (SPEM) [3]. The methodology presented in this paper can be seen as a framework for combining established software development processes with the MDA concepts, principles and technologies, and thus customising the specific software engineering process that may be used in an organisation. This allows organisations to profit from the benefits of applying MDA, like model reusability, preservation of application development investments and automated transformations, to name just a few.

The paper is further structured as follows: Section 2 gives an overview of our methodology, in terms of its main activities and phases, Section 3 discusses the activities of the project management phase, Section 4 discusses the project preparation activities, Section 5 presents the activities of the project execution phase, Section 6 illustrates some activities of our methodology with a case study on the development of a VoiceXML application and Section 7 draws some conclusions.

2. Development activities and phases

We start the identification of the development phases in an MDA-based project by classifying the users of MDA technology in three categories:

- *Knowledge builders*: people who build knowledge (repositories) to be used in multiple different MDA-based projects. This category includes systems architects, platform experts, quality engineers and methodology experts. We estimate that this group amounts approximately 5% of the total MDA users population;
- *Knowledge facilitators*: people who assemble, combine, customise and deploy knowledge for each specific MDA-based project. This category includes project managers and quality engineers. We estimate that this group amounts approximately 5% of the total MDA users population;
- *Knowledge users*: people who apply the knowledge built and facilitated by the other user categories, respectively. This category includes designers and software engineers. We estimate that this group amounts approximately 90% of the total MDA users population.

Figure 1 illustrates the three categories of MDA technology users.

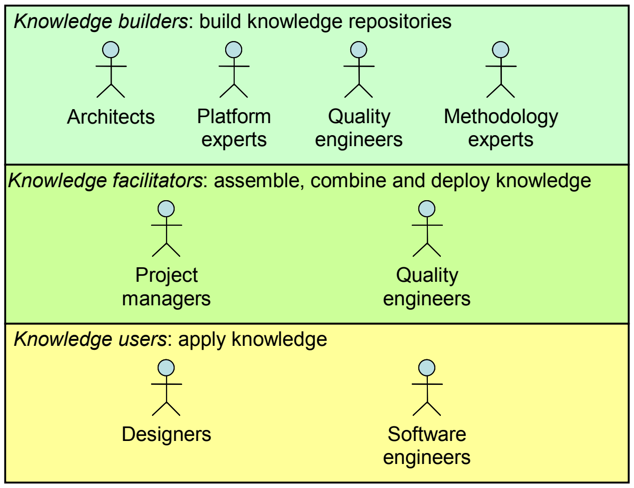
Knowledge builders: build knowledge repositories
Architects
Platform experts
Quality engineers
Methodology experts
Knowledge facilitators: assemble, combine and deploy knowledge
Project managers
Quality engineers
Knowledge users: apply knowledge
Designers
Software engineers

Figure 1. Categories of MDA users

Figure 1 shows that different roles and skills can be identified in the MDA users population. These roles perform different activities and require different tools.

In any MDA-based project, the distinction between preparation activities and execution activities is essential. Preparation activities are those that structure and plan the work, and as such they enable knowledge reuse, which is one the main benefits of the MDA. Preparation activities are mainly performed by knowledge builders and should start before the project execution activities. However, it should be possible to switch between preparation and execution activities, allowing the preparation activities to be revisited while the execution activities are being carried out. This is necessary because project requirements may change (e.g., change of platform), more detailed requirements may be defined (e.g., some requirements were not detailed enough) and problems may occur in the execution phase (e.g., selected modelling language is found too limited or not expressive enough), amongst others.

The MODA-TEL methodology identifies the following phases:

1. *Project management*: aims at organising and monitoring the project;
2. *Preliminary preparation*: aims at identifying modelling and transformation needs;
3. *Detailed preparation*: aims at obtaining the modelling and transformation specifications;
4. *Infrastructure setup*: aims at making tool support and metadata management facilities ready to use;
5. *Project execution*: aims at producing the necessary software artefacts and the final products.

Figure 2 shows the five phases of the MODA-TEL methodology and their relationships. For reasons of conciseness, in Figure 2 we have omitted the relationships between the project management phase and the other phases.

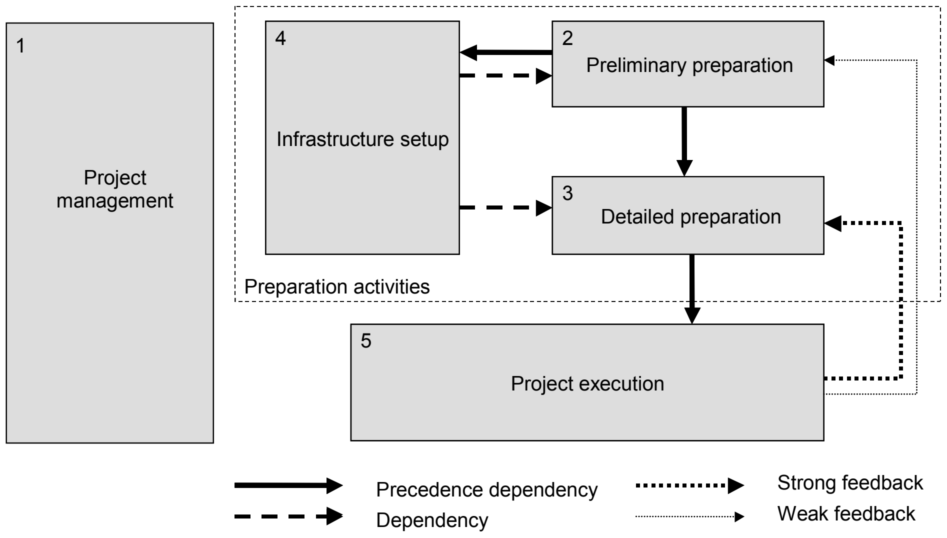

Figure 2. Development phases

The phases of our methodology correspond to the available and required expertise identified before, and, therefore, these phases can be directly associated with the partitioning of the MDA users expertise shown in Figure 1: phase 1 is mainly performed by knowledge facilitators, phases 2, 3 and 4 are mainly performed by knowledge builders, while phase 5 is mainly performed by knowledge users.

Figure 2 shows how the preparation activities have been structured in different phases. These phases are useful to understand and to describe the dependencies between the activities. Project management activities have a direct impact on all the other activities; in particular, the activity that defines the whole software development process prescribes the list of the execution activities to be performed, such as, e.g., the sequence of transformations to be implemented. Activities of the preliminary and detailed preparation phases, such as selecting a platform and deciding on the usage of a modelling language, are the key elements to enable reuse of knowledge in the project execution phase. Finally, the activities of the infrastructure set-up phase, such as, e.g., tool selection, influence the preliminary and detailed preparation phases, even if project managers have decided to be as much tool-independent as possible.

Figure 2 also shows that many dependencies have been identified between the development phases of our methodology, which means that these phases should be performed iteratively and incrementally. Feedback from the execution activities to the preparation activities, and vice-versa, should be taken into account in an effective way. The availability of model-to-model transformations, code generation techniques and well-defined traceability strategies are crucial for this purpose.

3. Project management phase

We distinguish between typical process management activities, such as keeping track of milestones and resource consumption, and activities that are directly related to management decisions absolutely necessary to setup the project, such as the selection of the engineering process. Additional activities known and applied from "best practices" in project management can still be added to this phase, but are not explicitly covered by our methodology.

The management activities identified here may be strongly influenced by preparation activities, e.g., in case SPEM [3] is used to explicitly describe the engineering process, and by execution activities, such as requirements analysis.

In the project management phase we have identified three activities:

- *Software Development Process (SDP) selection*, which results in the description of the software development process to be followed at the execution phase, in terms of specific sub-activities and the resulting work products. A discussion on the use of MDA in combination with some established software development processes can be found in [4];
- *Project organisation* (identification of roles), which results in the allocation of activities to process roles;

- *Quality management*, which defines procedures to enhance the quality of the development projects. Some aspects of quality management can be orthogonal to the SDP, such as, for example, the maturity levels of the Capability Maturity Model (CMM) [7].

Figure 3 depicts the activities of the process management phase and the relationships between these activities.

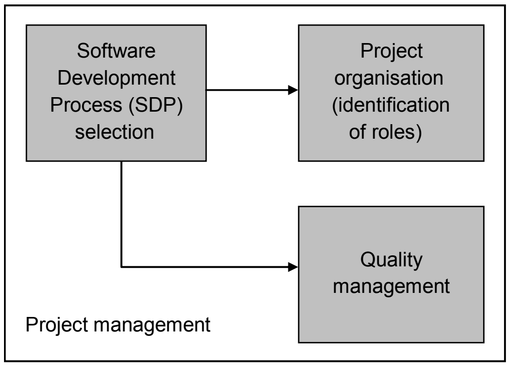
Software Development Process (SDP) selection
Project organisation (identification of roles)
Quality management
Project management

Figure 3. Project management activities

Since MDA is based on the principles of object-orientation and component-based development it fits well into most contemporary software development processes. MDA has been conceived to allow the existing development processes in organisations and projects to be reused to a large extent, since MDA concepts can be applied in the scope of these processes.

We use the term *Model Driven Engineering* (MDE) to denote the process of applying an MDA-based SPD. The engineering aspects, i.e., the designing, building and maintaining pieces of software, are dynamic and contrast with the static nature of a set of models. There is no single way to engineer software and many different alternatives can be found by reusing elements of some established software development processes.

Figure 4 shows the relationship between the SDP selection activity of the process management phase and the project execution phase.

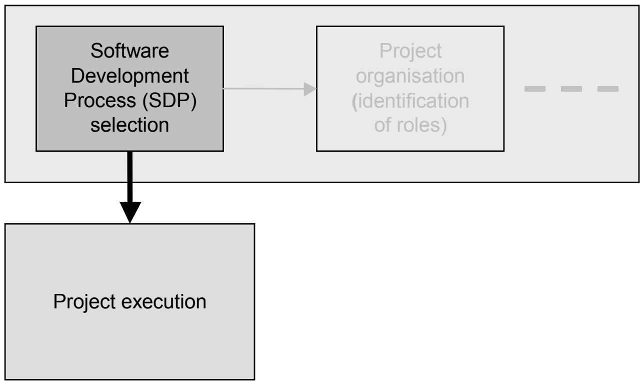
Software Development Process (SDP) selection
Project organisation (identification of roles)
Project execution

Figure 4. Influence of the SDP on the project execution phase

4. Preparation activities

The preparation activities have been grouped in three phases, namely preliminary preparation, detailed preparation and infrastructure setup. Each of these phases and their relationships with other phases are discussed below.

4.1. Preliminary preparation phase

In the preliminary preparation phase we identify four activities:

- *Platform identification*: a platform refers to technological and engineering details that are irrelevant to the fundamental functionality of a system (or system part). What is irrelevant and what is fundamental with respect to a design depends on particular design goals in different stages of a design trajectory. Therefore, in order to refer to platform-independent or platform-specific models, one must define what a platform is, i.e., which technological and engineering details are irrelevant, in a particular context with respect to particular design goals. In this activity we identify the concrete target platform(s) on which the application is supposed to be implemented and their common abstraction in terms of an abstract platform [1]. Concrete platforms may also include legacy platforms;

- *Modelling language identification*: models must be specified in a modelling language that is expressive enough for its application domain. This activity identifies the specific needs for modelling languages. Since models can be used for various different purposes, such as data representation, business process specification, user requirements capturing, etc., many different modelling languages may be necessary in a development project. Process roles for performing this activity include domain experts;
- *Transformations identification*: transformations define how model elements of a source model are transformed into model elements of a target model. This activity identifies the possible or necessary transformation trajectories from the abstract to the concrete platforms. These transformations have to take into account the modelling languages identified before;
- *Traceability strategy definition*: traceability in model transformation refers to the ability to establish a relationship between (sets of) model elements that represent the same concept in different models. Traces are mainly used for tracking requirements and changes across models. This activity defines the strategy to be applied in the definition of traces along the development trajectory.

Figure 5 shows the activities of the preliminary preparation phase.

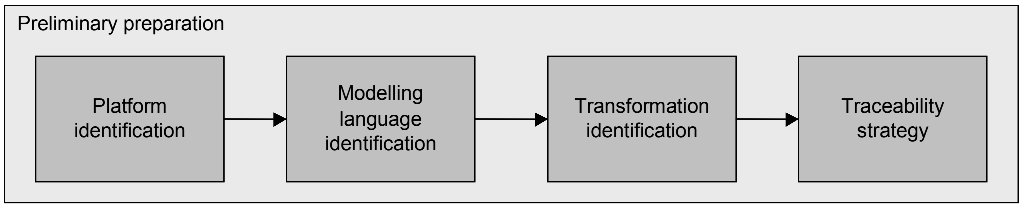

Figure 5. Preliminary preparation activities

The activities of the preliminary preparation phase often depend on the requirement analysis activity of the project execution phase (see Section 5), as depicted in Figure 6.

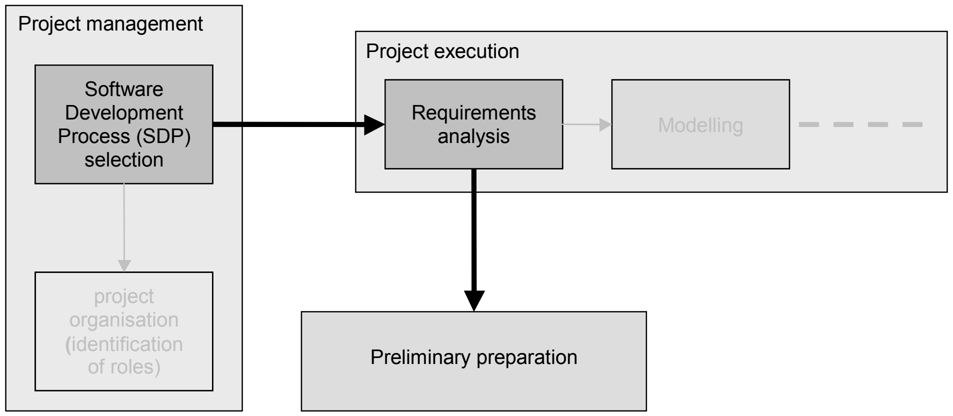
Project management
Software Development Process (SDP) selection
project organisation (identification of roles)
Project execution
Requirements analysis
Modelling
Preliminary preparation

Figure 6. Influence of requirements analysis on the preliminary preparation phase

In case model-driven techniques are used for requirement analysis, certain preliminary preparation activities may precede requirement analysis. For example, this can be the case if a UML profile or a metamodel is available for the User Requirement Notation (URN) [8]. Identifying such a profile or metamodel is a preliminary preparation activity to be performed before requirements analysis.

4.2. Detailed preparation phase

In the detailed preparation phase we have identified two activities:

- *Specification of modelling languages*: in accordance with the specific needs for modelling languages identified before, this activity identifies the concrete general purpose or domain specific modelling languages that shall be used in the execution phase. Source and target metamodels used in the transformations are also defined in this activity. Process roles for performing this activity include domain experts;
- *Specification of transformations*: model transformations need rules and annotations to control the transformation process. Rules control the transformation of an annotated source model to a target model. Rules have to be defined at the metamodel level, in order to be applicable to any instance of the source metamodel that is transformed to an instance of the target metamodel. Rules can be formalized in a certain modelling language or metamodel, or they may be defined as code in a scripting or programming language. Annotations are information related to a model, optionally defined in terms of elements of this model's metamodel. This activity is concerned with the specification of the necessary transformation rules and annotations.

Figure 7 shows the activities of the detailed preparation phase.

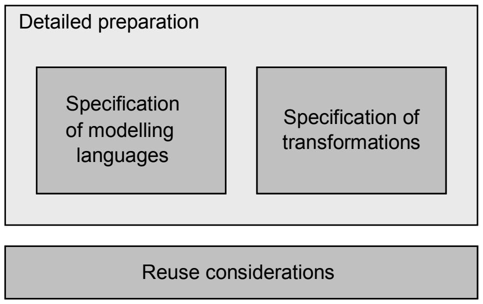
Detailed preparation
Specification of modelling languages
Specification of transformations
Reuse considerations

Figure 7. Detailed preparation activities

Language and transformation specifications produced in this phase are strong candidates for reuse, namely in future projects in similar application domains. Therefore these specifications should be somehow stored and catalogued for future use. These reuse considerations are also depicted in Figure 7.

4.3. Infrastructure setup phase

In the infrastructure setup phase we have identified two activities:

- *Tool selection*: a number of activities in our methodology have to be handled by tools, such as (i) the definition of models and metamodels, (ii) the transformation and code generation based on model information, (iii) the definition of constraints and rules to verify model compliance. This activity aims at selecting of one or more tools to support activities in the development process. For the selection of appropriate tools, all requirements from the software engineering perspective are identified and mapped to capabilities of existing tools available on the market;
- *Metadata management*: metadata provides in most cases information about the structure of data, e.g., which data types are available, the structure of these data types, what data aggregations are valid, etc. Different technology families usually define their own ways to manage metadata, as well as to generate and manipulate metadata repositories. Metadata can be used in different situations, like, e.g., to store information about transformations, to store information about available resources, to support migration or to support applications during runtime. In each project, the necessary support for metadata as well as the way to manage metadata is defined in this activity.

Figure 8 shows the activities of the infrastructure setup phase.

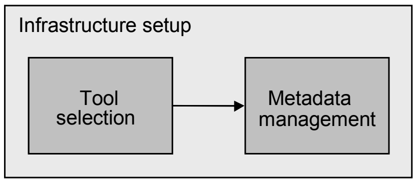

Figure 8. Infrastructure setup activities

The tool selection activity can be quite intricate. The choice of the most appropriate MDA tool depends mainly on the level of engineering support required in the project. In some projects, MDA tools may be required to support behaviour modelling and simulation. In general MDA tools should also give support to traceability, for example, to associate code fragments to their corresponding model elements in order to guarantee that changes in the code are reflected in the model and vice-versa. Extensibility, integration with XML-based techniques and interoperability with other tools may also be important requirements to consider. Furthermore, other circumstances like the availability of a certain tool in an organisation or the experience of the designers with some specific tool may strongly influence if not determine the choice. The tool selection activity may have an impact on each of the preparation activities, as well as on the metadata management activity.

5. Execution phase

The project execution phase is the main phase of a project, since in this phase the developers apply the acquired knowledge to produce software artefacts and deliver the final products. The specific activities of this phase depend on the selected SDP, which is described in terms of sub-activities and work products. However, for the purpose of our methodology we have identified general activities that appear in virtually any object-oriented or component-based SDP. Our methodology has identified seven activities in the project execution phase:

- *Requirements analysis*: this activity generally aims at (i) establishing a dictionary with well-defined terminology and (ii) structuring the requirements. Both the dictionary and the requirements are normally used as input to produce conceptual domain models. Requirements should also be associated to their corresponding model elements, allowing traceability from requirements to models or even to code. It may be even possible to have some model-to-model transformation that creates an initial platform-independent model (PIM) from requirements models;

- *Modelling*: this activity comprises the formal specification, construction, documentation and (possibly) visualisation of artefacts of distributed systems, using one or more modelling languages. This activity is concerned with the development of software engineering specifications that are expressed as an object or component model or combinations thereof. The products of this activity are specifications of the structure of these artefacts, such as names, attributes and relationships with other artefacts. Behaviour specifications describe the behaviour of the artefacts in terms of states, allowed transitions and the events that can cause state changes. The interactions between artefacts may also be represented in behaviour specifications. These models are created with the help of tools that support the representation of the artefacts and their behaviour;

- *Verification/Validation*: this activity is concerned with (i) determining whether or not the products of the modelling activity fulfil the requirements established by the requirements analysis activity, and (ii) evaluating whether the products of the modelling activity are free from failures and comply with the requirements established in the requirements analysis activity. Some existing technologies allow these activities to be performed (semi-) automatically by using tool support. A verification/validation strategy for the produced models has to be explicitly defined in this activity;

- *Transformations*: this activity is concerned with the refinement of the models produced in the modelling activity by means of rules and annotations that control the transformation process. The artefacts defined by the modelling activity are refined by defining data structures and procedures, defining message protocols for the interactions, mapping the artefacts into classes and mapping these onto constructs of a programming language (model-to-code transformations);

- *Coding/Testing*: this activity is concerned with the development of code that is necessary to complement the automated code generation. With current technology, somecoding is still required by developers after a model-to-code transformation has been performed. The same applies for the execution of test cases. Automatic testing is possible to some extent, but usually manual testing is also necessary to complement the testing activities;

- *Integration/Deployment*: this activity is concerned with the embedding of the newly developed systems into their operational environment. In large organisations, new services and applications have to co-exist with established systems and work on existing infrastructures. The MDA prescribes that (new) functionality should be modelled at the platform-independent level. Since platform-independent models of the existing (legacy) systems can be developed by applying reverse engineering, integration issues can be addressed already at the platform-independent level. The deployment sub-activity is concerned with the management of the life-cycle of component instances running on the nodes of a platform. This sub-activity handles issues like, e.g., the transfer of implementations to the appropriate nodes, and instantiation, configuration, activation and deactivation of component instances;

- *Operation/Maintenance*: this activity is concerned with the overall management of the life-cycle of a distributed application, including issues like, e.g., dynamic configuration, dynamic service upgrade, and service migration to different nodes;

Figure 9 shows the activities of project execution phase.

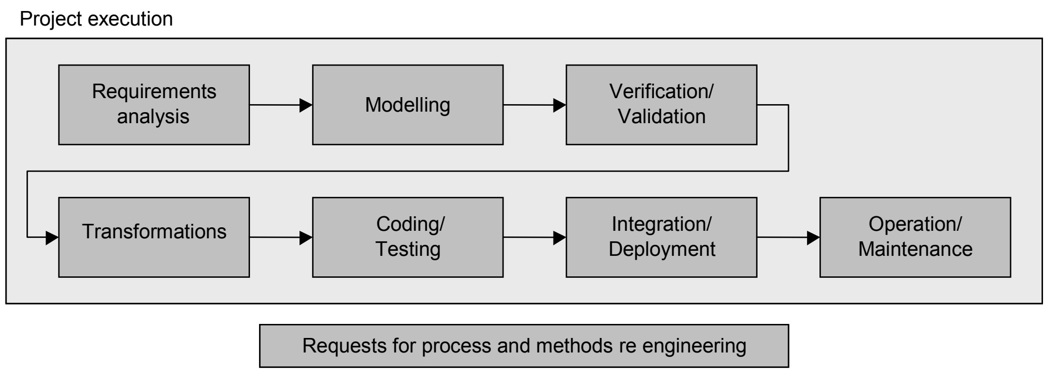

Figure 9. Project execution activities

In general, the activities in the project execution phase can be repeated more than once, e.g., if multiple development iteration cycles are applied or errors are found. In case failures, defects or other problems are discovered in one of the activities, the process should resolve the issue at the modelling activity, since models are supposed to drive the whole process execution phase. All activities of the project execution phase can generate feedback to refine and improve of the processes and methods, influencing in this way the preliminary or the detailed preparation phases or both, depending on the severity of the feedback.

6. Case study: a VoiceXML application

VoiceXML [9] is a technology that provides telecom operators with features to enhance their services with interactive voice responses. These services are voice-based and include Text To Speech (TTS), and Automatic Speech Recognition (ASR) subsystems. VoiceXML is a mark-up language (i.e., an XML schema), for specifying voice dialogues and has been introduced in order to free the authors of voice response applications from low-level programming and resource management.

In the MODA-TEL project we have a VoiceXML case study that aims at defining voice-based telecom services in a PIM model, and providing the transformation to a PSM model based on the existing infrastructures. France Telecom (FT) and Telenor will use different concrete platforms in order to stress the MDA character of the case study. Below we briefly discuss some activities of our methodology as it has been applied in the case study, focusing on the FT implementation.

6.1. Project management

MS Project has been selected as the tool for project organisation (work partitioning and identification of roles). We have decided to use a process definition tool to specify the engineering process to be executed and we are using SPEM [3] for this purpose. The requirements for the process definition tools are (i) support for the SPEM notation and (ii) the ability to export the model representing the engineering process. The SPEM UML profile implemented within the Objecteering tool [10] has been selected for this purpose.

Figure 10 illustrates the development process intended for developing the voice services on top of Euphonie, which a France Telecom proprietary VoiceXML platform that offers a framework for developing, executing and debugging interactive voice services using next generation technologies and VoiceXML. This process has been specified using Objecteering and the SPEM UML profile.

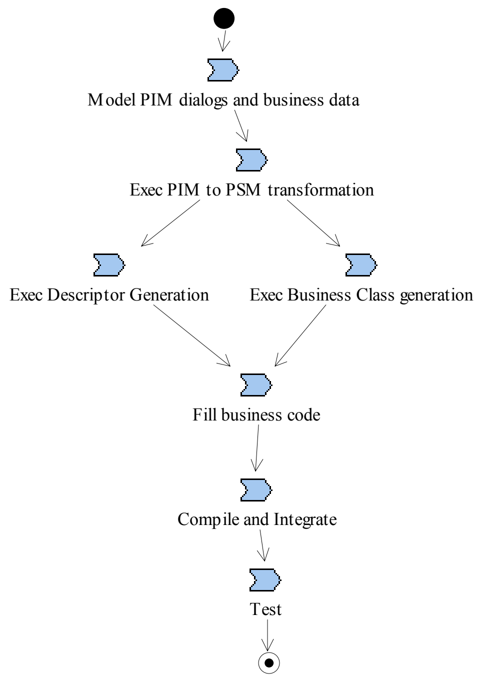
Model PIM dialogs and business data
Exec PIM to PSM transformation
Exec Descriptor Generation
Exec Business Class generation
Fill business code
Compile and Integrate
Test

Figure 10. Development process for voice services

6.2. Preliminary preparation phases

A specific voice-oriented UML profile should be used by end-users to model dialogs at PIM level. An intermediate PSM metamodel is used for representing the projection of a specific voice service into the Euphonie platform.

Two model-to-model transformations are being defined in the FT implementation: (i) from the UML profile to the PIM metamodel and (ii) from the PIM metamodel to the Euphonie PSM metamodel. Two code generators are also being defined: (i) for generating the state machine descriptors and (ii) for producing the Java classes compliant with the Euphonie platform.

We expect that all the code associated with dialog description will be produced directly from the PIM models and annotations. For business code only the skeletons of the classes will be generated. Up to now, no decision has been taken on how to handle traceability.

6.3. Detailed preparation

The Transformation Rule Language (TRL) [11] formalism and textual syntax will be used to specify model-to-model transformation. TRL is a language used to express queries and transformations on models in compliance with the metamodelling principles defined in the MOF 2.0 standard. TRL is under development and is part of the proposal of the Open QVT consortium in response to the MOF2.0/QVT RFP.

There are many alternatives for the implementation of the transformation rules and for code generation. A standard textual editor is being used to define the rules in TRL (XEmacs from GNU). The rules are being compiled and tested using the TRLengine model transformation prototype provided by FT.

For the PIM/PSM model-to-model transformation, the FT implementation is considering either to use the output of the TRLengine tool or to implement the rules using the J language supported by the Objecteering tool. For the last alternative, a UML profile associated with the Voice PSM metamodel is necessary and has to be defined.

For code generation rules the alternatives are to implement them using the J language within the Objecteering tool, to use the facilities for tool generation provided by the ArcStyler tool or to use the APIs provided by the Univers@lis tool [12]. Univers@lis is a model repository tool that allows one to store object-oriented models. Model elements are represented as instances of metaclasses that are defined according to an object-oriented metamodel. Univers@lis provides support for the UML 1.3 metamodel and the MOF 1.3 metamodel. Other metamodels can be supported by simply importing their specification into the tool.

An important requirement for implementing the model-to-model and code generators is the ability to access the input models, using an API or a dedicated model manipulation language, and the ability to export the output models once they are generated. The latter requirement is crucial to preserve tool independence.

For the definition of the PIM voice metamodel and the PSM voice metamodel any UML tool providing class diagram support and XMI externalization can be used. For the definition and implementation of the PIM-level voice UML profile, we intend to use the MDA tool provided by Objecteering.

Figure 11 shows the results of the detailed preparation phase represented in terms of a refined software development process showing the necessary models, metamodels and transformations, and their relationships.

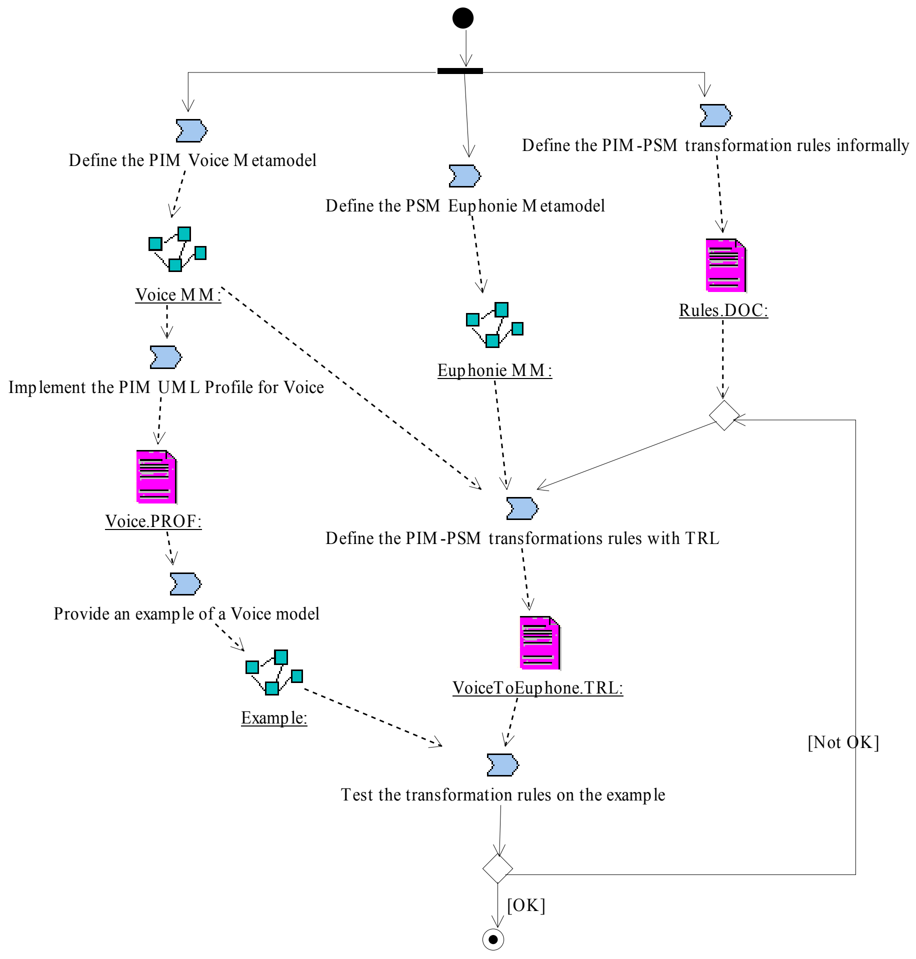
Define the PIM Voice Metamodel
Define the PSM Euphonie Metamodel
Define the PIM-PSM transformation rules informally
Voice MM:
Euphonie MM:
Rules.DOC:
Implement the PIM UML Profile for Voice
Voice.PROF:
Define the PIM-PSM transformations rules with TRL
Provide an example of a Voice model
VoiceToEuphone.TRL:
Example:
[Not OK]
Test the transformation rules on the example
[OK]

Figure 11. Result of the detailed preparation phase

6.4. Project execution (tool chain)

One of the purposes of this case study is to apply our MDA-based methodology to re-implement an address book service that has been developed before using a traditional (non-MDA) approach. Therefore, requirement analysis has already being done before for this service.

The dialogs of the address-book voice application are being specified in UML using the Voice UML Profile. The Objecteering UML modeller is being used for that. These dialogs were originally specified using a text-based proprietary formalism. The XMI exporter within the UML tool is used to export the model to the other tools, in particular the TRLengine used for model transformation.

Verification and validation on the models is planned to be performed directly within the Objecteering UML tool. We use the validation rules implemented using the J language for that purpose.

The implementation of the model to model transformations and code generation are considered as part of the preparation activity. During the execution phase, the end-user only needs to know how to invoke the transformation. We plan to add menu items within the Objecteering Voice UML profile to facilitate the invocation of the transformations for the end-user. For this purpose Objecteering should allow the definition of specialised menus.

The Java environment is used for coding and testing. Only some specific business operations are being implemented in Java. So far, no decision has been taken on the automation of the integration and the deployment phases.

7. Conclusions

A development methodology should define guidelines to be used in a development project, in terms of the necessary activities, roles, work products, etc. The methodology presented in this paper gives such guidelines and combines them with the concepts and principles of the MDA. The methodology itself is under development and its application on the case studies that are being performed in the MODA-TEL project will certainly provide the necessary feedback and refinement to improve its applicability.

An MDA-based development trajectory can require many different metamodels, models, transformations and their supporting tools. From our experience with the case study we can conclude that the MDA approach requires that the engineering process is explicitly described and documented in terms of the necessary work products and activities, such as illustrated in Figure 10 and Figure 11. The explicit definition of the engineering process makes an MDA-based project manageable.

References

[1] J.P.A. Almeida, M.J. van Sinderen, L. Ferreira Pires, D.A.C. Quartel. A systematic approach to platform-independent design based on the service concept. In *Proceedings of the Seventh IEEE International Conference on Enterprise Distributed Object Computing (EDOC 2003)*, Brisbane, Australia, September 2003.

[2] http://www.modatel.org

[3] MODA-TEL project. *Deliverable D3.1: Model-Driven Architecture definition and methodology*, 2003. Available at http://www.modatel.org/public/deliverables/D3.1.htm

[4] MODA-TEL project. *Deliverable D3.2: Guidelines for the application of MDA and the technologies covered by it*, 2003.
Available at http://www.modatel.org/public/deliverables/D3.2.htm

[5] Object Management Group. *Software Process Engineering Meta-model V1.0* (SPEM), formal/02-11-14, November 2002

[6] Object Management Group. *MDA-Guide, V1.0.1*, omg/03-06-01, June 2003

[7] Software Engineering Institute. *The Capability Maturity Model: guidelines for improving the software process*. Carnegie Mellon University. Addison Wesley Publishing Company, 1995

[8] ITU-T. Recommendation Z.150: *User Requirements Notation (URN): Language requirements and framework.* Geneva, February 2003.

[9] W3C. *Voice Extensible Markup Language (VoiceXML) Version 2.0*. W3C Candidate Recommendation. February 2003. Available at http://www.w3.org/TR/voicexml20/

[10] http://www.objecteering.com/

[11] *Response to the Query/Model/View RFP*. Revised submission 1.0. August 2003.
Available at http://www.omg.org/cgi-bin/doc?ad/03-08-05

[12] http://universalis.elibel.tm.fr/index.html