Applying MDA to Voice Applications: An Experience in Building an MDA Tool Chain

Maria José Presso and Mariano Belaunde

France Telecom, Div. R&D,
2, Av Pierre Marzin, 22307 Lannion, France
{mariajose.presso, mariano.belaunde}@francetelecom.com

Abstract. Before a development project based on MDA can start, an important effort has to be done in order to select and adapt existing MDA technology to the considered application domain. This article presents our experience in applying MDA technology to the voice application domain. It describes the iterative approach followed and discusses issues and needs raised by the experience in the area of building MDA tool chains.[1]

1 Introduction

Interactive voice-based applications are specific telephony applications that are designed to allow end-users to interact with a machine using speech and telephone keys in order to request a service. The interaction – called a dialog –typically consists of a state machine that executes the logic of the conversation and that is capable of invoking business code which stands independently of the user interface mechanism – could be web, batch or speech-based. Because state-machines can be specified and modelled formally, it is possible to design a tool chain that automates large amounts of the dialog implementation. The application of model-driven techniques to this domain is without any doubt very promising. However, the question that arises is about the methods and the cost of building a complete environment capable of taking full advantage of models: not only ensuring automated code production but also offering user-friendly interfaces to designers, model simulation and test generation.

A general methodology for MDA-based development has been defined in [1]. The authors define the main phases, and make a distinction between preparation and execution activities: execution activities refer to actual project execution, during which software artefacts and final products are produced, while preparation activities typically start before project execution, and setup the context that allows the reuse of knowledge during the project. The preparation activities can be seen as selecting and adapting existing generic MDA technology in order to define an MDA approach for the considered application domain and provide an appropriate tool chain.

[1] The work presented here has been partially carried out within the MODELWARE project. MODELWARE is a project co-funded by the European Commission under the "Information Society Technologies" Sixth Framework Programme (2002-2006). Information included in this document reflects only the author's views. The European Community is not liable for any use that may be made of the information contained herein.

Whereas [1] gives a general description of preparation activities and their chaining, there is currently little or no guidance available for them. In the current state of MDA technologies, these preparation activities demand an important effort which is not paid of by the first project and should be shared among a set of projects within the same domain. In particular, preparation has an impact on the first application project that follows it, as this first project necessarily requires iterations in the preparation activities.

This work presents our experience in building an MDA approach for the voice application domain and finishes by a discussion on the encountered issues and expectations.

2 MDD Preparation for the Voice Application Domain

According to [1], the preparation activities are divided into the *preliminary preparation phase*, the *detailed preparation phase*, and the *infrastructure set-up phase*. The *preliminary preparation* comprises the identification of the platform, the modelling language identification, the transformation identification and the traceability strategy definition. *Detailed preparation* comprises specification of modelling languages and specification of transformations. *Infrastructure setup* includes tool selection and metadata management.

In our project, these activities where performed in an iterative and incremental way, in order to better suit the needs of the users of the MDD environment involved in the execution phases, like voice dialog design, business application coding and functional testing. These users apply the tool facilities constructed by the preparation activities to produce the voice applications (the tool facilities are described later).

The preparation took place in three main stages. In each phase, some preparation activities were executed, together with some validation activities involving future users of the tool-chain, mainly service designers. The rest of this section presents the stages we followed.

2.1 Stage 1: Definition

In the first part of this stage, the current process of voice application creation was analysed and the integration of MDD techniques to this process was studied in order to identify the requirements for the voice development environment (VDE).

From this study, the following roles and their corresponding scenarios of use of the VDE were identified:

– the **service designer** uses the VDE model editor and simulator to model and simulate iteratively the dialogs of the application,
– the **usability practitioner** uses the VDE simulator to perform usability expertise on the dialogs of the service and he uses the VDE model editor to correct the dialog model,
– the **internal customer** (project owner) uses the VDE simulator a prototype of the service, to validate dialog design,
– the **service implementer** implements the service in the target platform, ideally by completing the skeletons generated by the modelling tool,
– the **service validator** produces the conformance test cases using the VDE test generation tool.

In order to serve as a conceptual basis for the VDE, a meta-model for platform independent modelling of voice applications was defined and UML 2 was chosen as a concrete syntax. Thus, a UML 2 profile for voice application models was defined[2].

Although the choice of UML for the concrete syntax may seem obvious, UML 2 being "the" modelling language standard, we will see later that this choice induces a significant cost in the infrastructure setup phase. For this reason, it is important to recall the rationale behind his choice:

- Voice application logic can easily be assimilated to a reactive state machine: the application reacts to user input such as voice and telephone keys, and produces output for the user: the vocal messages. The concepts of states and transitions are used in the voice application meta-model and supported by UML.
- Voice applications usually interact with the enterprise's information system. As UML is used as a modelling language in the information system domain, using the same language for voice applications allows to seamlessly integrate information system models with voice application models.
- Communication services are becoming integrated and multimodal. The use of a standard, largely used notation is expected to favour future integration with other services and modalities.
- Existing modelling tools and skills can be reused.

Also at this stage, the architecture of the VDE was defined (see Figure 1) and some of the tools involved were selected. A UML tool was chosen to play the role of model editor and model repository and the criteria for its selection were defined. Among the criteria defined, the ones that differentiated the tools were : i) the support for UML 2 transition oriented syntax for state machines (this notation had been found easier to read by users than the state oriented syntax), ii) the support for rigorous syntax checking (in particular for actions) and iii) model simulation/execution capabilities. TAU G2 from Telelogic was chosen as modelling tool.

Following this first round of preparation activities, we conducted some experiments in order to determine the ability of the profile to capture the intended service logic, verify that service designers could feel comfortable with the tool and to identify the necessary adaptations to the modelling tool (i.e. specific functionalities necessary to better support the voice application profile). These experiments consisted in modelling some existing services with the proposed profile and tool. Modelling was initiated by a modelling expert and a service designer working in pairs and finished by the service designer.

As a result of this phase, we could see that the proposed profile captured most of the necessary elements to describe a voice application, but it should be enhanced to support executable modelling of messages and the definition of grammars for voice recognition. Designers (who are not modelling experts) could get familiar with the modelling tool with a fair amount of effort. The main need for tool adaptation that came up was that a high level of support for the specification of messages allowing to reuse message parts, as well as facilities to read them during the dialog specification task. Also, high level commands for the creation of the other domain elements should

[2] This profile was later used as a basis for a submission to the OMG's RFP for a Metamodel and UML Profile for voice applications [3].

be available (such as creating a dialog). The restitution of the specification in the form of a document should be optimized in order to limit its volume and improve readability, hyperlink navigation should be available in the documents.

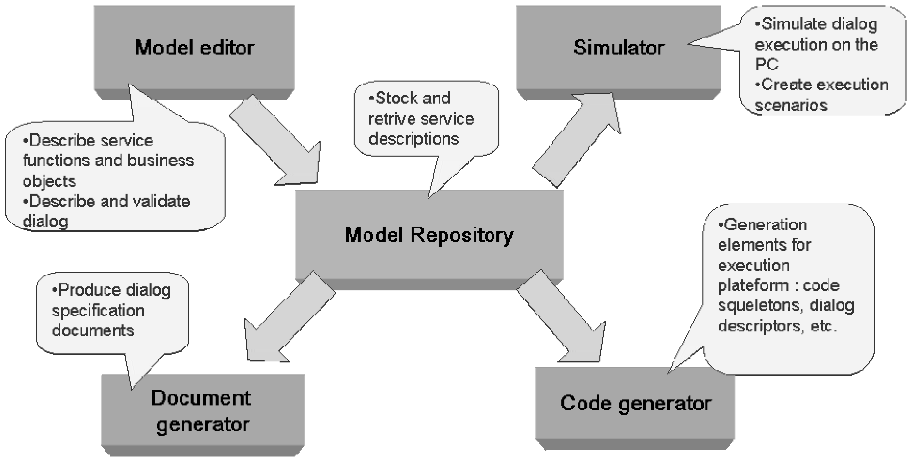
Model editor
Simulator
•Simulate dialog execution on the PC •Create execution scenarios
•Stock and retrive service descriptions
•Describe service functions and business objects •Describe and validate dialog
Model Repository
•Generation elements for execution plateform : code squeletons, dialog descriptors, etc.
•Produce dialog specification documents
Document generator
Code generator

Fig. 1. Architecture of the MDD Voice Development Environment (VDE)

2.2 Stage 2: VDE Development – Iteration 1

The second stage consisted mainly in the development of a first version of the tool chain, offering assistance for the creation of dialog modelling elements (dialogs, messages, recognition interpretation concepts, etc.), documentation generation and a limited form of dialog simulation. This version was developed using scripting and code generation capabilities provided by the modelling tool. Concretely, it appears as a plug-in to the modelling tool and a separate telephone-like GUI connected to a text to speech engine, that allows the designer to execute the dialog logic of the modelled service and evaluate its appropriateness, its ergonomics, etc.

After the development of this first version, a second row of experiments took place, which consisted in using the tool chain to enhance the service models produced in the first stage, and use the document generation and simulation functionalities for this models.

Although this first version of the tool chain was very promising and showed that useful functionality for service creation could be offered, it presented some limitations: the GUI for modelling assistance that could be developed through scripting was limited and not satisfying from the point of view of ergonomics; the service simulation was not able to propose the possible inputs in a given situation, neither to go arbitrarily back and forth into the execution tree. The scripting technique used for development posed maintainability issues and was not appropriate to support a growing software.

At the same time, studies where carried out about simulation and test generation for voice services. This studies showed that existing simulation technologies based on the IF language[2] provided the necessary level of support to build a simulator for voice services that overcomes the limitation of the method employed in the first version.

At the end of this stage the decision was taken to build a more industrial version of the tool chain based on a programming language (rather than scripting), to offer richer GUI capability in particular for message creation, and to provide the service simulation functionality through model simulation techniques, rather than code generation.

2.3 Stage 3: VDE Development – Iteration 2

This stage started by the definition of the architecture for the modelling tool plug-in, and the choice of the implementation technology. The main characteristic of this architecture was the definition of a layer that provides a view of the underlying UML model in the terms of the voice application metamodel. This layer implements an on-the-fly bi-directional transformation between UML and the voice application metamodel. This layer provides an adapted API and is used as a basis to develop the GUI and a set of generators that implemented various model transformations. Also, the architecture proposed a way for simple integration of the different generators in the plug-in. The generators provided at this stage were:

- document generators, which produce documents according to different templates and in html and MS Word formats. These generators use an intermediate XML generation phase, followed by XSLT transformations,
- an XMI generator, which exports the model in the terms of the voice application metamodel. This generator uses the adaptation layer API, and was automatically generated (and re-generated as needed) from the voice application metamodel,
- a generator that produces an IF model for service simulation and test generation,
- a code generator having as target an n-tier architecture using VoiceXML. The generated code executes in the application server tier and produces on-the-fly the presentation pages in VoiceXML. The generated code integrates in a framework (which was also developed in this phase), that provides the basis for the execution of a dialog state machine and VoiceXML generation.

The first three generators above were directly integrated into the plug-in, in order to facilitate the installation of the toolkit in the user's workstation and its use by the service designers, while the last on is external and uses the results of the XMI export.
Something important to note about this stage is that the metamodel was called to change often, as the implementation of transformations asked for corrections or improvements. As different transformations were developed in parallel, the changes asked by one of them had an impact on the others.

2.4 Stage 4: Pilots

The last stage is that of pilot projects. These are the first projects using the MDD chain (in the terms of [1], the first runs of the "execution" phase). The stage is still in progress at the time of writing. During this phase, iteration with preparation activities goes on, mainly to adapt the code generator to the project's target platform and to add extra functions asked by the pilot projects. An important effort in this stage is spent on user training and support.

3 Discussion

The result of the preparation activities described in the previous section is a process and a tool chain providing a high degree of automation. Starting from the PIM model, the tool chain produces automatically a simulation of the dialog, functional test cases, and the executable code for the dialog logic, and a is good representative of the MDA vision. However, these activities consumed an important effort, that should be shared by several projects. In this section we briefly discuss some of the issues and expectations that come from our experience in building this MDA tool chain.

In our experiment, we encountered a strong user's demand to have a rich GUI for modelling in terms of the domain vocabulary. This appeared as a critical issue for the adoption of the tool chain. As UML had been chosen as a concrete syntax, this request lead to important extensions to the modelling tool in the form of a plug-in. The ability to extend the modelling tool using a full-fledged programming language was necessary to develop the required GUI and to apply good engineering practices (such as the MVC pattern) to this development.

The above issue comes to the famous problem of whether a general-purpose modelling tool should be specialized or whether the tool should be built from scratch. Beyond tool usage is the question whether the specific language for the considered domain – in our case voice dialog definition – has to be built on top of an existing language – like UML, or a new language should be defined – typically using MOF or an XML schema. It is easy to adhere to the principle of maintaining the distinction between the abstract syntax (the domain metamodel) and the concrete syntax (given by a UML profile or a textual notation) since it provides potentially much more freedom – ability to use various concrete syntaxes - and is more comfortable for domain designers – since the vocabulary used is directly the one of the domain. However, as our experiment has demonstrated, maintaining this distinction potentially induces a high cost to the development of the tool chain. In our experiment, we used an "API adaptation" technique which allows to program a specific GUI and model transformations within the UML tool by using an API dedicated to the domain metamodel – instead of using the general-purpose UML-based API. This technique presents interesting advantages from the engineering point of view : i) the knowledge about the mapping between UML and the domain metamodel is localized, ii)the coding of the GUI is simplified, since the complexity of UML is hidden iii) the model transformations can be implemented in domain terms and are thus facilitated, and iv) the XMI generator exporting the model in the voice metamodel terms can be produced and updated automatically from the metamodel itself. However, one of the important problems encountered at this level was the instability of the metamodel, which in general changed more often than the graphical and the textual notation. Ideally, to solve the instability problem, the mapping between the metamodel and the concrete syntax should be defined in one single place and then the "API adaptation" generated automatically. This is indeed easy to say but not necessarily easy to put to work, especially when the evolving API is already used in various places.

To conclude with the "API adaptation" technique, our experiences showed us that this technique has good properties, but is costly when the metamodel is not stable and there is no specific advanced support for maintaining mapping coherence. At the moment we don't know what the cost would be without this technique, may be it would be higher. In other words there is a need for tools that will offer an explicit support of API adaptation. The current notion of UML profile is not sufficient as a specialization mechanism since it only addresses notation customization but not API customization.

Concerning metamodel stability, the implementation of code generation and simulation model generation asked for much more changes to the metamodel than documentation generation and GUI development. In our experiment this meant that the metamodel changed more during stage 3 than stage 2. A possible conclusion from this observation is that transformation development should happen earlier in the preparation phase, in order to rapidly stabilize the metamodel, before heavy development such as tool adaptation take place. In order to put this scheduling in place, we need a way to easily produce metamodel compliant models to serve as an input to test model transformations. Tools able to rapidly produce a friendly GUI or text notation from the metamodel definition could be very useful in this situation.

Another interesting issue is about reuse of metamodel patterns. The PIM Voice metamodel reuses various common constructs that are already found in UML, such as the differentiation between operation definition and operation call – which has a variant in dialog definition versus sub-dialog invocation concepts. The UML2 infrastructure metamodel is currently defined using an extensive package decomposition and the merge mechanism is intended to pick the packages that are needed. In theory, the semantics of the imported merged concepts is preserved. UML Profiles is a restricted way to perform metamodel extension with semantics preservation of the reference metamodel. Ideally, we would like to see a tool that will allow plain metamodel extension that will ensure – when applicable – semantic preservation of the reused patterns. This would be an option to the "API adaptation" technique described above that could avoid maintaining the distinction between the concrete and the abstract syntax.

A last issue concerns the need of providing an environment that integrates the different parts of the tool chain in a single workspace to improve easiness of use and installation. This is still a difficult matter with existing MDA technology. In our experiment this need led us to integrate some of the transformations directly into the modelling tool plug-in, where a model transformation engine would be more appropriate from an MDA point of view. In spite of this integration effort, we still needed to deploy additional tools and technology in the users' site and decided to host code generation in a web server, to simplify the installation on the developers' site. Better integration of MDA tools is necessary to easily build and deploy an integrated MDD environment for a given domain.

4 Conclusions

Preparation activities are an important and unavoidable phase in an "MDA development trajectory". We have presented an experience which applied an iterative approach to preparation activities and led to an MDA tool chain with a high level of automation. These preparation activities need an important effort and thus impact on MDA ROI. Better specific tool support and integration are necessary to lower the cost of this phase and improve the integration level of the resulting tool chain.

Acknowledgments

Several people contributed to the design and implementation of the presented tool chain : Miguel Alabau, Fabrice Dubois, Gregoire Dupé, Mikaël Marche, Sébastien Poivre (France Telecom Div R&D) , Charles-Henri Jurd (Telelogic), Eric Bousquet and Cedric Le Maitre (Capgemini).

Special thanks come to Gregoire Dupé and Sébastien Poivre for interesting discussion on issues and possible improvements on the development process of an MDD tool chain.

References

1. A.Gavras, M.Belaunde, L. Ferreira Pires, J. P.A Almeida. Towards an MDA-based development methodology for distributed application, in EWSA 2004 : 230-240.
2. M. Bozga, S. Graf, I. Ober, I. Ober and J. Sifakis. Tools and Applications II: The IF Toolset. In Flavio Corradinni and Marco Bernanrdo, editors, Proceedings of SFM'04 (Bertinoro, Italy), September, 2004 LNCS vol. 3185, Springer-Verlag
3. Joint Initial Submission to the UML Profile and metamodel for voice-based applications RFP. August 2004.