Using BPEL as a workflow engine for local enterprise applications

Nicolas Biri, Pascal Bauler, Fernand Feltz, Nicolas Médoc, Céline Thomase

Centre de Recherche Public – Gabriel Lippmann
rue du Brill 41,
4422 Belvaux
Luxembourg
{biri, bauler, feltz, medoc, thomase}@lippmann.lu

Abstract: This paper gives an overview on the integration of a BPEL workflow engine into an enterprise application in order to decouple business processes and application code. The technical complexity of this innovative approach is hidden by means of Model Driven Software Development (MDSD) techniques and several component frameworks. By referring to a research project realised in collaboration of the Centre de Recherche Public – Garbiel Lippmann and the Luxembourg National Family Benefits Fund (CNPF), with the overall goal to optimise the IT environment of the CNPF, this paper shows how the proposed approach is particularly adapted to agile and iterative development projects.

1 Introduction

The project presented in this paper is realised in collaboration with the Luxembourg National Family Benefits Fund of Luxembourg (CNPF (Caisse Nationale des Prestations Familiales)). This administration is in charge of the payment of family allowances for people working in Luxembourg. During the last few years, the CNPF has started a modernisation and optimisation process highly relying on information technologies. There are mainly two reasons for this process: First of all, the modernisation is mandatory to enable the handling of the growing workload and the complexity of the underlying treatments, which mainly result from the increasing number of borderline commuters and the extension of the European Union. A second reason for this modernisation effort consists in offering adequate e-government solutions improving the quality of service offered to the citizens and so increasing the interest of the population in modern computer technologies. Due to the geographical situation of Luxembourg and the high number of borderline commuters, complex cross-administration and cross-country solutions have to be designed and data exchange protocols have to be specified, in order to enable the access to the heterogeneous IT environments of the different neighbouring countries. Strategic investment in modern IT solutions is justified, as the workforce of the CNPF basically remained unchanged, although the number of commuters significantly increased over the last decade. After some preliminary and

internal discussions, the Centre de Recherche Public – Gabriel Lippmann got involved in the modernisation process [Ba06, HF06], to design innovative solutions. In this paper we present a part of this modernisation project by mainly focusing onto the design of enterprise solutions handling the computations of family allowances for commuters. We show how modelling technologies combined with agile management concepts, significantly help to successfully accomplish this project and to move into production.

Considering the short development phases and the numerous changes required in the IT environment of the CNPF, we heavily rely on the SCRUM concepts [SB01] to drive the project. This approach is well adapted for projects with short deadlines running in an evolving context. The key principle of this method inspired by the agile development method, is to define priorities on the requirements and to establish short incremental development cycles (called *sprints*), each of these containing a few goals to achieve (called *backlogs*). Several operational and functional tests have to be passed before a development cycle can be closed. Furthermore, a development cycle is usually ended by a practical demonstration involving the partner and the end-user. In the particular context of the CNPF we consider 2 types of demonstrations, either showing newly designed business functionalities or discussing the progress in the specification protocol with the neighbouring countries. This differentiation between operational and specification results, is due to the need to collaborate with foreign development teams often relying on the Waterfall model.

To take maximum advantage of the iterative development cycles introduced by the SCRUM approach, the used technologies and development framework are selected based on their compatibility with iterative development. This stresses the decision to realise the application flow by means of executable workflows, in addition Model Driven Architecture (MDA) and Model Driven Software Development (MDSD) are used to generate technical and repeatable code segments and as a consequence, to improve the overall development process.

From a technical point of view, a BPEL (Business Process Execution Language) workflow engine is introduced to coordinate the core workflows and business processes of the proposed solution. The common usage of BPEL engines consists in orchestrating services (usually based on web-services) including handling of incoming messages, message transformation and message routing. BPEL engines offer by default a message centric approach, where the analysis of incoming messages determines further treatments, either by initiating new processes or by passing progress information to existing processes. Introducing appropriate MDSD frameworks, which hide technical aspects of the overall solution, facilitates efficient usage of BPEL. The design decision to build enterprise application architectures around a BPEL engine is justified as follows:

- The family allowances business domain is frequently adapted to political decisions and legal changes, which results in regular changes of the underlying business processes. By decoupling application code and application workflows, maintenance and enhancement aspects can be optimised.

- Furthermore, decoupling of application workflows and code is especially adapted to agile and incremental development projects. The IT teams can start with simplified workflow skeletons, which are systematically enhanced and adapted during the various development cycles of the project.

Below, these aspects are discussed in more detail. Section 2 presents the project context with an overview of the proposed solution. In section 3, the orchestration solutions and more precisely the BPEL specificities are exposed. The advantages and issues resulting from the use of BPEL as a workflow engine in an incremental development process, and its integration into our solution, are explained. This section also discusses some technical aspects required to avoid de-synchronisation between BPEL workflows and the application code. Section 4 explains how MDA technologies (Model Driven Architecture) facilitate this synchronisation and how this technology fits with the agile approach. Section 5 concludes this paper.

2 Project overview

2.1 Project working plan

The general project goal is to automate the computation of the family allowances for the people working in Luxembourg. We distinguish the family allowances for Luxembourg citizens on one side and for commuters on the other side. The complexity of the family allowances results from a European decision saying that family allowances are exportable. So each person working in Luxembourg, independently of his or her residence country, is granted the Luxembourg family allowances. Furthermore the citizens get the allowances from where they are the highest, either from the residence country or from the working location. As in Luxembourg the allowances are higher than in the neighbouring countries, the practical situation is somewhat simplified. As a consequence, each family with incomes resulting exclusively from activities in Luxembourg is treated as resident in Luxembourg. The situation is more complex for families with incomes resulting from activities in different countries. The current procedure consists in having the residence country pay the family allowances on a monthly basis. The difference between the local and the Luxembourg's allowances are calculated twice a year and are directly paid to the citizen. As this process is error prone and tedious, a first project goal is to replace this process in order to pay the allowances on a monthly basis and to delegate eventual clearing operations to the back-end IT systems. This improvement however requires an excellent collaboration between the Family Allowances Funds of the neighbouring countries. Special political agreements have to be established before facing technical burdens related to the heterogeneous IT environments. These technical issues are discussed in detail in the following paragraph.

Due to the high increase of the number of commuters in Luxembourg, the CNPF noticed in 2001 that they were no longer able to handle all the files manually. At that time, roughly half of the commuters came from France. That justified the decision to tackle the French commuters in priority and to start discussions with French allowances offices. An interesting factor was that the French Family Benefits Fund is organised in a semi-centralised way, with every region relying on an independent family fund, however all IT services being provided globally by the French National Benefit Fund (CNAF).

After some delays, the CRP-GL got involved to work on an innovative approach to sort out these issues and to work out a project plan to tackle the commuter problem. To find an answer to this tricky situation, a two phases plan was defined. The first phase had to quickly realise a production ready system, able to handle the French border commuters. The proposed solution imported family allowances data from the French local benefit funds of Metz and Nancy and computed on a semestrial basis the difference between French and Luxembourgish allowances. Development started beginning 2005 and this semi-automated solution went into production in August 2005. It was extended to Belgian and German commuters in 2006. This phase, which can be considered as a preliminary work, is not being discussed in detail in this paper. The second project phase consists in developing an extendable IT system able to offer fully automated handling of the French commuters. This solution must perform the monthly computation of the family allowances and synchronise data between Luxembourg and French Family Funds. The results of this second project phase are currently in a pre-production phase at the CNPF and production is scheduled for October 2007. This modular system is supposed to be extended in order to handle all Luxembourgish commuters within a 2 years timeframe.

2.2 Solution description

As mentioned above, this chapter puts the focus onto the second project phase. The proposed solution had to show operational results within 12 months, while it had to remain extendable to handle on a mid-term basis the family allowances for all neighbour countries. Another key aspect of the proposed solution was to offer extensive verification and validation mechanisms, in order to avoid incorrect or double payment of the family allowances.

The error detection is particularly tricky as the French and Luxembourgish Family Allowances Funds are involved. An extensive exchange protocol, composed of 3 sub-components, had to be specified and implemented to automatically handle those error conditions:

- The first part details how master data concerning the citizens involved in the cross-border processes are exchanged between the French and Luxembourg Family Benefits Fund. This section of the protocol specification also defines the active process for a given citizen, with eventual suspensions of the payments for a given month.

- The second part consists in a detailed error handling protocol. When abnormal situations are detected, the family funds are informed and the payments are suspended. Dedicated message exchanges have been defined to deliver status updates to the peer country in order to avoid incorrect payments. Due to the international character of these processes, it is indeed hard to get badly paid money back, especially if the involved citizens no longer live in the involved countries.

- The last part of this communication protocol handles normal/regular data exchanges between the French and Luxembourgish benefits funds. The exchanged data inform the peer country of the paid allowances and define the appropriate feedback.

During this project, a close collaboration between the French and Luxembourgish IT teams is mandatory in order to overcome organisational constraints. In addition, the communication protocol has to take the differences between the French and the Luxembourgish IT systems into account and to guarantee compatibility with both environments. Luxembourg has the advantage of being able to start with a new IT system with limited historical data and no technical constraints. The French environment is mainframe based and in production since several years. All new developments have to be carefully thought through, realised and tested. By no means new developments may negatively impact the running processes and the daily operations. The different design methodologies adopted by the two development teams also has consequences onto the working plan and the project schedule. The Luxembourg team uses agile development approaches based on the Scrum concept, whereas the French team uses the classical waterfall approach. As a consequence, the data exchange protocol has to be specified and implemented following a waterfall approach. As a consequence this sub-task has to be decoupled from the back-end system design.

This second project phase started in September 2005 and, since March 2007, is progressively moving into production stage.

Below we concentrate on the Luxembourgish part of the cross-border project, by exposing the design decision, the business processes and data manipulations at the CNPF. The core application is built around the validation workflow, which consists of several controls depending on the particular situation of the involved citizens and the corresponding allowances. The main workflow collects the appropriate data, coordinates the validation process and computes the appropriate results.

3 Integrating BPEL in a local application

3.1 Motivation

The first definition of the business processes are realised by means of EPC (*Event-Driven Process Chain*) ARIS diagrams in close collaboration with the CNPF business analysts. The EPC diagrams are handed over to the project team and are manually translated into BPEL workflows. We use a BPEL engine to handle workflow execution, as it is an orchestration language for web-services, initially designed by IBM and then standardised by OASIS. The language provides a way to describe the behaviour of business processes enabling transactions with remote services and ensuring interactions between them. The basic tasks of BPEL are service calls, message reception, message filtering, conditional routing and a compensation mechanism to recover from external failures. The processes are executed inside a BPEL engine, which offers management functionalities like the creation and the termination of processes according to a basic lifecycle mechanism.

Using a workflow engine to execute the underlying business processes provides an easy way to separate the behaviour of a process from the rest of the business logic. Thanks to this property, we can easily adapt a process to new requirements. Changing a test or adding a service access to a process can easily be done as the workflow is clearly separated from the rest of the code. Furthermore, most of the BPEL process editors provide a graphical representation of the BPEL process. Even if these representations are not normalized, they are informative enough to be used during discussions with non-IT people at the CNPF.

The environment of the CNPF and the nature of the processes lead our choice towards the BPEL solution. The use of BPEL is especially adapted when orchestrating fully automated workflows without human intervention, running in a distributed IT environment [Si05]. In our particular project context however, the prime goal of BPEL is to coordinate local services and to ensure execution of business processes. As a consequence several adjustments are mandatory to adapt the BPEL engine to our special needs.

3.2 Benefits of BPEL combined with agile development

This section shows how, in an iterative development environment, BPEL workflows can facilitate the definition of business processes. We want to emphasize, that a BPEL based business process definition approach, is particularly adapted to small and incremental development cycles. The defined business processes have to be generic and flexible enough to follow the actual status of the underlying development of business functionalities. In practice, business processes have to be extended on a regular basis in order to integrate new business functions realised by the development teams.

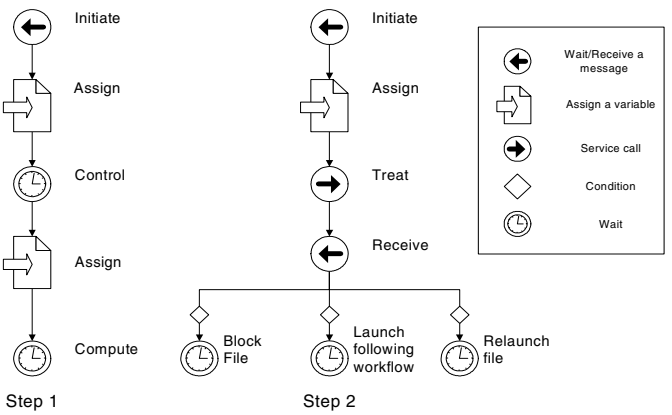

Figure 1. Example process: the first steps

The first modelling task consists in defining an ARIS EPC skeleton of the underlying business process. This initial BPEL process contains some place holders in form of wait actions, which are progressively replaced over the various production cycles. The next modelling steps consist in refining these processes. We distinguish between two kinds of refinements:

- Refinements, which integrate newly implemented tasks and business functions
- Refinements which modify the initial process by adding new structural elements in order to represent significant workflow changes

A typical example of these types of refinements of the BPEL workflow is given in Figures 1 and 2. For a better readability, we use a graphical representation of the BPEL processes. In Figure 1, we have the first version of the process, where potential external calls are replaced by wait tasks. In the second step we use two new features: the control task and a choice for the last step of the process. In the last step presented in Figure 2, we introduce a loop on the different files controlling the received data and an error handling. The left part of the "loop box" corresponds to normal behaviour; the right part corresponds to the error handling.

This example shows that BPEL processes are well adapted to iterative enhancements through short development cycles. The systematic refinement of the processes has two advantages: the process can easily be adapted to only access available services and though have testable workflows, and the multiple cycles of development give us many opportunities to correct possible errors introduced in previous modelling phases.

3.3 Integration issues

As the BPEL engine is used in an unusual way to orchestrate local services inside an enterprise application, we have to adapt the underlying architecture as well as the behaviour of the BPEL engine to fit these special needs. The architectural changes and adaptations are discussed in this section together with some best practices identified during the modelling process of the BPEL workflows.

The BPEL standard heavily relies on web-services. All communication with the BPEL engine relies on this technology, which introduces a certain performance overhead. Extensive performance tests however showed that this overhead is marginal compared to the underlying computations and as a consequence, the proposed solution mainly has to try to optimise the number of required web-service calls.

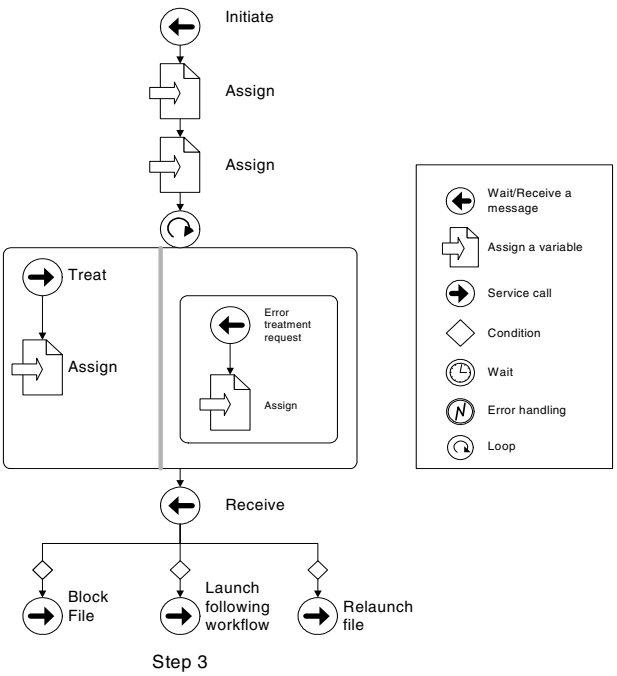

Figure 2. Example process: the final step

Using a BPEL engine at the core of the enterprise application requires the development of appropriate communication interfaces between the workflow engine and the other parts of the project. We consider three types of communication:

- Communication of Data from the application to the BPEL engine: the application server sends messages to the BPEL processes deployed as web services on the BPEL server. These messages can either start a new process or respond to an active process waiting for a specific event.

- Message flow from the BPEL engine to the application: the BPEL server accesses business functionalities deployed as web services on the application server. We use a facade pattern [Ga97] to realise the interface between the web services and the real implementation of these functionalities.

- Communication inside the BPEL engine: the main process dispatches incoming messages to the appropriate BPEL sub processes. This is done using the correlation feature offered by the BPEL language. Each sub process is accessed as a web service by the main process.

The main difficulty in this collaborative context is to ensure coherency between the processes related to the various actors in the communication. As the BPEL solution is process oriented, it is message centric. This means that the behaviour of the processes depends on the received messages and is not state-transition oriented. Consequently we have to integrate a mechanism introducing this notion of state inside the processes. This is done by means of a coordination component in charge of the synchronisation of the application state and the BPEL workflow state. The proposed coordination component can be divided into 4 parts:

1. A *State Machine* framework deployed on the application server. This framework is used by the enterprise application to trace the expected state of the BPEL process.

2. The coordination component offers query possibilities on the BPEL engine, checking if the process states inside the application code and the workflow engine are identical.

3. Automated handling of error conditions also relies on the coordination component to restore consistent status at the application and the workflow level. This error handling may result in rollback operations.

4. An event correlation module makes sure that incoming messages only influence the concerned processes. For instance, a main BPEL process catches all the incoming messages and dispatches them to the sub processes. A special identifier determines the process instance concerned by a given message. A sub process catches an incoming message only if it is actually waiting for this particular type of message. This offers an additional degree of protection and increases the reliability of the overall solution.

4 Model Driven Software Development and agile method

Model Driven Software Development (MDSD) is a core technology of the proposed project. It helps to encapsulate most of the underlying technical aspects of the enterprise application and to focus development efforts onto the business part. The UML based platform independent model (PIM), describing the technical aspects of the project, is enhanced by several stereotypes to obtain a platform specific model (PSM). This PSM is used as input for the generator framework to produce platform specific code. In addition to the generation of the persistency layer by means of Enterprise Java Beans (EJB), several models define orchestration context behaviour. In this part, we present how MDSD is used in our incremental and collaborative conception process and we give some more information about the orchestration specific models.

4.1 MDSD development cycle

The proposed MDSD approach relies on several scientific papers explaining how the agile development paradigm can be applied to MDA [Me04] or to MDSD [St06]. The proposed project validates the use of the theoretical approach by a development team in a practical environment.

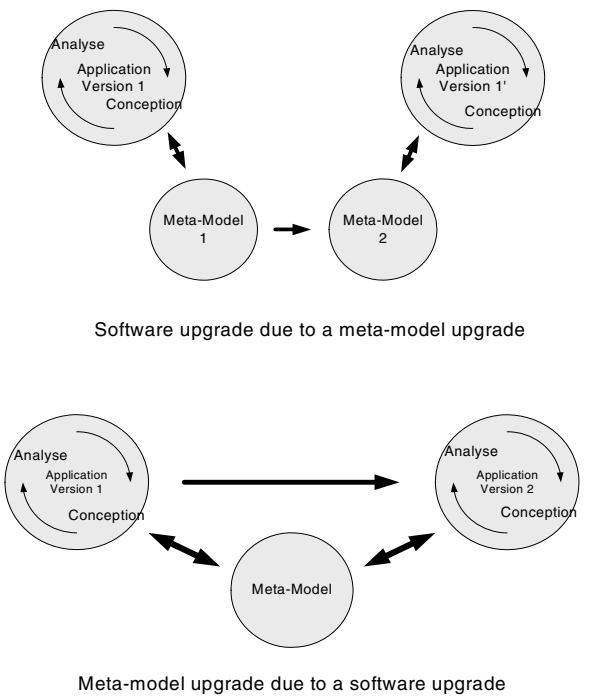

Figure 3. Correlated upgrade of the meta-model and of the application

Conceptually we distinguish the meta-model defining the general architecture and the application specific model. Both meta-model and application model evolve independently, have however mutually influencing side effects. Each development cycle relies on previous iterations, but may also require some adaptations at model, meta-model or code generation side. These changes may be caused by new functional requirements, which result in enhancements of the underlying meta-model or by architectural improvements within the meta-model. As a consequence, each development cycle can be seen as a new test for the robustness of the code and the generic aspect of the proposed models. Another advantage of the proposed approach is a quicker and more robust development. Our practical experiences are in line with the theoretical results on the common usage of MDSD technologies in an agile development environment.

Decoupling meta-model evolutions from the agile development cycles has positive side effects on the overall architecture and on the resulting enterprise applications. Figure 3 schematically shows the relationship between the application development and the meta-model evolution.

MDSD techniques significantly reduce the development effort when applied to repetitive or technical tasks, are however of little advantage when representing business logic or application specific code where manual coding is more efficient.

Another MDSD specific problem encountered during the above-mentioned project is that existing modelling tools offer only very limited multi-user support. Model sharing, or model versioning features are not ready for productive use and merging UML models is prone to error. As a workaround, we use a planning document to indicate who in the development team has ownership of the various models. This workaround introduces some overhead which is however fully acceptable in this particular project.

4.2 The State Machine model

In the overall MDSD approach we also introduce state machine support built around the state pattern proposed in [Ga97]. According to its definition, this pattern is applicable in the following context:

- The behaviour of an object depends on its state and it must change its behaviour at run-time depending on that state.
- Operations have large, multipart conditional statements that depend on the object's state. The State pattern puts each branch of the conditional structure in a separate class.

In our particular context, this definition exactly corresponds to the definition of the workflows state management. Each workflow contains two classes to handle its state transition: a *state* class that provides the core operation for state transition and a *context* class with the information that has led to the current state.

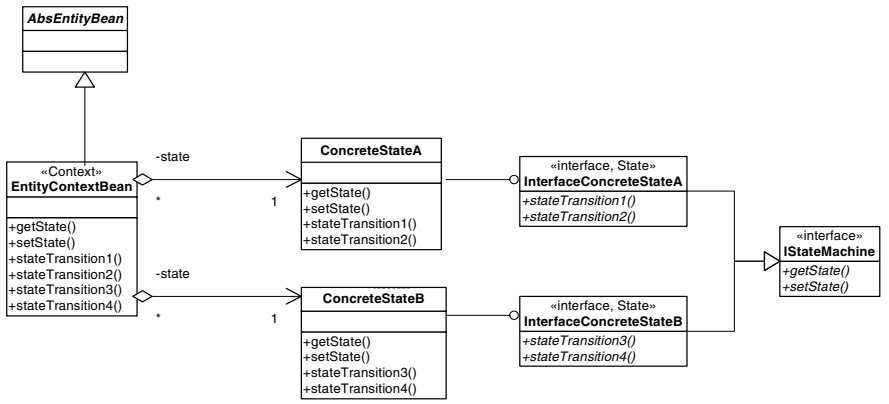

Figure 4. A State framework instance

The development team can limit itself to defining the major states (*ConcreteStateA, ConcreateStateB)* regrouping all states of a given state machine, as well as the appropriate transitions (*stateTransition1..4*). This UML model shown in Figure 4 is used as input and applied to the underlying meta-model. The code generator framework establishes the link with the abstract state machine, giving a generic behaviour to the workflows and with the persistence layer offering data persistence by means of Enterprise Java Beans (EJB). If the default behaviour is not appropriate, the developer may use inheritance mechanisms to overwrite the default.

5 Conclusion

This paper shows the benefits and difficulties encountered when integrating a BPEL engine into enterprise applications and when relying on BPEL processes to manage business workflows. The use of such a solution in an agile development process improves the flexibility of the proposed solution. It enables systematic enhancements of the business processes by adding new components to the workflow while maintaining a loose coupling between the enterprise application and the workflow engine. The resulting application shows significant better adaptability to changes. A key challenge of the proposed solution is to guarantee synchronisation between the application code and the workflow engine, by combining a message centric behaviour with a state-transition behaviour. This synchronisation component extensively uses various Model Driven Software Development techniques to offer an appropriate framework for integrating the enterprise application and the BPEL engine.

The overall experience of using a BPEL workflow engine in the particular context of Luxembourg National Benefits Fund is positive. The modernisation of the IT environment of the CNPF is ongoing while showing success stories, validating the underlying design decisions. Some additional conceptual complexity is added in the first project phases, which is compensated by a better adaptability of the overall solution. Further improvements concentrate on performance tuning in order to reduce the overhead introduced by the web-service approach of the BPEL.

Bibliography

[Ba06] Bauler P., Feltz F., Biri N., Pinheiro P., Implementing a Service-Oriented Architecture for Small and Medium Organisations, EMISA'06, Germany, 2006.

[Co05] Contenti M., Mecella M., Termini A., Baldoni R., « A Distributed Architecture for Supporting e-Government Cooperative Processes », E-Government: Towards Electronic Democracy, Lecture Notes in Computer Science, vol. 3416, Springer, p. 181-192, 2005.

[Ga97] Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley, 1997.

[HF06] Hitzelberger P., Feltz F., An Interoperable Communication Platform for a Public Agency. 5th international EGOV conference, Krakow, Poland, 2006..

[Me02] Martin R.C.: Agile Software Development: Principles, Patterns, and Practices. Prentice Hall, 2002.

[Me04] Mellor S.J.: Agile MDA, A white paper. http://omg.org/mda/mda_files/AgileMDA.pdf , 2004.

[Sc95] Schwaber K., SCRUM Development process, OOPSLA'95 Workshop on Design Patterns for Concurrent, Parallel, and Distributed Object-Oriented Systems

[SB01] Schwaber K., Beedle M., Agile Software Development with Scrum, Prentice Hall PTR Upper Saddle River, NJ, USA, 2001.

[Si05] Silver B., Agile To The Bone, Intelligent Enterprise, http://www.intelligententerprise.com/showArticle.jhtml?articleID=57702677, 2005.

[St06] Stahl T., Völter M., Bettin J., Haase A., Helsen S., Model Driven Software Development – Techonology, Engeneering, Management, Wiley, 2006.