A Methodology for Development of Web Service-based Business Processes

Dimka Karastoyanova
Technische Universität Darmstadt
dimka@gkec.tu-darmstadt.de

Abstract

This paper introduces a methodology for development of WS-based processes, also called WS-Flows. This procedure is based on a detailed WS-Flow life cycle. The procedure aims at automating the process of modelling and generating WS-flows definitions in multiple languages by making use of process templates and meta-programming techniques. Process definitions can be generated from templates of coordination protocols and thus provide support for standard B2B interactions. The procedure promotes the creation of an abstract process model and a complementary meta-data repository. One of the main principles followed in this methodology is deferring the selection of process definition language to a latest possible point in time and the commitment to concrete WS instances up until the execution of the process instance. This provides for the creation of WS-flows able to adjust to the changing business environment and exhibiting complex dynamic features during run time. The procedure and the process features it targets have certain implications on the process model and the infrastructures for executing this methodology. These implications are also briefly discussed.

1. Introduction

"Web services" is a technology aiming at application integration across enterprises and over the Web. It has become the emblematic embodiment of the service oriented architecture (SOA). SOA is an architectural paradigm that uses a distributed environment to expose, discover and manage service-oriented business logic. The three main participants in the SOA are service provider, service requester and service registry. SOA rests on the following basic principles: dynamic discovery of business logic, separation of service description and implementation, and composability of services. The Web Service (WSs) technology does not yet exhibit all features of a mature service oriented middleware technology, though [1], [8]. Development of complex WSs is one of the areas in which WS technology should further evolve, that is, to provide better support for recursive service composability – one of the features characterizing SOA. One way to create complex WSs is to use service composition as implementation (such WSs are also referred to as composite WSs). This requires providing a way to create business processes that use WSs for performing tasks on their behalf. Such processes are referred to WS-based business processes, or WS-flows [7]. In fact, there are already two specifications for composing WSs: the Business Process Execution Language for Web Services (BPEL4WS) [5] and Business Process Modelling Language (BPML) [2] but none has reached the required maturity. Either of the two, however, has the potential to become broadly accepted.

The development of WS-flow is a complicated procedure. Such a methodology, if it existed, would be accepted by developers only if it could be automated and if it could shorten development time and hide complexity. It has to also enable the creation of WS-flows with characteristics appropriate for the requirements of the environment. We elaborate on these requirements in section 2.

In section 3 of this paper we introduce a methodology for creation and execution of WS-based business processes. This generic procedure is based on a revised version of the process development lifecycle. The procedure contributes to the support of standardized inter-organizational interactions by exploiting the relationship between B2B coordination protocols and service composition. We demonstrate how the procedure can be mapped on the development and execution phases of WS-flows implemented using the existing process definition languages, namely BPEL4WS and BPML (section 4). The whole methodology aims at making process development faster and easier by means of automation, creating WS-flows supporting B2B standards, and above all

providing dynamic features and adaptable behaviour of the processes at run time.

2. Requirements on the design of WS-based processes

The traditional workflow technologies are not well suited for execution in a distributed service-oriented B2B environment [4], [12]. They are developed to support mainly intra-enterprise interactions, where no trust and organization boundaries are crossed. Apart from that the workflows do not assume services to be performing work on their behalf as participants or resources. As a result of the above traditional workflow processes do not obey the SOA principles; moreover they render only insufficient support for long-running transactions and their models do not provide for collaboration with partners. The workflow management system (WfMS) implementations strongly dependent on the platforms and programming languages, they are based on vendor specific process models, and in most of the cases lack flexibility and adaptability. Traditional workflow technology, and especially the process models and accompanying languages are therefore not appropriate to be used for the implementation of composite WSs. WSs expose applications in platform- and language-neutral manner. The technology defines unified communication protocols utilizing the Web and a common language for service interface description. These characteristics of WSs enable the inter-organisational communication across enterprise boundaries and over the Web but for very simple interactions among simple WSs. The inherent WSs features are not sufficient for composing WSs in complex business processes. Besides, the environment in which the WS-based processes have to operate is quite different from the one the traditional workflows operate in. There are additional requirements imposed on the design of business processes involving WSs. Those requirements are mainly determined by the highly distributed environment and the business rules, and include [10], [11]:

- ability to describe processes spanning software platforms and organizational boundaries
- ability to model collaborations, partners and their roles in complex interactions
- ability to recursively combine processes – requires that a process exposes a WS interface
- asynchronous service invocation to allow for performance, reliability and scalability
- flexibility and adaptability to changes in the business needs and environment
- availability of exception handling mechanism
- transactional support, especially for long-running transactions
- compensation of finished work on behalf of the process, without terminating the process itself

These requirements are imposed on the models for WS-flows and on their corresponding definition languages; the infrastructure for modelling and executing the processes is also influenced by these prerequisites.

In the next section we introduce a methodology for developing and executing business processes that helps to meet the above requirements.

3. A methodology for development and execution of WS-flows

In this section we introduce a simple methodology for design and execution of WS-flows.

The WS-based processes (WS-flows) are meaningful combinations of tasks for the solution of a business problem, which require work to be done by discrete Web services in a predefined order and according to rigorously stated business rules. The tasks (units of work) a WS-flow combines are represented by distinct elements of a process definition language, called activities. Different types of activities correspond to different types of tasks, e.g. simple and composite activities; data manipulation, exception handling, compensating activities etc.

Next we pay attention to the development life cycle of such processes. In traditional workflow the process development life cycle is defined in terms of only two phases: build time and run time. This division is very useful but not sufficient to define a procedure for creating WS-flows with the desired characteristics. It reflects the lack of effort towards standardizing process models development procedure in the field of traditional workflows. Here we briefly present a refined process life cycle with additional development phases (Figure 1). Those phases are:

- Process template modelling and assembly phase
- Process definition generation
- Compile time
- Pre-processing time
- Deployment
- Execution time
- Post-run time

Each phase addresses a different aspect of a process definition. If it is needed a step in the development of the process may be skipped; or a phase may be split into multiple sub-phases. The WS-flow life cycle is the

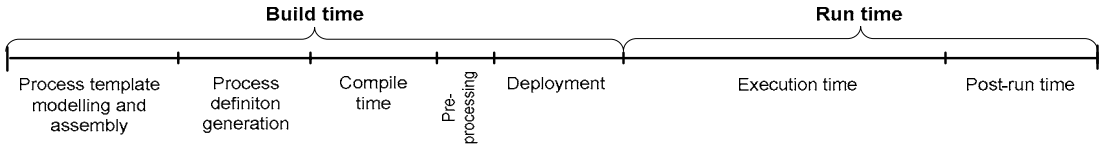

Figure 1. Life cycle of a WS-flow

framework on which the methodology we introduce next is based.

In the *process template modelling and assembly phase* (Figure 1) all standard and/or frequently used combinations of activities are grouped in templates. Templates can represent collections of activities implementing patterns, activities with special features, algorithms, place-holders for tasks defined by other standards, mappings and extensions. Apart from creating templates to be used in processes, whole non-executable process definitions can be assembled out of activity templates, and stored as process templates in a template repository. The purpose of using templates is to foster reusability and promote automation of WS-flow development. A process definition template is based on a process meta-model and represents abstract description of the WS-flow to be developed. This meta-model defines the structure of a WS-flow: it defines types of activities that are used to create the WSs composition, and data structures typical aspects such as nesting or inheritance. A language corresponding to the model must exist to represent the templates and the WS-flow definitions in computerized form. Provided that the process definition language is an extensible one, all additional extensions can also be modelled and implemented in this phase. As a result of this phase the developers obtain a reusable process definition.

The *process definition generation phase* is the one during which the process definition programmatic representation is generated. Having the process logic modelled during the previous stage we can generate the process definitions in any WSs-based process definition language. For this the process definition, i.e. the template we obtained in the first phase, must undergo one or more transformations. This phase can be split into several sub-phases, depending on the number of transformations that have to be performed. It highly depends on the degree of customization we wish to incorporate into the process definition, on the targeted adaptability features, and on the particular definition language we want our executable definition to be coded in. The principles of the model-driven architecture (MDA) [9] are extensively exploited during this phase, revealed by the fact that process definitions in multiple languages are based on a single, common process model for one thing, and the use of on-purpose transformation tools to generate language-specific process definitions from that model for another.

During the transformations the definition is enriched with additional details and data related to the business logic, to the abstract WS types used in the process, to the data structures used in the executable process. The tools needed to perform the transformations of the process definitions generally include code generators, compilers and other meta-programming techniques. All the transformations of the process definition render it closer and closer to the targeted executable process definition. Real flexibility of the WS-flows can be achieved if the commitment to a specific process definition language is deferred to the latest possible transformation; moreover, for providing dynamic features to the process at run time no bindings to any concrete WS instances should be specified.

The output of all the transformations is a process definition that is either ready for deployment or has to be *compiled* before deployment. In some cases a *pre-processing* might be needed. For example BPEL processes do not require compilation step, therefore it is skipped. The location of each WS invoked by the process has to be inserted into the definition, though; therefore the pre-processing step cannot be skipped because it will be used to provide the WSDL descriptions of the participating WSs, as well as for instance to generate the WSDL interface description of the process itself.

In principle, the result of the procedure up to the deployment phase is an executable WS-flow definition with minimum reference to WS instances. Those WS instances can be selected from a group of similar WSs and bound to the process instance during run time. This is dynamic invocation of WSs and provides for adaptability of the process. Dynamic features of a WS-flow are of great advantage. Additional degree of flexibility can be achieved by providing reflection support to the WS-flows. This is a feature missing in both traditional workflow and WS-based compositions.

Upon *deployment* a process definition is usually enriched with execution environment specific data and/or data related to the application execution. For instance, in order to deploy a BPEL process on a BPWS4j engine [6] it is required to provide the WSDL interface description of the process (being a WS itself) and the WSDL interfaces of all participating partners' WSs. Once deployed, a process can be executed.

At *run time* a process is instantiated from its schema (the definition) and executed. The process follows the execution order scheduled by the process control flow, and data is manipulated and exchanged between process and the invoked WSs as it is defined by the data flow. Depending on the process model and the definition language, a process instance might exhibit advanced dynamic features such as: finding the most appropriate WS for performing a task at run time and binding to it; use of complex choice policies based on quality of service (QoS) parameters; and even undertaking definition changes at run time. These features should be supported by the engine implementation.

It is always useful to gather information about the process execution during run time, which can later be used in the *post-run time* to analyse the process logic and change it accordingly.

This procedure aims at providing certain features to the WS-flows and has specific implications on the process model and the infrastructure. We summarize them next:

- *Automation* of the process definition development is supported by using activity and process templates, and a modelling tool. Additionally, the availability of code generation tools (i.e. meta-programs) and transformation tools to create the process executable form *shortens the development time.*
- Using predefined templates allows for *hiding the complexity* from the developers. Templates can be created for activities that consume and generate messages, perform special complex algorithms, etc., even templates for whole conversational patterns of which the developer does not need to know in detail. This however requires the availability of special template library or repository for storing them. *Separating the concerns* of process developers from the ones of template developers is an approach usually preferred when developing complex applications.
- *Flexibility* of the approach is achieved by postponing the choice of a language for the definition and thus providing the developer with the free choice of technology and process execution engine. By this process definition reuse and technology leverage is also promoted.
- *Adaptability* of the process definition is facilitated by deferring the binding to specific WSs to the latest possible point in time. This deferred binding to WSs instances has to be enabled by special model constructs and their corresponding language elements, and by the implementation of the WS-flow engine. The engine must be capable of executing process definitions with no given concrete WS instance. It has to provide a so-called "find and bind" mechanism for locating available WS instances of the same type (defined by the process schema) and binding to them during run time. Choosing the most appropriate WS instance is related to the concept of choice policy. Therefore it is necessary to have a policy description language in place, which should consider the quality of a service in terms of different criteria, as well as its availability and based on this to specify selection rules. The model in turn has to accommodate this mechanism and provide the corresponding constructs (activities) for it. This mechanism facilitates dynamic invocation of WSs at run time, which is one of the ways to adapt a process to the changing environment. Reflective support at run time is a feature that also promotes adaptability. This is a feature missing in the field of traditional workflow. It requires a reflective repository storing the meta-model of each process definition language and a WS-flow engine supporting reflective activities.

Creating WS-flows is only a step toward conducting multi-party business interactions. WS-flow definitions define only that part of such an interaction a single party is responsible for; other participants must implement their own parts of the interaction accordingly. The interaction among parties is usually described by an agreed-upon protocol, which has a significant influence on the implementations of each partners' internal business process. In the next section we extend the procedure for developing WS-flows by exploiting this fact.

3.1. Relationship between coordination protocols and WS-flows

In the previous section we introduced a general procedure for developing WS-flows, which aims at providing such processes with specific (and desired) features. Developing WS-flows and being able to execute them is only a small part of carrying out inter-enterprise interactions. Complex multi-party business

interactions are enabled by implementing complex sequences of operations among complex WSs in the correct order. This is a topic described by the term coordination [1], also known by the term choreography [10]; for clarity the term coordination will be used throughout this paper. Coordination is a term describing the message sequence among multiple partners, i.e. the public message exchange among WSs. The coordination protocols are specifications of the set of correct message exchanges, called conversations, between parties in an interaction. A coordination protocol assigns a role to each party in the interaction, too. Each role in a coordination protocol has its own role-specific view on the overall interaction. The party implementing a specific role has to receive and send all the messages specified by its view of the interaction and in the prescribed order. Therefore the party implementing a role has to produce and consume exactly these messages as specified by the role description. This fact reveals the influence coordination has on the implementation of complex WSs.

The WS-flows are one way to implement complex WSs. WS-flows are also denoted by the terms orchestration [10] and service composition [1]; these terms are used to denote executable business processes or the private implementation of a composite WS. In this sense, when implementing a WS using WS-flows, the service orchestration definition has to implement the role-specific view on the multi-party interactions.

Clearly, this relationship can be used to generate WS-flow definitions that comply with coordination standards [1], [11]. Based on the process model a developer can create templates implementing role-specific views defined by existing standards. These templates can be further enriched by business logic activities and/or whole business logic templates during the first phase of the process life cycle. This means that the composition schema must contain activities that consume and send messages as it is prescribed by the coordination and in the precise order [1], the implication being that the model defining the WS-flow schema must at least model activities for sending and receiving messages, and in some cases data structures to support message correlation. To allow for more flexibility of the approach, here one should be allowed to choose from among existing coordination protocols. Therefore very generic activities receiving and sending messages must be specified by the model and then later, using a transformation, be mapped on the specific constructs of the coordination protocols. The mapping can be done during the modelling phase or if we prefer to maintain greater flexibility, the commitment to a protocol can be made in phase two during the transformations of the WS-flow definition. One implication on the platform for executing WS-flows supporting coordination standards is that since it is not the process engine to perform conversation routing a *coordination controller* [1] must be included into the infrastructure. The conversation controller can either be a separate component of the platform or integrated in the process engine. It is basically responsible for performing mapping from the message format of the coordination protocol to the message format used by the engine and vice versa, for message routing, and for correlation of messages to the correct process instances.

Important for the success of such an approach is the availability of process definition transformation *tools* for any process definition language. This success can be magnified by the availability of a variety of coordination templates (role-specific) – residing either at the model repository or at a special-purpose standard-specific repository, as well as the availability of tools to transform those templates into the syntax of the different choreography protocols. Thus it will be possible to create quickly and easily reusable process definitions that can be transformed into any WS-flow language definition, and that additionally provide support for any coordination protocol.

To summarize, WS-Flow definitions exhibiting support for standard B2B protocols can be created by enhancing a role-specific conversation definition written according to an existing B2B interaction standard with the proper business logic. This can be achieved using conversation templates during the first phase of the methodology. The use of templates makes it easier for the process developers to concentrate on the proper definition of business logic. The templates hide complexity of the conversations behind activities that consume and produce messages. As a result from the first stage an abstract description of a process is created without any relation to any definition language, WS instances or execution environment. In the next sub-section we provide a short example that shows how a coordination protocol template can be used to create a WS-flow definition.

3.2. Implementing coordination protocol role using a WS-flow

In this section we elaborate on the relationship between the implementation of a composite WS and the coordination protocol it takes part in using a simple example. Consider the following coordination protocol that defines the message exchange sequence between four partners: a client, a converter, a bank and a

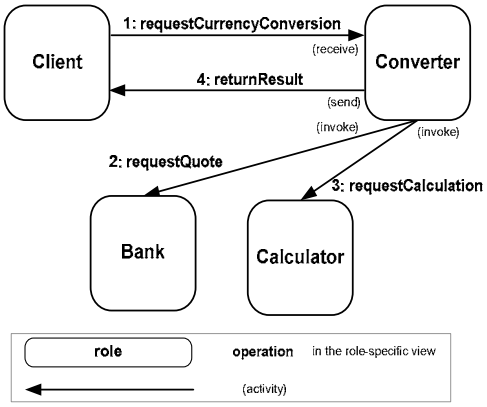

Figure 3. An example of a simple coordination protocol

calculator. The interactions among those parties are the following:

1. The *client* requests a cross-currency calculation from a party playing a converter role. The client sends two strings (identifiers for the two currencies) and a number, representing the amount in the original currency.
2. Upon receiving a request, the *converter* sends a message to a *bank*, which provides a service for cross-currency exchange rates, and gets the rate value.
3. The converter sends the exchange rate value and the amount in the original currency to a *calculator* service, and receives the result of this simple calculation.
4. The converter sends the result of the conversion to the client.

We call this very simple coordination protocol the "cross-currency calculation protocol". It is depicted in Figure 2.

The converter role in this protocol has a specific view on the whole coordination and it is shown using an activity diagram in Figure 3.

According to some simple rules [1] the operations described in Figure 2 are mapped to activities, i.e. messages the converter role has to accept or produce in Figure 3. The converter role specific view represents the converter's part in the overall interaction, i.e. its public process. The public process can easily be mapped to activities of the process model and a reusable template can be constructed for exactly this role and coded in the language corresponding to the model. In the modelling phase a developer can pick this template and add some business logic to generate a non-executable WS-flow definition for the converter

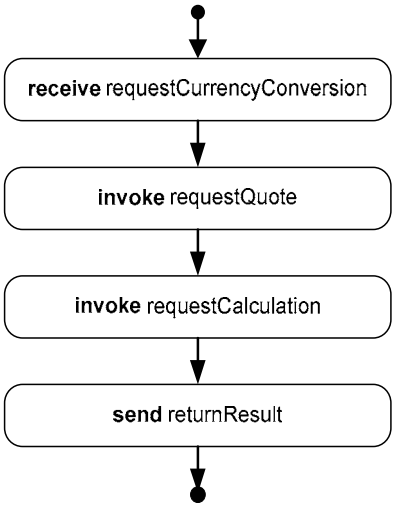

Figure 2. Activity diagram for the converter role interactions

role. This in turn can be considered as another template for the business process to be implemented by a party that will play the converter role, especially if its representation does not refer to any specific language.

Having the abstract definition of the WS-flow the procedure reaches its second phase – the process definition generation phase. During this phase additional details are inserted into the definition and as a result an executable process definition is generated from an abstract one. In the pre-processing phase or during compilation (or both) additional data can also be supplied. Depending on the targeted process definition language, during deployment additional data can be inserted into the process code. The WS-flow is then executed.

To prove the plausibility of this procedure we show how it can be used in the context of the existing WS-flow technologies by means of the simple example, presented in the next section.

4. Using the overall procedure to generate definitions in existing languages

In this section we show a simple example and explain how the procedure introduced earlier can be mapped on a development of a BPEL4WS definition and on a BPML process.

If we choose to implement the converter role of the "cross-currency calculation" coordination protocol in BPEL we can use the approach considered in the previous section; Figure 4 represents how the proposed methodology would look like in the context of BPEL.

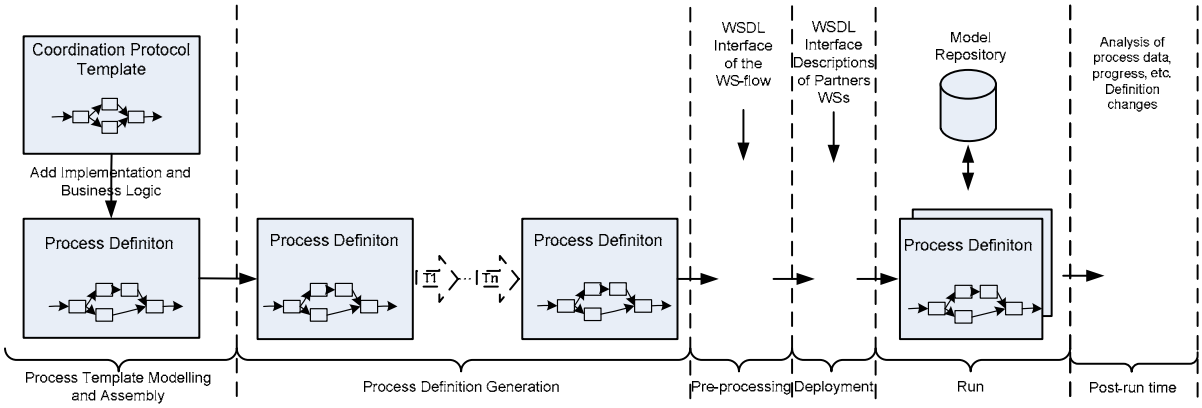

Figure 4. Development life cycle of a BPEL process reflecting the proposed methodology.

In the first phase business logic is added to the coordination protocol template for the converter role. Such logic includes tasks such as copying data from one variable to another, choice of conversion method based on the data provided by the client, selection of a bank if the client has explicitly specified such, and so on [7]. This concludes the job to be done in the modelling stage and we can proceed towards the definition generation, where the input is the process template (written in XML) and the result we wish to obtain is a process definition in the BPEL language. We have two alternatives in this case: we can either use a code generator and a meta-repository to generate an XML document with the BPEL syntax, or we can perform simple transformations on XML documents using XSLT. Having in mind the current version of the BPEL specification [5] the number of transformations can be very small (most probably up to two) in order to get an executable BPEL process. The required WSDL interface of the BPEL process can be provided during the pre-processing step. This document can either be written from scratch by the developer, or it can be generated based on the process definition itself. For the deployment of a BPEL process on a BPEL engine, e.g. BPWS4j [6], it is necessary to specify the WSDL descriptions of the process partners, in this case the WSs implementing the "bank" and the "calculator" role. After a BPEL process is deployed it can be executed. There is no reflection support specified and provided for BPEL processes and for any other WS-flows so far, but for reason of completeness a model repository is also depicted in Figure 4.

The procedure we considered in this paper can be used for creating executable WS-flow definitions and for generating their public processes/interactions. This can be done by performing code transformations on the role-specific coordination protocol templates. In the case of BPEL this can be done very easily even during the modelling phase, because the syntax the BPEL4WS specification [5] defines is the same for both executable processes and abstract BPEL processes. One can use the same transformation tools, but the end result does not have to be an executable process. This does not apply for the Business Process Modelling Language (BPML) [2], because it only deals with executable process definitions, but it is closely related to the Web Service Choreography Interface (WSCI) [3]. WSCI is an interface description language that prescribes how role-specific views in coordinated interactions are created. Different tools are needed to generate automatically the public interface of a BPML process.

In addition, WS-flows expose WSDL interfaces in order to be accessible over the Web; these interfaces can be generated, too, using some special-purpose tools during the process definition generation phase.

5. Related work

The approach for development and execution of WS-flows we introduced is closely related to the MDA [9]. The process definitions are generated as a result of transforming a WS-based process model into language specific process definitions.

Lots of attempts had been done to provide for automation of the development process and flexibility of workflows. For example, an approach similar to the one discussed here is represented in [11]; it, however, deals mainly with integrating traditional workflow processes with B2B interactions standards, such as Rosetta Net PIP, CBL and others. The authors provide a complete solution for automatically generating and using process and service templates that comply with

B2B standards. The solution provides for the integration coordination protocols with internal workflow processes, uses a Conversation Manager for conversation control, and specifies and uses a repository of B2B services and process templates. It aims at the easy and rapid adoption of B2B interaction standards for coordinating conversations among complex workflow processes. This approach is not meant for the service-oriented systems; it includes neither Web services nor any other kind of services as workflow participants.

The relationship between coordination protocols and process internal implementation and its implications are discussed in [1]; the same relationship is also considered in [10]. The discussion in [1] is mainly focused on the characteristics of an infrastructure for executing composite WSs that support coordination protocols. An overview of the architecture of an infrastructure for supporting such complex WSs is provided. The authors elaborate on the basic functions of a conversation controller and protocol handlers and their place in such an infrastructure for ensuring correctness and consistency of the interactions. How to model the correct set of conversations that a complex WS takes part in and the restrictions it places on the internal implementation of the WS are represented in terms of examples.

6. Conclusions

This paper presents a methodology for development and execution of WS-based processes, also called WS-flows. It is based on a detailed and revised process life cycle phases. During the build time phases of the life cycle a WS-flow definition is modelled and generated from templates. The methodology allows also the use of templates that comply with B2B interactions protocols; thus it exploits the inherent relationship between the definition of public interactions a process takes part in and the internal implementation of the process. Depending on the process model characteristics it should be possible to generate flexible and adjustable business processes supporting different coordination protocols. This procedure reaps all advantages of MDA; process definitions in multiple languages can be created based on a common process model. Applying the methodology could reduce manual work of developers and help them easily create WS-flows. The use of this procedure fosters process definition reuse and allows developers to take advantage of existing WS-flows technologies. The procedure can also be used to transform process definition written in one language into a definition in another, using the common model as basis for the conversion. Developers additionally benefit from the approach because the WS-flows can include support for existing standards for inter-organizational interactions without being experts in this field.

7. References

[1] Alonso, G., Casati, F., Kuno, H., Machiraju, V., *"Web Services. Concepts, Architectures and Applications"*, Springer-Verlag. Berlin Heidelberg New York, 2003.

[2] Arkin, A. et al., "Business Process Modeling Language", BPMI.org, 2002.

[3] Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi-Nagy, P., Trickovic, I., Zimek, S., "Web Service Choreography Interface v. 1.0 (WSCI)", BEA Systems, Intalio, SAP, Sun Microsystems, 2002.

[4] Curbera, F., Khalaf, R., Frank Leymann, Sanjiva Weerawarana, "Exception Handling in the BPEL4WS Language", *In Proceedings of the BPM2003*, 2003.

[5] Curbera, F., Goland, Y., Klein, J., Leyman, F., Roller, D., Thatte, S., Weerawarana, S., "*BusinessProcess Execution Language for Web Services (BPEL4WS) 1.0*", August 2002, http://www.ibm.com/developerworks/library/ws-bpel

[6] IBM AlphaWorks, "IBM Business Process Execution Language for Web Services Java™ Run Time (BPWS4j)", IBM, 2002, http://www.alphaworks.ibm.com/tech/bpws4j

[7] Karastoyanova, D., "Creation and Deployment of Web Services and Web Service Flows", A Tutorial, *In iiWAS2003, The 5th International Conference on Information Integration and Web-based Applications & Services*, Austrian Computer Society, September 2003.

[8] Karastoyanova, D., Buchmann, A., "Components, Middleware and Web Services", In *IADIS International Conference WWW/Internet 2003*, Volume II, IADIS Press, 2003, pp. 967-970.

[9] Kleppe, A., Warmer, J., Bast, J., "*MDA Explained. The Model Driven Architecture: Practice and Promise*", Addison-Wesley. 1st Edition. 2003.

[10] Peltz, Ch., "Web Services Orchestration and Choreography", *IEEE Computer*, October 2003, Volume 38, Number 10, pp. 46-52.

[11] Sayal, M., Casati, F., Dayal, U., Shan M.-Ch., "Integrating Workflow Management Systems with Business-to-Business Interaction Standards", HP Laboratories Palo Alto, 2001.

[12] Shapiro, R. "A Comparison of XPDL, BPML, and BPEL4WS", Cape Vision, May 2002, http://xml.coverpages.org/Shapiro-XPDL.pdf