Development Life Cycle of Web Service-based Business Processes. Enabling Dynamic Invocation of Web Services at Run Time

Dimka Karastoyanova and Alejandro Buchmann

Technische Universität Darmstadt,
Department of Computer Science
Wilhelminenstrasse 7,
64283 Darmstadt, Germany
[1]dimka@gkec.tu-darmstadt.de
[2]buchmann@informatik.tu-darmstadt.de

Abstract. Web service technology aims at application integration by providing stable service interfaces and standardized communication protocol. However, this is not yet a mature technology; it lacks certain features, among which ability to compose services in the most flexible way. We begin with a comparison of traditional workflow and the existing Web Services-based process technologies; the advantages of the emerging technologies and how they meet the new requirements imposed by both the business and Web services worlds are pointed out. We revise the process life cycle by including additional phases to the traditional division in only build time and run time. This fosters standardization, and allows for modeling adaptable business processes. We concentrate on the dynamic invocation of WSs from within a process instance and present a new way of finding, binding to and invoking WSs during process runtime. For this we introduce an additional run time sub-phase to accommodate the so-called "find and bind" mechanism, which involves policy-based selection of services and binding to them at run time. The implications of the "find and bind" mechanism on the process model and the implementation of the execution environment are also discussed.

Keywords. Web Services, WS-flows, WS-flows development life cycle, dynamic invocation of Web services.

1 Introduction

Web services are the newest technology for application integration. This technology's main advantage is facilitating the application integration across organizational boundaries and using the Web – a cheap, simple and ubiquitous communication medium. Service interface description and communication protocol specifications for Web services (WSs) already exist (WSDL [25], SOAP [23]), as

well as a specification for Web services registry (UDDI [4]). The technology is however still immature. It exhibits characteristics of conventional middleware but there are still some missing or not completely specified or implemented features [1], [18] such as reliable messaging, caching, conversational support, coordination and transactional support, compositions made up of Web services, and others, which would make Web services as important and reliable as the existing middleware services.

In this paper we focus on compositions of WSs, also known as Web Service Flows (WS-flows) [17]. There are quite successful attempts to define Web service compositions and some corresponding implementations. Even though we find certain similarity between the available WS composition specifications and the traditional workflow technology, the WS compositions do not yet support all needed features. The existing WS composition languages do not yet define distributed business processes [9], and do not provide *dynamic invocation of WSs* during runtime. Moreover, the existing specifications provide insufficient support for *flexibility* and *adaptability* of processes to the continuously changing business environment. To provide this support a strictly specified methodology for development and execution of WS-flows is necessary, in addition to a common process meta-model for WS-flows.

In section 2 we provide a brief comparison of traditional workflow technologies and the features of the emerging business process technologies for WSs. Based on this comparison we comment on the additional requirement imposed on the design of the WS-based processes determined by the characteristics of highly distributed environments.

The definition of a business process development life cycle and its distinct phases is not fully explored in the fields of traditional workflow and WS-based processes. We refine the process life cycle known from conventional workflow and so adapt it to processes involving WSs (section 3). Each of the phases accommodates particular approaches addressing different aspects of creating flexible WSs-based business processes. It is meaningful to present such a revised formulation of a process life cycle, for it provides useful directions for modeling and developing flexible WS-flows, and facilitates reusability of process definitions. The life cycle definition provides also clear guidance and framework for defining a methodology for creation and execution of WS-flows and motivates the creation of a common process model.

In section 4 we pay special attention to the additional “find and bind” run-time sub-phase that helps solving some problems of the existing systems, rooted in their lack of dynamic features during process execution. We conclude with a simple example of how the “find and bind” mechanism functions in practice in the context of a BPEL process.

2. Web services and traditional workflow

Web services are currently used to perform only very simple computations. For mission critical applications it is required to combine today’s simple WSs into complex ones and enable complex coordinated interactions among them; such

complex WSs would be more suitable for achieving complicated business goals. Therefore a common business process model and corresponding definition language specification are necessary. For the purposes of our further discussion in this respect, this section gives a brief overview of the existing WS-based composition technologies while comparing them to the traditional workflow approaches.

Usually, when composition of tasks is discussed one thinks of the existing workflow technologies. Workflow technology has matured in the last decade. It has been the subject of extensive standardization and is broadly accepted. The most prominent workflow standardization community is represented by the Workflow Management Coalition (WfMC). The WfMC has provided a reference model for Workflow Management Systems (WfMS) that allows for the interoperability between WfMSs developed by different vendors. Interoperability is assured by implementing standardized APIs exposed by the WfMSs and all auxiliary tools supporting the development and execution of workflows [14]. There is no standardized model for the process definition itself, and therefore all vendors are free to develop their own workflow models. However, the WfMC specifies the XML Process Definition Language (XPDL) that is used for exchange of process definitions across different WfMC-compliant workflow engine implementations [15], [23]. Apart from the WfMC's there are other models and approaches for defining workflows [12], [16].

In the field of WSs there are two competing industry-driven specifications for service compositions. These are the Business Process Execution Language for WSs (BPEL4WS, shortly BPEL) [9] and the Business Process Modeling Language (BPML) [3]. They specify languages for defining business processes that use WSs for performing certain tasks on behalf of the process, i.e. WS-flows [17]. Either of the two specifications has the potential of becoming a de facto standard.

When comparing the traditional workflow technology with the newly developed technologies involving the use of WSs, considerable similarities are recognized. Basically, the WS-flows technologies are following the traditional workflow technology terminology and general principles. For example, a business process defined by BPEL or BPML is described in terms of the same general *aspects* a traditional workflow description exhibits. Even though there is no terminologically established explicit distinction of the perspectives of a process definition, both BPEL4WS and BPML define control and data flows [20] (corresponding to behavioural and information perspectives), distinguish between simple and complex activities (reflecting super- and sub-workflows, i.e. functional aspect), and specify the WSs that are going to perform on behalf of a particular process activity (operational perspective [16]). There are however differences, which can be ascribed to the basic principles of the WSs paradigm; WS-flows do not describe a process in terms of its relationship to some organizational structure – WSs are meant to provide an organization-neutral, transparent access to services over the Web.

To the differences counts the fact that the existing WS-flow languages are block-structured [22], whereas the traditional workflow languages (XPDL, MOBILE [16]) are based on a directed-graph model. These are two groups of languages, based on *different process calculi* [10] and therefore they have distinct operational semantics. The differences in the semantics have important implications. Block-structured languages are more suitable for expressing much more complex control flows.

Moreover, they are able to provide support for distributed business processes in which exception handling can be interleaved with the business logic [13]. It is very important to realize that block-structured process definition languages are more suitable for enabling transactional support for long-running business interactions. In the case of *long-running transactions* compensating activities have to take a meaningful corrective action that allows the process execution to continue, rather than simply reversing the effect of completed activities and terminating the whole process. Even better, in this respect, is the approach BPEL4WS provides, which is a hybrid-solution combining language elements corresponding to both block-structured and graph-based models [10]. There are two other specifications closely related to BPEL4WS that add features to the capabilities the applied process calculus provides. These are WS-Coordination [6] and WS-Transaction [7]. They provide a coordination framework and pluggable transaction protocols, respectively, extending the capabilities of BPEL4WS represented explicitly by the concept of scopes.

Since WSs is a technology meant to support application integration across organizational boundaries it is relevant to consider both traditional workflow and WS-flows according to their *suitability for business-to-business* (B2B) *interactions*. In addition to the potential exception handling and transactional support capabilities WS-flows meet some additional requirements imposed by the B2B environment [21]. WSs enable inter-enterprise interactions in a standardized way (standardized communication protocols and formats). On the one hand WS-flows take advantage of this by involving WSs to perform on behalf of a process, as opposed to conventional workflow, where WSs are not among the set of applications and resources responsible for performing tasks. On the other hand, some WS-flow languages carry this further by exposing the business processes as WSs (e.g. BPEL4WS), thus ensuring the reuse of functionality among enterprises, via a uniform interface and using a standardized communication protocol.

Certainly, WS-flows have advantages over conventional workflow processes in a distributed business environment but there is a lot more to be done for them to qualify for mission critical applications. There is no specification that states completely how dynamic invocation of services (from within a business process instance) is to be performed. Processes are currently modeled as collections of activities bound to specific WS instance (or at least to their abstract description). This proves restrictive for providing flexible and adaptable processes based on WSs. Therefore, a meta-model for defining WS-flows is needed. The existence of such a model would enable the creation of abstract process definitions that would allow postponing the binding of the process definition to particular WSs instances to a later phase of the process life cycle. To relate to the exact point in time in which the process definition is bound to specific WSs, special attention has to be paid to the life cycle phases of a WS-flow.

In the field of traditional workflow the *life cycle* is divided into two stages: build time and run time (Figure 1) [14], [16], [20]. This division is important but it is not sufficient enough for the purposes of developing flexible WS-flows; a more detailed specification of a process life cycle is needed. Therefore in the next section we revise the phases of the process life cycle; it simplifies and clarifies the development of adaptable processes. It gives also guidance about what kind of process model is needed and what transformations its computerized representation must undergo before becoming an executable process definition. Such an explicit definition of distinct

process life cycle phases fosters process definition reusability and allows for clear separation of concerns of process and system developers.

3 Revision of the WS-Flow development life cycle

The WfMC's standardizing effort is mainly focused on standardization for interoperability among WfMSs; hence it did not put any restrictions on the process model the WfMSs vendors use, and did not specify any standard procedure to guide the development of a process. As a result it specified only the distinction between build time and runtime (Figure 1).

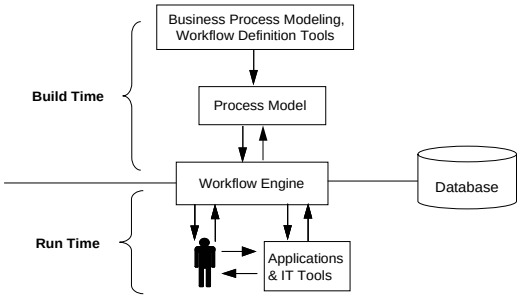

Figure 1 Development life cycle of a traditional workflow.

This is quite a general view which should be explored and refined further. The detailed statement of a process development life cycle is not a new way of considering application development, as process development certainly is. Typical phases in the life cycle of an application include production time, compile time, link time, load time, run time, and post-runtime [11] in a continuum. The application phases depend on the application programming model used. For instance, the above mentioned division is not relevant for highly available distributed systems, because they are almost always at runtime; but for components being replaced in the system, those life cycle phases are relevant. In the context of typical C++ applications slightly different life cycle phases are distinguished; these are source time, pre-processing time, compile time, link time, load time, and run time [8]. Component generation, as another example, might require two more phases: code generation and code assembly.

3.1 Model considerations

Given our goal is to create adjustable business processes and the corresponding process management systems to execute those processes, it is necessary to develop a common *process model*. Such a model has to define a process without precisely specifying the WSs to be invoked; in other words, no locations of WSs should be incorporated into the process definition. For an even more flexible, and of course more complex solution, the model might also exclude any explicit statement of the

abstract definitions of a WS such as portTypes, operation names etc. In order to get an executable definition of the WS-flow the abstract process definition based on the model should undergo different transformations in a predefined order, specified by the process life cycle. During the various transformations the process definition will be enriched with the necessary data, and in some cases meta-data might be required depending on the process model used; for instance, the moment a WS-flow is executed, it might be necessary to supply it with meta-data related to the WSs it invokes, or meta-data might also be required during transformations in the phases prior to run-time. It is necessary to relate the model development, as well as the process definition transformations to a standardized methodology supporting process development. Such a methodology should be guided by a precisely specified process life cycle. Such a procedure does not yet exist; moreover process development life cycle is a topic not fully explored in the context of both traditional workflow and WS-flows. Therefore in the following section we introduce our view on the development life cycle for a WS-Flow and its phases.

3.2 WS-flow life cycle

To accommodate the above mentioned considerations we introduce a detailed and refined WS-flow life cycle (Figure 2).

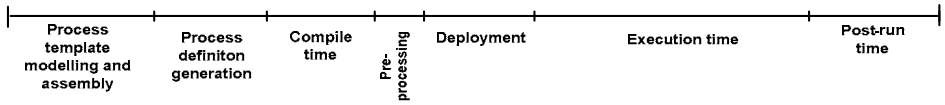

Figure 2 WS-flow life cycle phases.

It includes the following phases:

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

Depending on the application scenario these phases may be further split into sub-phases; phases can also be skipped.

The process *template modeling* and *assembly* phase is the one in which the process is modeled and a process definition template is created. The *WS-flow template* is a collection of activities, defining the process abstractly. Depending on the process model, the activities comprising the process template may themselves be templates, e.g. representing frequently used complex activities, design patterns, decision activities, algorithms, business rules, place holders, etc. To reflect this fact, this life cycle phase might be split into two sub-phases, for example: *template generation* - for creating templates for design patterns, special business logic activities, and others,

and *template assembly* – combining those templates into process definitions. The process definition resulting at the end of this phase should exclude any specification of the exact WSs instances, as well as any commitment to a process definition language. In other words, the output of the first life cycle phase is a non-executable abstract process definition. It is created on the basis of a common meta-model and common model constructs represented by the elements of a corresponding definition language. In order to obtain an executable computerized representation of the process additional details have to be inserted. This is done during the next step.

During the *process definition generation phase* the process definition created in the previous phase is transformed into an executable process definition. The process definition might undergo several transformations, and during each transformation it is enriched with further details and data related to the executable process. Again, this phase may also be split into several separate sub-phases. Each of these sub-phases corresponds to a particular transformation imposed on the process definition. Those sub-phases involve the use of meta-programs performing transformations based on a meta-model, e.g. code generators, compilers and so on. As a result a much more detailed process description is obtained. It might also be a description of a WS-flow in one of the existing languages; this in turn might result in skipping some of the succeeding phases, due to the characteristics of those existing technologies. This phase could also be used to translate a process definition written in one language into another one. Enabling such conversion fosters the reuse of process definitions on different workflow engines based on the process model, rather than on language mappings.

Depending on the implementation approach the process definition can be enhanced by meta-programs during *compile time*, too; but this is not to be considered an obligatory step. For instance, if a process definition in BPEL has to be created and executed, the compile phase would not be relevant to the process definition itself. However, this step may include compilation of parts of the process definition.

In some application scenarios a *pre-processing* step could also be required. Pre-processing involves enriching the process definition with additional data, as well as in some cases providing interface descriptions of WSs participating in the process.

Usually upon *deployment* the process definition is enhanced with execution environment-specific information. In some cases during that phase the system is provided with auxiliary information related to the application execution; take for example component models such as the J2EE one. Such a declarative specification of properties can also be useful for WS-flows development. Depending on the process model and the implementation approach different data can be appended to the process definition during the deployment phase. For instance, the BPEL4WS specification requires that the WSDL documents of all participating WSs are provided, as well as the WSDL interface of the process itself. Moreover, additional code is generated within the process interface definition that states the binding information necessary to access the process as a WS – port locations, access mechanism.

After being deployed a process can be executed. During *run time* a process instance is created and runs according to the process schema. In traditional workflow all applications, resources and human participants are stated prior to the execution, or as it is the case with some more flexible systems, participants and resources can be

picked up dynamically from a pool of available ones. The existing WS-flows specifications do not yet provide a fully dynamic invocation of WS for performing a task; either the exact location of the WS is specified together with all the binding information (BPEL4WS), or at least the abstract description of WSs is stated. These cases do not provide for fully adaptable and flexible WS-flows. The latter case is difficult to deal with having in mind the state of the WSs technology. It is not yet possible to search for WSs according to the functionality they provide, due to the lack of sufficient semantic description approaches for WSs; therefore it is not yet feasible to postpone the binding to an abstract WS (i.e. its functionality, portType) up until run time and that is why it has to be done during some of the stages before the execution. The former case, in which the binding to a concrete WS location is involved, is more restrictive with respect to adaptability features but can easier be dealt with. Under certain assumptions (reflected by the process model as well) it is possible to choose an instance of a WS that can perform work on behalf of a process activity, based on its abstract description. In general, to do this a search for WS instance has to be done before each single invocation of a WS, and when found its binding information (WS instance location) has to be incorporated into the activity definition, in order to perform the service invocation itself. This is a recurrent sub-phase that we call "*find and bind*" phase. We draw the attention to this particular life cycle phase in the next section. Note that this phase is a part of the runtime of a WS-flow and in principle it has to take place immediately before each WS invocation.

As a final phase in the WS-flow life cycle we define the *post-runtime* phase, which allows making changes in the process definition, depending on the progress and results of the execution of a process instance, and on changes in the business requirements. Such changes are classified as static (process) configuration [11].

Having stated clearly the framework for WS-flow development and execution allows us to concentrate on the different approaches and tools needed for addressing each step in the life cycle. In the next section we discuss an approach for enabling dynamic invocation of WSs during the run time of a business process.

4 Dynamic choice and invocation of a WS. Find and Bind sub-phase

In this section we consider in more detail the "*find and bind*" sub-phase of the process run time phase. We explain how it accommodates and enables dynamic invocation of WSs by WS-flows, and what the advantages and disadvantages of this approach are. We comment on the implications of this sub-phase on the process model and the infrastructure implementation.

We include a "find and bind" sub-phase within process run time to enable dynamic invocation of WSs. Note that in this paper we refer to *dynamic invocation* as the combined action of selecting a particular instance of a WS and subsequently invoking it during the run time phase of the process life cycle. These are in essence the actions that have to be performed during each "find and bind" phase before each WS invocation (see Figure 3). The selection of a WS instance should be based on various criteria such as: compliance to a WS abstract description (WSDL abstract

description), availability at the moment of process execution, cost, performance and other quality of service (QoS) characteristics. This approach has several implications on both the process model and the infrastructure for modeling and execution of the WS-flows. Moreover it is related to other complex issues such as the ability to describe QoS characteristics of WSs, selection policy description and the respective description language, and WSs discovery mechanisms.

The *modeling* implications are in the fact that the process model has to support generic activities that perform the search and selection of a WS on the one hand, and on the other invoke a WS (Figure 3).

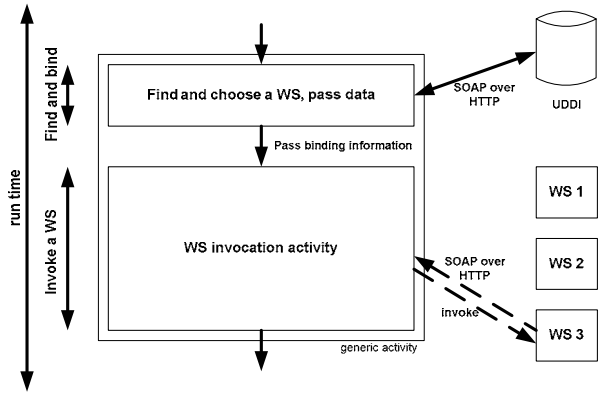

Figure 3 WS invocation activity extended with a "find and bind" element

Process definitions based on such a model should not include any reference to concrete WS instances, meaning that the definitions should be describing a WS-flow only on the basis of abstract information about the participating WSs. The existing specifications of WS-flow languages (BPEL, BPML) involve WSs invocation from within a process instance, but they do not enable dynamic invocation in the meaning of this term we defined above. However, it is possible to extend those specifications with such activities/constructs that either search for and invoke a WS, or only search for an appropriate WS and pass the necessary information to the already available activities that perform service invocation. We provide a simple example of a possible extension in the context of BPEL in section 4.1.

As regards the *implementation* characteristics, a suitable mechanism is needed to perform the look up of Web service instances and selection of a single WS according to (probably parameterized) criteria specifications, and then bring the information to the activities invoking the services. Consider that it might be necessary to keep this mechanism transparent for the users in order to hide the complexity of the whole approach. The basic idea behind the find and bind approach is similar to the one implemented by the Web Service Invocation Framework (WSIF). WSIF provides a mechanism to access WSs based on their abstract description (WSDL portTypes and operations), i.e. independent of their binding information (access mechanism and protocol) and port addresses. The goal of this client-side framework is however different; it aims at providing the WSs clients with the possibility to invoke services

using different invocation mechanisms and communication protocols (SOAP, java calls etc.) depending on their needs [2]. Following the idea of WSIF to use only the abstract descriptions of WSs and providing only this description to the process (without any detail on the concrete WS instances), a mechanism for finding all WSs implementing a given WSDL abstract description in a UDDI registry [4] can be developed (Figure 3). Having found all these services a single WS instance can be selected based on the given criteria. Then, after already having parsed the concrete definition of the selected WS, and having found the port at which it is available, we can pass this information to the activity dealing with the invocation of this particular service. In practice this can be done in two ways: the engine implementation could use variables and store the information there, or process variable can be used instead for the same purpose.

In the general case the finding and binding must be performed before each process activity that invokes a WS is started. However this might again have implications on both the process model and the execution environment (Figure 3). As a simple example, consider the idea of incorporating such an approach into a BPEL4WS process. Having in mind the specification, an extension to the language schema is required to accommodate the look-up for the most appropriate WS and the selection procedure, either for each WS invocation activity (invoke, reply, receive) or in some generic way. This influences inevitably the implementation of the process engine as well.

Introducing this additional repeating sub-stage during the run time phase has the *advantage* of allowing the process to select the most appropriate service at run time. Thus at least the availability of the invoked WS can be ensured, because the service's appropriateness depends on various criteria, including its availability. Each time a WS is not available for some reason, the find and bind approach can ensure that another WS implementing the same abstract WSDL description is used; compare this to the case in which the service location (port) is determined upon deployment – no other service can be used instead, and the process description has to be changed and redeployed. This provides a WS-based process with additional flexibility and increases the degree of availability of the process as a whole. A model that ensures this kind of flexibility at run time can also be considered an adaptable one; even though it is a quite restricted representation of adaptability, the WS-flow can adjust to changes in the environment, in this case exhibited by the changing QoS characteristics of a WS and more importantly its availability. To the flaws of the approach counts mainly the one additional call to the UDDI registry (SOAP over HTTP) executed before each WS invocation. This is a *disadvantage* that results in performance loss of the overall process. The realization of the approach is complicated by the lack of specifications related to QoS issues in the WSs world; therefore it is difficult to provide feasible selection policies today; and even if they existed, it would take computational time for the evaluation of the WS appropriateness.

Even though in this generic approach it is required to perform the "find and bind" phase before each WS invocation, in practice a more efficient implementation solution can be provided. The solution can be optimized with respect to performance by performing search for and binding to a WS only when it is needed. Usually in a B2B environment business partners know each other's processes and services; therefore they do not need to check the QoS characteristics of a partner's service each

time it is used. This means that a process should find and bind to services it invokes ones and perform subsequent search and selection only if a drastic change in the service's QoS characteristics happens. The trade-off between dynamic features at run-time and performance of the process depends on the implementation specifics. The success of the approach depends also on how the system is informed about the changes in the environment and the way the changes are described.

4.1 Example

To make the approach clearer we show an example. Consider a simple WS-flow implemented in the BPEL4WS language; the process implements a simple currency converter, and invokes two WSs with simple functionalities provided by partners – a service providing cross-currency rate quotes, and a service that does the calculation [17]. The most important steps in the process control flow are shown in Figure 4. A client sends a message to the WS-flow that is exposed as a WS; upon receiving the message the WS-flow invokes a partner WS that generates and returns a cross-currency rate quote; the WS-flow then sends the received cross-currency rate value to a simple converter WS (performing a simple multiplication) and gets as a result a value that it subsequently sends to the customer. The figure represents the WS-flow with the find and bind functionality already inserted as additional activities.

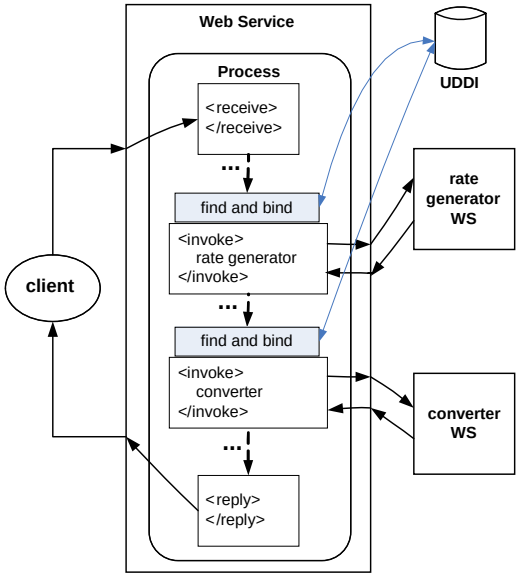

Figure 4 Currency converter WS-flow – an example

An example of a BPEL activity that invokes a WS can be seen in the next code listing; the code presents a BPEL process definition (details are omitted due to space limitations). The search for and binding to a WS is mapped to a language element, which can be appended to the BPEL4WS schema.

```
<process name="MyConvertCurrencyBP">
<!-- additional details -->
…
<!-- find and bind -->
   <find>
      <bpws-uddi:find_businessService()/>
      <bpws-uddi:get_bindingTemplate()/>
      <bpws-uddi apply_policy()/>
      <variable name="instanceURL">
          <wsdl:message name="URL-Return">
             <part name="URL-String" type="xsd:String"/>
          </wsdl:message>
      <variable/>
   </find>
<!-- invoke Converter -->
   <invoke partner="converter"
      portType="nsws:CurrencyConvService"
      operation="usd2eur"
      inputContainer="Currency_and_Rate"
      outputContainer="result"
      name="ConversionRequest"/>
<!-- additional details -->
…
</process>
```

The functions used to query the UDDI registry (qualified by the bpws-uddi namespace) can also be defined in an appendix to the BPEL specification. Of course this is a sample definition of those functions, the syntax can be different; moreover they can also be provided by the process engine and performed before any service invocation activity, and therefore the process developer would not need to care about coding but would lose the flexibility of specifying choice policies. As for the location of the chosen service, there are some alternatives for passing the information to the invocation activities. We can declare additional process variables in the BPEL process to store the WS endpoint address. It is also possible to make the engine care about the port address. Whatever alternative is chosen for the implementation, it is very important to allow the WS-flow to choose from a set of services providing the same functionality at run time.

5 Conclusions

In the developing field of WS-based business processes there is a need for a precise methodology for their development and execution. Such a methodology is closely related to and guided by the process life cycle. The development life cycle of business processes is an issue not completely explored in the field of traditional workflows. In this paper we refine the traditional view on the division of the life cycle phases and adapt it to the WS-based processes. The detailed description of the life cycle phases is meant to guide the creation and execution of WS-flows with desired features. Among those features are flexibility of the WS-flows and adaptability to changes in the environment. One way to provide those features is to enable dynamic

selection of WS instances and their subsequent invocation during the execution time of a process. For this, we introduced an additional sub-phase in the run time phase of the life cycle. It is named the "find and bind" phase, and as the name implies it accommodates a mechanism for finding the most appropriate WS instance for performing on behalf of the process and binding to it. The "find and bind" mechanism is related to the concepts of QoS characteristics and selection policies. This approach aims on the one hand at providing dynamic features to the WS-flows but on the other hand it requires that a common process meta-model be developed and extensions to the existing technologies and their implementations be made. The success of the "find and bind" approach is largely dependent on its influence on the overall WS-flow - it should be performed only when it is needed. This issue, however, is related to how the system is informed about changes in the environment and how it reacts on these changes.

References:

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V., "*Web Services. Concepts, Architectures and Applications"*, Springer-Verlag. Berlin Heidelberg New York, 2003.
2. Apache <Web Services/> Project, "*WSIF – Introduction*", 2002. http://ws.apache.org/wsif/
3. Arkin, A. et al., "Business Process Modeling Language", BPMI.org, 2002.
4. Bellwood, T., Clément, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y.L., Januszewski, K., Lee, S., McKee, B., Munter, J., von Riegen, C., "*UDDI Version 3.0*", IBM, HP, Intel, Microsoft, Oracle, SAP. 2002. http://uddi.org/pubs/uddi_v3.htm
5. Brittenham, P. Cubrera, F., Ehnevuske, D., Graham, S., "*Understanding WSDL in a UDDI registry*", IBM, 2001.
6. Cabrera, F. et al., "*Web Services Coordination*" (WS-Coordination), 2002. http://www.ibm.com/developerworks/library/ws-coor/
7. Cabrera, F. et al., "*Web Services Transaction*" (WS-Transaction), 2002. http://www.ibm.com/developerworks/library/ws-transpec/
8. Coplien, J. O., 1998. "*Multi-Paradigm Design for C++*", Addison-Wesley, Reading, MA, USA. (as cited in [Czarnecky, 2002])
9. Curbera, F., Goland, Y., Klein, J., Leyman, F., Roller, D., Thatte, S., Weerawarana, S., "*Business Process Execution Language for Web Services (BPEL4WS) 1.0*", August 2002, http://www.ibm.com/developerworks/library/ws-bpel
10. Curbera, F., Khalaf, R., Leymann, F., Weerawarana, S., "Exception Handling in the BPEL4WS Language", *In Proceedings of the BPM2003*, 2003.
11. Czarnecki, K., Eisenecker, U., "*Generative Programming: methods, tools, and applications*", Addison-Wesley. 2nd edition, 2002.
12. Dayal, U., Hsu, M., Ladin, M., "Business Process Coordination: State of the art, trends, and open issues" *In Proceedings of VLDB 2001*, 2001.
13. ebPML.org, "*XPDL*", 2001. http://www.ebpml.org/xpdl.htm
14. Hollingsworth, D., "*The Workflow Reference Model*", Document Number TC00-1003. The Workflow Management Coalition, 1995. www.wfmc.org
15. Hollingsworth, D., "*Events*", A White Paper, The Workflow Management Coalition. 1999. www.wfmc.org

16. Jablonski, S., Bussler, C., "*Workflow Management. Modelling Concepts, Architecture and Implementation*", International Thomson Computer Press, London, 1996.
17. Karastoyanova, D., "Creation and Deployment of Web Services and Web Service Flows", A Tutorial, *In Proceedings of iiWAS2003*, Austrian Computer Society, September 2003.
18. Karastoyanova, D., Buchmann, A., "Components, Middleware and Web Services", *In Proceedings of IADIS International Conference WWW/Internet 2003*, Volume II, IADIS Press, 2003.
19. Leymann, F., Roller, D., "*A quick overview of BPEL4WS*", IBM Developer Works, 2002. http://www-106.ibm.com/developerworks/
20. Leymann, F., Roller, D., "*Production Workflow. Concepts and Techniques.*", Prentice Hall Inc., 2000.
21. Peltz, Ch., "*Web Services Orchestration and Choreography*", IEEE Computer, October 2003, Volume 38, Number 10, pp. 46-52.
22. Shapiro, R. "*A Comparison of XPDL, BPML, and BPEL4WS*", Cape Vision, May 2002, http://xml.coverpages.org/Shapiro-XPDL.pdf
23. Workflow Management Coalition, "*Workflow Process Definition Interface – XML Process Definition Language*", Document Number WFMC-TC-1025, Version 0.03, 2001. www.wfmc.org
24. World Wide Web Consortium (W3C), "*SOAP Version 1.2*", W3C Recommendation, 2003. http://www.w3.org/TR/soap12-part0/
25. W3C, "*Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*", W3C Working Draft, 2003. http://www.w3.org/TR/wsdl20