A Platform-Independent Mechanism for Deployment of Business Processes Using Abstract Services

Adrian Mos, Thierry Jacquin
Xerox Research Centre Europe
Grenoble, France
adrian.mos@xrce.xerox.com, thierry.jacquin@xrce.xerox.com

Abstract— This paper deals with the problem of efficiently deploying business processes in service oriented environments with the purpose of enhancing operational agility and improving governance. Today’s solutions rely on early coupling between the business process layer and the underlying service oriented architecture layer and this constrains evolution in either of the two layers to the ability of the other layer to adapt. The presented approach leverages abstract architectural mappings that indirectly connect business process activities with service-based assets. It consists of a deployment mechanism that is capable of taking these mappings and generating fully executable artifacts for a variety of runtime platforms. The deployment mechanism is encapsulated in a self-contained entity that is instantiated and executed each time a business process needs to be deployed. This entity contains all the necessary data for the deployment, while ensuring complete accountability of operations. This approach promotes late binding of processes to the underlying SOA and it has the potential to significantly increase the speed of implementing changes in business operations while ensuring long-term sustainability. In addition it promotes independent evolutions of processes and services while encouraging incremental convergence of the SOA towards the needs of the business domain.

Index Terms— BPM, SOA, deployment, governance, agility, BPMS

I. INTRODUCTION

Managing business processes from their creation to their execution, has typically been the responsibility of Business Process Management Suites (BPMS). BPMS typically use a generic business process modelling language, such as Business Process Model and Notation (BPMN) [1] for defining business processes (BPs). BPMN models often lack the information that is required to connect process activities to executable services in the enterprise portfolio. This problem is sometimes addressed by using activity annotations that indicate the type of concept being used, or even the services to connect to. In addition, process template libraries where these connections are implemented provide ready-to-use models that designers can instantiate. The problem in these cases is that the capacity to quickly adapt to business changes is restricted by the template library or the availability of matching executable services in the IT portfolio. When the required changes fall outside the templates or when the executable services do not match well, the changes become very costly. In addition, the changes that affect several process models need to be replicated for each of these processes.

The approach presented in this paper aims to build on existing solutions while improving the capacity to support changes in BPs. The approach entails the creation of additional separation layers in the typical deployment structure of BPMS.

Deployment is an essential part in the lifecycle of the BPs and it is usually done using mechanisms specific to individual BPMS. In our context, deployment of BPs assumes only the existence of abstract mappings between business process activities and abstract services and does not mandate any other BPMS specific capabilities. In fact, our approach preserves compatibility with a wide range of BPMS and indeed any service-oriented execution platforms that users might employ. These abstract mappings are simple architectural entities (composite structures) that connect business process activity types to “idealized” services, or abstract services. These are meant to have perfect granularity and interfaces to ideally correspond to the business domain in question. These abstract mappings will eventually need to be bound to the real services. But this separation mechanism together with the deployment approach presented has the potential to significantly improve the adaptability of BPs to the business changes.

Our contribution focuses on the deployment mechanism that takes BPs with abstract mappings as input and creates the required artefacts to be executed in enterprise service oriented architecture (SOA). More precisely, the proposed mechanism leverages the architectural mappings associated to BP activities and assembles them into complete business process mappings. These elements bridge the business processes to the SOA assets in an abstract manner, as described later in the document. The mechanism employs one deployment package per BP which performs the required operations needed to eventually create the execution artefacts appropriate for the chosen execution platform. The deployment process itself is packaged in a generic architectural composition that represents a template to be instantiated differently for each business process to be deployed and for each platform that is targeted. This guarantees generality and reuse across platforms and also helps with the archival and management of deployment operations ensuring potential roll-back when needed.

The paper is organized as follows. Section 2 gives a high-level overview of our approach and introduces the main concepts that are used in the description of the mechanisms.
The mapping composites and the abstract services are introduced in Section 3 together with a simple scenario that illustrates their application in a typical environment. The deployment process that makes use of these elements to make business process descriptions operational is described in Section 4. Section 5 briefly discusses related work by explaining differentiating elements with other approaches before concluding in Section 6.

II. SUMMARY OF THE APPROACH

Our starting assumption is that the activities used in the input BPs have architectural mappings defined and associated. Although the engineering approach for their creation is not in the scope of this paper, an example is presented in Section 3, which illustrates how this can be achieved. Each activity is associated to an abstract architectural description that uses composition to group together various IT assets, namely services in the enterprise SOA. For human assets, particular types of services could be used as proxies in order to leverage a consistent architectural representation. For this reason in this paper we do not differentiate between human and non-human activities in business processes.

This mapping composition represents the concept's grounding into the IT platform and is the basis for mapping the business processes onto the platform. The services that are used in these compositions are not real services from a particular enterprise repository; rather they represent Abstract Services (AS) that correspond to an 'idealized' SOA view. Later in the deployment process, these abstract services need to be connected (bound) to the real services in the SOA thus enabling the actual grounding of the BPs to take place.

Since each business process is constructed using activities that are mapped to technical abstractions, its overall technical representation corresponds to a valid abstract composition of its IT mappings. A BP-level abstract composition is therefore obtained by grouping individual activity compositions and adding the required 'glue' needed to execute the business logic of the process. Through hierarchical composition, from activity maps to business process maps, consistency is established between the business elements and the IT assets. The deployment process takes the BP abstract composition and creates the corresponding technical artefacts that are appropriate for the chosen execution platform and the target SOA. This involves connecting the AS to the real services that are available, either directly or through adaptations, if required. This connection, or binding, is performed by a specialized human role that is presented with the appropriate context, the list of AS to map as well as a means to connect them to the enterprise SOA.

By providing the deployment package for BPs as an architectural template itself, we promote a mechanism that instantiates the package differently for each BP to be deployed. In effect, this approach entails having a deployment system instance created and archived for each deployment operation. When operators choose to deploy a BP (or indeed when BP designers deem a BP to be ready for deployment), a deployment package is created for this BP, containing all the required data and behaviour necessary to generate the BP composition, bind it to the appropriate execution platform, and make the necessary connections for various governance operations. This deployment package instance can then be saved and archived for later use, in case it needs to be re-enacted in the future. Since the deployment package instance is self-contained with respect to the data it requires, any execution of its operations is guaranteed to be independent from potential evolutions of the mappings to the underlying SOA. Long after a deployment has been successfully performed, in a completely new context, an old deployment that functioned correctly can be repeated at any time. Of course if operators want to deploy a BP in a new context, they would simply have to generate a new deployment process instance which will automatically take into account any changes in the business process activities and their mappings.

Although our proposal is not dependent on a particular technology, we have chosen a technological standard that is appropriate for its implementation and which further helps in clarifying the various parts of the contribution. This standard is Service Component Architecture (SCA) [2], an established component-based framework for service-oriented applications. It is important to note that SCA is our technology of choice for implementing the deployment package composition, as it provides all the required support for its functionality. However, for the mapping of business processes to the SOA assets, SCA is just a possible choice, which we believe is appropriate.

III. OVERVIEW OF SERVICE COMPONENT ARCHITECTURE

Service Component Architecture is an OASIS standard [2] for describing and specifying application architecture in service-oriented environments. The specification's main elements are the Composite and the Component. Components are atomic units of specification (black boxes) that have an implementation type (e.g. a language such as BPEL [3] or Java) as well as dependencies, configuration parameters and exposed services (i.e. access means for the outside world). The Composite may contain components connected to each other and, as any other component, it can offer services and require dependencies. In fact, composites can be used as components in other composites, thus realizing a fractal-like hierarchy of functionality. It is important to note that SCA promotes two types of reuse: logic reuse in the form of components that are instantiated in the context of their surrounding composite; and instance reuse in the form of services that are connected to dependencies (i.e. external functionality that is required for tasks). Services offered by components can become 'external' service if they are consumed as dependencies, so a component could be used either as a service provider or as an 'instantiable' brick in various contexts. When used as a service, it is assumed to run continuously in its environment and preserves its data regardless of its usage context. When used within a composite, as a reusable brick, it is instantiated with the composite and its data is therefore not shared with other execution contexts.

In SCA, dependencies are expressed as References that components and composites expose. They need to be bound to Services, which are provided either by other SCA
composites/components or indeed by any service provider (such as an Enterprise Service Bus).

An important part of SCA is the separation between logic/architecture and technological details. In fact, technological aspects of service invocations, component execution and various bindings are expressed as orthogonal concerns and connected to the architectural artefacts through configuration properties. Thus, architecture can be kept at a functional level for the purpose of architecture design and collaboration, and when required, it can be deployed to an SCA runtime after specifying 'how' various elements should be executed. Moreover, since the logical specification is separated from the technical details, one can easily vary execution properties while preserving the same architecture. A typical example is choosing between various binding types (i.e. technologies for invoking services) for dependencies as well as for component services. One can write a component and offer its functionality over various network protocols, or one can use an external service through a variety of protocols; the SCA runtime handles the conversions.

The unit of deployment in SCA is the composite. When deploying SCA applications to SCA runtimes (such as Apache Tuscany [4], or commercial ones from IBM, Oracle and other providers), composite files (.composite) are provided following standard conventions.

In this paper, when mapping BPs to executable assets, we propose SCA primarily as an architectural specification standard as it offers a useful approach to hierarchical composition and reuse in the service oriented environments in which business processes are typically executed. We consider SCA as a useful conceptual bridge between BPM and SOA, however we also promote SCA for the specification of the deployment package and therefore our implemented solution would also require the availability of an SCA runtime for running it, in addition to the other engines that are necessary for executing the business processes once deployed.

IV. ABSTRACT MAPPINGS

When defining business processes, individual business process activities can be connected to the service-execution capabilities of the enterprise, thus allowing any business process to be easily translated into an executable workflow on the platform of choice. This capability is enabled in our approach by providing mappings for each activity in order to specify how it should be grounded in the SOA. These mappings are done with idealized or abstract services in order to ensure better portability across the enterprise, as well as encourage proper adoption of good SOA-practices in future evolutions of the enterprise SOA. These abstract services (AS) would then be further connected to the real services in the repositories. The creation of these mappings would typically be performed by IT experts that have a good understanding of the domain and who envisage an ideal connection to a SOA, i.e. to imaginary services that would best be suited for executing business processes in that domain. These imaginary idealized services, referred to later as Abstract Services would not necessarily correspond on a 1 to 1 basis with business activities. That is because there are important differences in concerns when defining business elements and when defining the service infrastructure, due to varying needs for reusability, performance and evolution of these two layers.

We assume in this section that SCA is used to provide mappings, and we introduce the notion of one Mapping Composite per concept. Note that we assume the use of SCA here because we believe it best fits this role, but this is not a strong requirement. The actual mapping process, from concept to architectural composite is out of the scope of this paper.

This Mapping Composite (MC) contains the necessary logic for the usage of the concept in various contexts. The logic may involve several of enterprise SOA services, human task capabilities or other assets and it can indeed require new functionality to be created (typically this would take the form of 'glue' code for combining existing assets correctly). Glue code refers to behaviour that would not normally reside in a reusable service in the SOA yet would be identified as important for executing an activity. The services that MCs refer to are in abstract form, i.e. they are not initially bound to existing web-services in the enterprise repositories. Rather, these bindings are done at the last moment in the deployment process, as presented in the sections below. This ensures that MCs contain an idealized SOA representation of the concepts they represent, which can be moved around various enterprise deployments and SOA repositories. This is important in cases where a company needs to provide the same business process capabilities to different customers with various legacy SOAs.

Fig. 1. gives an example of a possible layout for a SCA Mapping Composite. Dark blue boxes represent components. Green inputs represent services offered (by components and the composite as well). Magenta outputs represent references (dependencies). In the example, the composite contains three components that together provide the internal logic of the mapping. The Orchestration Logic component deals with combining the results from the remaining two components and providing the service to the composite ("promoting" its service as the composite service, in SCA jargon).

The Developed Functionality component illustrates the fact that some activities may need entirely new logic to be developed, while Service Adaptation highlights the possible reuse for different needs of common abstract services in the idealized SOA. A Mapping Composite is developed only once per activity. This effort is typically done by IT experts together with domain experts in order to properly connect the business concepts to the IT assets.

Shifting the focus to business processes created using activities, it is important to note that just as for each activity
there is an SCA composite, in the case of BPs there will be a corresponding composite as well. This BP Composite will connect the business process to the IT assets by grouping the individual MCs of the activities involved in the process into a larger entity. By automatically reusing the connections that individual MCs have to the SOA services, the BP Composite will inherit this connectivity, exposing it in a particular way that is appropriate for the business process. A possible pattern for creating a BP Composite for a given BP is illustrated in Fig. 2. and involves several architectural choices (note that discussing specific patterns for creation of BP Composites is not directly in the scope of this paper but a description of the concept is useful for understanding the functionality of the deployment process). There central part is the orchestration component that contains the BP logic. This orchestration component is connected to the individual composites corresponding to each activity used in the BP. The MCs corresponding to the individual activities used in the BPs are used in the BP Composite as components, exposing (promoting) their references to the outside of the BP composite (the BP Composite will therefore have a set of references obtained by the reunion of the sets of references exposed by the individual MCs).

Note that each MC is used only once in the BP Composite even if it appears multiple times in the BP. This is because the orchestration logic in the BP Logic component deals with possible repetitions through the same dependency.

To illustrate the effects of using the abstract services in the deployment system, we consider the case of a simple business process consisting of four activities of various natures, connected to the company’s SOA. Fig. 3. illustrates the typical approach used in many environments today, with the BPMS acting as a black-box to bind process activities to services. In contrast, Fig. 4. illustrates our approach by showing the connection across the enterprise achieved by bridging the process activities to the services layer through the mapping composites in the deployment operation.

The top layer shows the business process with its activities mapped to individual Mapping Composites.

The second layer shows the resulting Business Process Composite, obtained through the combination of individual Mapping Composites corresponding to its activities. As mentioned above, the dependencies of these Mapping Composites are inherited and exposed by the BP Mapping Composite. The third layer shows the Abstract Bindings repository containing the bindings that connect abstract services to existing services. The dotted red arrows connecting the BP Composite dependencies with the elements of the ABR show the effect of the binding operation in the deployment.
process. Since the bindings make the connection to the services in the SOA (illustrated by dark dotted arrows), the connection is established between the BP Composite and the SOA services. This cascading connectivity achieved through the deployment mechanism ensures that the connection between the business processes (first layer) and the enterprise services (last layer) is done in a portable and transparent manner. If services in the bottom SOA layer evolve, only the ABR and the services layers are affected, leaving the business process layer intact. IT experts can perform changes to the ABR at any time in order to update it with regard to the evolutions in the real services of the enterprise SOA. Conversely, if business processes or their mappings to idealized services need to change and evolve, these changes are not constrained by what is available in the SOA layer. In fact, because the ABR elements corresponding to individual BPs are stored and archived with the corresponding deployment operations, previous versions of the mappings between BPs and the SOA can be reused, in cases where existing evolutions do not correspond to the requirements. This freedom to change the ABR without fear of breaking existing working deployments can encourage evolution in the services layer (supported by proper versioning mechanisms). This is in contrast to the majority of approaches today where business process design is intertwined with capabilities in the services layer either directly or indirectly through the use of inflexible templates connected to the SOA.

V. THE DEPLOYMENT PROCESS

The BP Composites are used by the deployment process which, in its proposed implementation has several notable properties. It is independent of the execution platform, thus guaranteeing platform and vendor independence. In addition, deployment operations are controlled and archived, allowing advanced governance in line with agility needs of dynamic enterprises, where BPs can change quickly and often. The deployment system itself is based on open-standards (most notably SCA), facilitating evolutions. Lastly, since the deployment process has the same nature as the business processes that it targets, it enables seamless reuse of governance solutions that are available for BPs. This can facilitate the evolution of the deployment process itself during the lifetime of the application. The activities of the deployment process are:

1. Obtain the mappings for all the activities in the BP (i.e. the Mapping Composites)
2. Generate the business process composite, which contains the aggregated mappings for the entire process
3. Validate the composition by making sure the process is executable
4. For each of service dependency of the MC, look up its binding in the Abstract Bindings Repository (ABR) which conceptually sits 'above' the SOA repositories available in the enterprise and contains binding pairs whose sole purpose is to connect Abstract Services from the mappings to 'real' services in the SOA. The 'real' services can be services found directly in the SOA repositories available, or indeed newly created adaptors that have no significant business logic but that offer technical glue code to interface with the existing SOA services.

5. Analyse the target system and generate the appropriate artefacts based on its requirements. The generation process uses the BP Composite as the starting point and proceeds to generate executable artefacts such as BPMN 2.0 or BPEL scripts, SOA constructs such as deployment units for deploying on an Enterprise Service Bus, or indeed an executable BP Composite that has the details necessary for a potential SCA runtime (if used as the SOA platform of choice for BPs).

6. Connect to the target platform and request deployment of the generated artefact, while notifying the governance system that the deployment operation has finished.

The deployment process is contained in the deployment package which has the SCA representation shown in Fig. 5. It is designed as an SCA composite that contains the required logic and connectivity for realizing the deployment operations. It is important to note that this composite corresponds to a deployment package instance, for a given BP. Such a composite will be generated for each BP when ready for deployment. This generation will create a .composite file that will contain the deployment logic and data for the respective BP including the current set of activity mappings and the entire context of the BP design. By archiving this .composite file we are able to preserve a historical perspective of each deployment operation for each BP and to revert to any previous successful deployment even if elements in the design have changed. The structure of this composite matches the required operations for deployment as well as the services and references to control and monitor it. In addition the composite contains a bridge for connecting design tools with monitoring information.

Fig. 5. The Deployment Logic Composite

The **Deployment Engine** component is the central element of the composite. It orchestrates (executes) the deployment process containing the 6 activities described above. Note that the use of a central orchestration component in a composite that
realizes the functionality of a process follows the same pattern that could be used when mapping BPs to BP Composites. The Deployment Engine exposes the Deploy Service which is used by external entities (such as BP editors or other user interfaces) to initiate, stop or resume a deployment process for a given BP. It also exposes a Supervision Service that can be used to interrogate about the status of the deployment process, which could be useful for longer-running deployments. Apart from the references that this component needs for connecting to other internal components, the Deployment Engine also exposes a reference to an external Design Governance System that can be connected to any service offering governance capabilities where deployment is of particular concern. The main envisaged use for this is for notifications of deployment operations (success/failure) to be sent to interested listeners, such as the design tools that have triggered the deployment or indeed any other external governance tools that need to be aware of them. This push-type functionality is complementary to the Supervision Service, which in turn aims to provide pull-type, on-demand information while the deployment process is running. The external dependencies of the Deployment Engine can be satisfied by specific adapters that correspond to the execution environments that are targeted. Many enterprise-level BPMS and SOA environments expose APIs that can be leveraged for this purpose.

The BP Provider component actually contains the BP description embedded in it. This information has been attached to the component upon the creation of the composite. This information can be stored for archival reasons. Similarly, the Activity Mapping Extractor contains the activity mappings that are relevant for this process, i.e. all the MCs for the BP stored in the BP Provider. This information is also stored as part of the composite archival. Together, these two sets of data ensure that the deployment package instance is self-contained and, if required in the future, any deployment operation can be examined and repeated using its full original context, regardless of whether BPs or their corresponding mappings have evolved.

The Process Composite Generator component contains the logic necessary to build the BP Composite that represents the business process. It uses the MCs provided by the Activity Mapping Extractor and generates the BP Composite as detailed above. After the BP Composite is created, the Deployment Engine needs to instantiate the bindings that refer to idealized services and connect them to actual services in the enterprise SOA. Initially these bindings refer to abstract services as they are inherited from MCs. When generating a deployable artefact, the engine needs to swap these AS for their counterparts in the SOA. This binding process will perform the following steps, for each abstract service encountered:

1. Check the Abstract Bindings Repository for existing associations between AS and real services
2. If an association has been found (because a service mapping has already been performed), save and return the association and continue the deployment process.
3. If there is no association, the Deployment Engine has no way of choosing which service to bind the AS to. Therefore a notification will be sent to the governance system to trigger a service mapping request. The deployment will not continue until the request has been satisfied.
4. The service mapping request will be presented to the appropriate enterprise role/person which will be responsible for connecting the AS to an actual service through a key-value pair in the Abstract Bindings Repository. Services to connect to may come from the existing overall enterprise repositories or indeed be specific composite services created specifically for the purposes of mapping with the corresponding AS from the MCs. The role that is responsible for the mapping will decide whether there is an appropriate service available or one must be created.
5. If a mapping service needs to be created, then any technology may be used (SCA is a good candidate). The mapping service is then bound to the AS. If an appropriate service already exists, the mapping is done directly to it.
6. The process finishes when each of the exposed dependencies from the BP Composite is satisfied with an appropriate binding in the Abstract Bindings Repository and ensures that these bindings are saved together with the rest of the data in order to make the operation self-contained and repeatable.

The Deployer component is invoked by the Deployment Engine when the BP Composite is available in order to generate the actual artefacts to be deployed on the target platform. The BP Composite itself is not deployable in its initial form, as it does not necessarily contain all the details required by the platform and its form may not be suitable for all platforms. Therefore the Deployer takes the BP Composite and sends it to the Artefact Generator that is the component capable of generating the actual deployable artefact. The logic in the Artefact Generator analyses the target platform and generates the appropriate artefacts depending on platform requirements. It can support multiple types of target platforms and the illustration shows the internals of its architecture with components for dealing with BPMN 2.0 scripts, executable SCA, as well as a placeholder for any other platforms. Note that the Artefact Generator component is in fact a composite that is used in the context of the deployment composite as a component, thus illustrating the hierarchical nature of SCA composition.

The Deployer also exposes a dependency, promoted at the composite level, to the Execution Platform. This dependency can be used to connect the deployment system to a service that can offer information about the target platform as well as potentially offering a mechanism to actually enact the deployment onto the execution platform itself. The deployment composite also offers the possibility to connect any Monitoring Service consumer to an external Runtime Governance System (available as APIs in many commercial BPM offerings). This can be used to provide real-time or aggregated monitoring...
information about the deployed BP as well as about other events coming from the runtime platform, to any consumer. The important aspect in this design choice is that the deployment composite itself, being the unit of deployment for a BP, becomes also the connection between design tools and the runtime platforms, upon its configuration by deployment specialists. Therefore, BP-related tools need not be aware of runtime platforms and instead connect only to the SCA composite that they generate, after its configuration and installation into the SCA deployment runtime. Furthermore, the Monitoring Service endpoint can be circulated among various interested parties which, by using it, would become instantly connected to monitoring information about the running BP to which this composite relates.

VI. RELATED WORK

Most existing commercial BPM solutions do not use DSLs nor do they promote the definition of concept maps, rather they promote the use of generic business process definition languages such as BPMN, which are then translated into executable artefacts such as BPEL scripts and deployed on runtime engines [5][6][7]. Therefore, deployment in such contexts is performed in proprietary ways for each industrial product. The translation between business processes and SOA is usually done either using tools for individually connecting BPs to execution capabilities or through industry vertical-specific templates that are already bound directly to the SOA (e.g. templates for financial processes or for healthcare processes). Once the deployment operation has executed, traces of its execution are not preserved so reverting to an older deployment environment is limited to versioning capabilities for the execution artefacts. In our approach the deployment operations themselves are versioned and administrators can revert to an older set of bindings independently of the evolution in the execution layer.

In the more general area of deployment for applications, deployment solutions include approaches and standards such as OMG Deployment & Configuration specification [8], DeployWare [9], Jade / Jasmine [10] or Smartfrog [11] which focus on generic mechanisms that are useful in describing individual software components and their dependencies such as web-pages, web servers and individual node configurations for cluster-based deployments. Such approaches are not particularly fit for business processes where a much finer granularity is needed in controlling various mappings between business process elements and IT assets. Naturally, these frameworks could be used to deploy BPs as black-boxes, but they would not bring any significant advantages for improving agility or governance, which are the main targets of our approach.

VII. CONCLUSIONS AND FUTURE WORK

We believe that the use of domain specific languages or the definition of well-defined domain concepts bound to an idealized SOA has great potential for service-based enterprises. Part of the problem in implementing end-to-end business process management solutions based on these paradigms is to efficiently integrate business process design editors and governance tools with operational concerns and runtime environments. Deployment systems are essential pieces in this puzzle because they play an important role in bridging design with runtime spaces.

This paper proposes a novel approach to deployment which is geared towards highly evolving business process management environments. The mechanism we describe leverages state-of-the-art SOA-based approaches and standards to create unique deployment compositions that can be stored, shared and executed in order to deploy business processes. In contrast to the other approaches we are aware of, our solution entails a portable deployment packaging mechanism that is instantiated for each business process and that stores and executes individual deployment operations in a manner that is independent of, but which leverages, existing BPMS and SOA environments. This ensures good governance of business process lifecycle, an essential ingredient for agile business systems.

The paper does not present in detail the algorithms executed by each component in the presented architecture. We limit the descriptions to functional representations of the deployment process. Our contribution focuses on the proposed architecture and technological context of the mechanism that ensures that business processes are deployed in an efficient manner across a wide range of platforms, while integrating advanced governance solutions. For instance, we believe that the mapping process could be improved and even parameterized for different needs of enterprises. Perhaps for each BP Composite there could be extra components to realize non-functional work that applies to all BPs. The pattern of using a central orchestration component connected to the composites corresponding to MCs is the simplest form this BP Composite could take. It could easily be accommodated to have more functionality.

We also do not describe the technical choices for the generation of the composites as we believe they are details out of the scope of the paper. There are various ways of doing such generation, and we envisage the use of metamodel-based generational frameworks such as the Eclipse Modelling Framework [12] that would naturally integrate with the wide variety of process editors based on the Eclipse SOA platform [13].

Work is underway to implement a fully operational prototype of the presented approach that will be integrated in an environment representative of typical enterprise deployments.

REFERENCES