BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications

Oliver Kopp, Tobias Binz, Uwe Breitenbücher, and Frank Leymann

Institute of Architecture of Application Systems,
University of Stuttgart,
Universitätsstraße 38, 70569 Stuttgart, Germany
{kopp,binz,breitenbuecher,leymann}@iaas.uni-stuttgart.de

Abstract. TOSCA is an upcoming standard to capture cloud application topologies and their management in a portable way. Management aspects include provisioning, operation and deprovisioning of an application. Management plans capture these aspects in workflows. BPMN 2.0 as general-purpose language can be used to model these workflows. There is, however, no tailored support for management plans in BPMN. This paper analyzes TOSCA with the focus on requirements on workflow modeling languages to come up with a strong link to the application topology with the goal to improve modeling support. To simplify the modeling of management plans, we introduce BPMN4TOSCA, which extends BPMN with four TOSCA-specific elements: TOSCA Topology Management Task, TOSCA Node Management Task, TOSCA Script Task, and TOSCA Data Object. Portability is ensured by a transformation of BPMN4TOSCA to plain BPMN. A prototypical modeling tool supports the strong link between the management plan and the TOSCA topology.

Keywords: Cloud Computing, Service Management, Management Plans, BPMN Extension.

1 Introduction

To decrease cost and prevent vendor lock-in, portability of applications is—especially in the area of cloud computing—very important. To face this challenge, the *OASIS Topology and Orchestration Specification for Cloud Applications* (TOSCA) [1] is a way to describe the structure of portable services in a topology and their management as workflows, so called plans. A topology consists of node templates which offer management operations to create new instances or deploy software artifacts, for instance. Currently, the BPMN management plans directly point to the service interfaces and are not linked to the topology anymore. Therefore, we propose BPMN4TOSCA, a domain-specific BPMN [2] extension, which enables convenient integration and direct access to the TOSCA topology and provided management operations.

J. Mendling and M. Weidlich (Eds.): BPMN 2012, LNBIP 125, pp. 38–52, 2012.

Our contribution is fourfold: (i) Analyzing the requirements for modeling TOSCA management plans using BPMN, (ii) the BPMN extension BPMN4TOSCA allowing tight integration of topology data and management operations into plans, (iii) a transformation of BPMN4TOSCA into standard-compliant BPMN, and (iv) a prototypically implemented BPMN4TOSCA support in a TOSCA modeling tool.

The paper starts with a general introduction to the concepts behind TOSCA: the topology templates and the management plans (Sect. 2). Section 3 presents a concrete TOSCA use case, where the concepts of TOSCA are detailed. Based on this use case, general requirements on the plan modeling language are derived in Sect. 4. Based on the requirements, Sect. 5 presents BPMN4TOSCA, a domain-specific variant of BPMN supporting TOSCA management plan modeling. As typical workflow engines are not capable of executing extended BPMN, we present in Sect. 6 how to transform BPMN4TOSCA to plain BPMN 2.0 to enable execution on standard workflow engines. Section 7 presents a prototype supporting modeling TOSCA documents including management plans expressed in BPMN4TOSCA. Subsequently, Sect. 8 surveys on related work including the field of modeling composite applications and service management. Finally, Sect. 9 concludes and presents an outlook on future work.

2 Fundamentals

The *Topology and Orchestration Specification for Cloud Applications*, TOSCA for short, is an exchange format to describe the components of composite applications, their relations, as well as how to manage them. TOSCA is currently standardized in an OASIS Technical Committee[1]. Its main goal is enabling portability of composite applications between different cloud management environments to prevent vendor lock-in and increase automation in service management. To facilitate this, a *service template* is described in TOSCA, as denoted in Fig. 1. It consists of two major parts: the service's topology and management plans. The *topology* captures the structure of the composite application as a graph of node templates which are semantically connected by relationship templates. Each template is of a certain type. The type defines its properties, lifecycle states, policies, related artifacts, and management operations. Types in TOSCA are extensible, i. e., they can be defined as part of the service template and are not a predefined closed set. *Deployment artifacts* attached to a node define how this node is implemented. For instance, a virtual machine image may be a deployment artifact for the Linux node type or a Java Web archive for the Web application node type. The management operations supported by a node, for example, start, backup, or upgrade a node, are defined as WSDL Web service, REST service, script, or a combination thereof. If a management operation is not provided by the deployment artifact itself, e. g., an application server offering an JMX management service, or an external service, i. e., Amazon EC2 to start up virtual machines in their cloud, it can be included inside the service

[1] `http://www.oasis-open.org/committees/tosca`

template as so called *implementation artifact*. This enables service creators to ship management and administration services as part of their service. All in all, a service template consists of node templates and relationship templates. Each node template has a node type and a relationship template has a relationship type. A service template is instantiated to a service instance, where the node templates become nodes and the relationship templates become relationships.

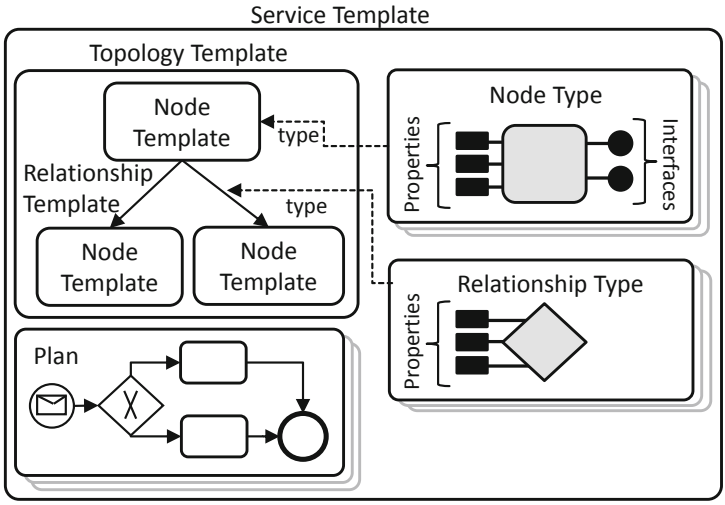

Fig. 1. Overview of TOSCA Building Blocks (adapted from [1])

TOSCA enables service creators to model the management aspects of services into plans. Plans express higher-level management tasks, which are, for example, how to setup the service, how to scale it up, back it up, or upgrade all operating systems. Having the management explicitly in the service template makes the management knowledge portable, reusable, and enables automation. Plans are modeled by the developer of the application or experienced operators ensuring widespread usage of their accumulated best practice knowledge and relieves enterprise IT from some of the management burden. Plans orchestrate the different management operations offered by the nodes to fulfill their task. The TOSCA specification defines three types of management plans: Build, modification, and termination plans. Technically, plans are workflows written, for example, in BPMN [2] or BPEL 2.0 [3]. Binz et al. [4] discuss the advantages of using workflow technology for plans. The key benefits are fault handling, compensation, auditing, parallelism, and integration of humans.

TOSCA requires a compliant management environment to run the service templates. We call such an environment "TOSCA container". After importing a new service template, the TOSCA container ensures, for example, that the implementation artifacts implementing the management operations are available before the service is instantiated for the first time. Additionally, the container's

responsibilities are to manage service templates and their instances, offer access to the topology model and instance data, and handle the deployment artifacts accompanying the service template. Before deploying the plans on a plan engine, the TOSCA container binds the plans to the respective endpoints. This is necessary as it is not known where the management operations have been deployed and in which management environment the service template will be executed. This binding is key to enable portable service management between different TOSCA containers. In this paper, we focus on the mechanisms required to use management operations and TOSCA container services in management plans.

TOSCA recommends BPMN 2.0 as workflow language to model management plans. Other workflow languages—such as BPEL 2.0—may also be used. In contrast to BPEL 2.0, BPMN 2.0 is currently the preferred choice as it offers a standardized graphical rendering and does not force the workflow graph to be acyclic [5]. Starting in version 2.0, BPMN has also a well-defined execution semantics. Tasks and the control flow between them are the central elements of BPMN determining what the workflow does and in which order. BPMN defines tasks to call services (*service task*), to execute scripts (*script task*), to trigger human actions (*human task*), and others which are not important in our context. In contrast to BPEL, data flow in BPMN is explicitly modeled by using data objects. Tasks and events read from and write to data objects by using data associations which may contain data transformation rules.

3 Use Case

In this section we describe a TOSCA use case used in the following section to derive the requirements towards BPMN4TOSCA. The use case describes an online bookstore whose architecture is presented in Fig. 2. Each component is rendered as a box representing a TOSCA node template. The dashed arrows denote relationship templates of type "hosted-on". The application uses Java Servlet and Java Server Pages technology and is packaged into a single WAR (Java Web archive) file. To run the application, the WAR file has to be deployed on a servlet container, *Apache Tomcat* in our case. The servlet container is hosted on an operating system, *Ubuntu 12.04 LTS* in the use case, which is hosted on a Amazon EC2[2] virtual server.

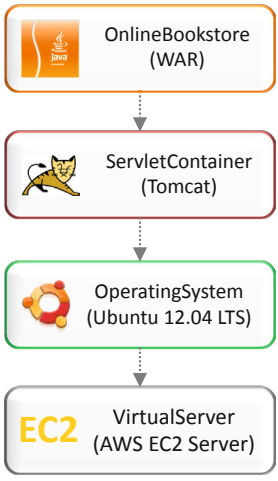

Fig. 2. Online Bookstore

In the following, we describe how to use the concepts of TOSCA to deploy and manage the online bookstore application without BPMN4TOSCA to show the limitations and inconveniences. To deploy the online bookstore, its components virtual server,

[2] `http://aws.amazon.com/ec2/`

operating system, servlet container, and the online bookstore application itself have to be deployed in the right order, typically from bottom up. The build plan orchestrates the management operations, scripts, and operations offered by the TOSCA container. The first step of the build plan in our use case is to create and start the virtual server using the operating system image defined in the topology, which establishes the hosted-on relationship defined in Fig. 2. Afterwards, the build plan invokes a management operation of the operating system to copy a bash script[3] onto the operating system. By invoking another management operation the script is invoked and installs the Tomcat servlet container on the operating system. This script uses Ubuntu's package management system to install Tomcat. After Tomcat is installed on the operating system, the servlet container is started by calling the operation `startService` offered by the operating system implementation artifact. The last step of building the application is deploying the online bookstore application on Tomcat. Therefore, the build plan invokes the `deployWar` operation implemented by the implementation artifact of the Tomcat node type and passes a reference to the WAR file. The operation deploys this WAR file—which also may be stored online—into Tomcat. All in all, the whole application is now deployed, running, and can be used.

4 Requirements

Based on the use case, we identified three requirements towards a solution which facilitates the tight integration of the management plans with the managed application topology.

TOSCA management plans typically process and manipulate properties of nodes and relationships, for example, the IP address of the virtual server node in our use case. In order to access and modify these instance properties, BPMN service tasks are used to call the respective TOSCA container APIs. Due to the fact that properties play a central role in management plans and, therefore, are heavily accessed by different tasks, the management plans get polluted with BPMN tasks. Thus, the management plans become complex and hard to understand. Business process research has shown that the maintainability and understandability of business processes decrease rapidly with the number of tasks. For example, Cardoso [6] proposes a measure for control-flow complexity increasing with the number of tasks and Reijers et al. [7] discuss the business process complexity in the context of modularization which is used to reduce the number of tasks. Therefore, requirement R1 for BPMN4TOSCA is reducing this complexity by providing ways to access and modify properties of nodes and relationships without modeling overhead in terms of business process elements.

In addition to R1, management plans must be able to access the TOSCA service topology model which is provided by the TOSCA container. TOSCA model access is required for dynamic plans, for example, to retrieve all nodes of a certain type. Therefore, requirement R2 is to enable the management plans to access the TOSCA topology model.

[3] `http://www.gnu.org/software/bash/manual/bashref.html`

To deploy, instantiate, and manage topologies, plans typically invoke management operations offered by nodes. The number of available management operations offered by nodes may become unmanageable for the modeler when there are many different nodes in the topology. Furthermore, as many operations will have similar names, e. g., `deploy` is a common operation name in this domain, and as operations may be spread across multiple TOSCA files, it becomes a complex task for the modeler to select the right management operation of the correct node. Thus, requirement R3 for BPMN4TOSCA is to ease the selection of management operations and to provide support for strong integration of management plans and management operations offered by nodes.

Scripts play an important role in the management of composite applications, especially during their deployment. They are widely used to perform installation and configuration tasks in systems management. Typically, these scripts are copied to the respective nodes and executed locally. TOSCA supports this concept by providing script operations which are attached to nodes. Hence, requirement R4 is: BPMN4TOSCA must support an easy and comfortable way to execute scripts on nodes.

5 BPMN4TOSCA: Enabling TOSCA Plan Modeling

In this section we introduce the BPMN language extension BPMN4TOSCA. To meet the requirements stated in the previous section, the design of BPMN4TOSCA consists conceptually of two parts: The first part provides a BPMN language extension (this section) and corresponding processing model, which defines the semantics of the extension (Sect. 6). The second part defines additional functionalities, which have to be provided by the modeling tool in addition to the language extension to provide the functionality (Sect. 7). Although the second part makes BPMN4TOSCA to more than an extension of BPMN, we nevertheless call the BPMN language extension BPMN4TOSCA, too.

The language extension consists of four new BPMN4TOSCA-elements, each accompanied with a graphical representation: TOSCA Topology Management Task (Sect. 5.1), TOSCA Node Management Task (Sect. 5.2), TOSCA Script Task (Sect. 5.3), and TOSCA Data Object (Sect. 5.4).

5.1 TOSCA Topology Management Task

The TOSCA Topology Management Task extends the BPMN service task in a way that standardized topology management operations offered by the container are predefined and can be directly used. An example operation is `getServiceTemplate` to get the TOSCA service template the plan works on. The selected operation is put in the attribute `operationRef` [2, p. 159]. This addresses requirement R2 of Sect. 4: By using a Topology Management Task, the operation can be directly chosen.

5.2 TOSCA Node Management Task

The TOSCA Node Management Tasks simplifies selecting and invoking management operations of nodes. It extends the BPMN Service Task. The node template id to work on is stored in the existing attribute `implementationRef` [2, p. 106]. The id itself is contained in the namespace of the service template. The selected operation is put in the attribute `operationRef`. This fulfills requirement R3: By using a Node Management Task, the user directly selects the node template to work on and the operation to call.

5.3 TOSCA Script Task

The TOSCA Script Task meets requirement R4 of Sect. 4 by providing the opportunity of referencing scripts and corresponding nodes on which they shall be performed. The TOSCA Script Task offers two possibilities to define scripts which should be performed on nodes: First, scripts can be defined inline the task itself, i. e., the script is part of the task description. Second, the TOSCA Script Task can reference scripts defined in TOSCA files as they are identified by unique ids. In addition to that, a TOSCA Script Task specifies the node on which the intended script has to be performed by using a unique id referencing to the corresponding node template defined in the TOSCA file.

The scripts must be able to be copied automatically to the nodes and executed on them. This is managed by the TOSCA container. For that, the container needs special operations provided by the nodes to enable this kind of generic script handling: Each of the target nodes has to provide an implementation artifact implementing an interface defining a set of pre-defined management operations prescribed by BPMN4TOSCA. The implementation artifact does the transformation from the generic container operation to the specifics of the respective scripting language, e. g., transforming parameter data types and setting environment variables.

The interface defines three main operations: `deployScript`, `runScript`, and `undeployScript`. `DeployScript` gets the actual script passed as parameter and returns a unique id which identifies the deployed script. `RunScript` gets this id and the input parameters for the script passed as parameter and returns the result of the script execution. `UndeployScript` also gets the id of the deployed script and undeploys it from the respective node.

The TOSCA Script Task inherits from BPMN's script task. In case the script is part of the task, the script task semantics and its attributes `scriptFormat` and `script` is re-used. In case the task references a script stored in the service template, these two attributes are not used. Instead three attributes are added: `scriptReference`, which references a script defined (or referenced) in

the TOSCA file and `targetNodeTemplateId`, which references the node template on which the script has to be executed, `targetNodeInstanceId`, defining the concrete instance of the node template if multiple instances are allowed.

5.4 TOSCA Data Object

The language extension meets requirement R1 of Sect. 4 by introducing a *TOSCA Data Object (TDO)*, which automatically provides access to runtime property information of nodes and relationships. TOSCA Data Objects provide information without the need to explicitly model BPMN service tasks requesting the respective information from the TOSCA container and sending modifications to the container. The data handling between TOSCA Data Object and TOSCA container is done automatically "behind the scenes" and invisible to the plan modeler. As soon as a TOSCA Data Object is defined in a plan, the referenced information is accessible and can be modified. The issue of dealing with multiple different plans using TOSCA Data Objects representing the same nodes or relationship properties concurrently lead to well-known transactional problems such as lost update or dirty reads breaking ACID properties [8]. This concurrent access to any information objects defined in the topology through different plans is a general problem, thus our approach does not need to deal with this issue as we assume that the TOSCA container is responsible for avoiding the concurrent execution of plans accessing the same information objects concurrently, i.e., the TOSCA container is responsible for plan scheduling. Plan scheduling denotes the order in which the plans are executed and not a refinement of the plans itself.

TOSCA Data Objects extends the BPMN data object by adding two TOSCA-related attributes to identify the nodes resp. relationships the TOSCA Data Object is reffering to: (i) A reference to the corresponding node template or relationship template whose properties should be reflected by the TOSCA Data Object, named `nodeTemplateId` and `relationshipTemplateId`, respectively. (ii) The optional attributes `nodeInstanceId` and `relationshipInstanceId` identify the concrete node instance or relationship instance if there are multiple instances, as defined by the min and max instances attributes in TOSCA. In case the data object is a collection, only the template id attribute is allowed. Iterating over each referenced node (or relationship) instance is enabled by the `inputDataItem` property of a looping BPMN task [2, p. 192].

6 Processing BPMN4TOSCA

The BPMN4TOSCA extension leads to a non-standards-compliant BPMN and, therefore, needs special treatment. The presented extensions are run-time extensions: They introduce new functionalities which are not natively supported by the executing environment. Nevertheless, there are two options to enable the extensions during runtime: (A1) extend the modeling tool *and* the workflow engine to support the new functionality and (A2) transform the functionalities into

standards-compliant executable elements before deployment [9]. When choosing option A1, the workflow engine is extended to support the new functionality. The drawback is that the extension is supported by standards-compliant engines only if they also support the extension. To avoid requiring an implementation of the extension at workflow engines, one may provide a transformation of an extended process model to a standard process model (option A2). This generated model typically depends on external services to offer the functionalities.

In the case of BPMN4TOSCA, the transformation to plain BPMN is possible. We opted for a transformation instead of extending the workflow engine for the following reasons:

- Extending the execution environment for new capabilities renders TOSCA non-portable, because only standards-compliant BPMN is portable across different execution environments.
- Providing a modeling tool supporting the BPMN4TOSCA approach enables transformation of BPMN4TOSCA to plain BPMN by the export functionality of the modeling tool. Thus, BPMN4TOSCA is only visible in the tool and transparent to the execution environment as the semantics and functionalities remain only implicitly in the exported files while keeping all benefits of the approach for the modeler.

The following subsections show how the four BPMN4TOSCA tasks are transformed into standards-compliant BPMN:

6.1 TOSCA Topology Management Tasks

The TOSCA Topology Management Task references operations provided by the TOSCA container. A new implementation reference is added. It points to the concrete port type, where the WSDL service of the TOSCA container is offered. This indirection is necessary as the management operation interface is not standardized in TOSCA.

6.2 Transformation of TOSCA Node Management Task

The TOSCA Node Management Task references a node template in a service topology and one operation. This information is replaced by a reference to the concrete WSDL port type and WSDL operation of the referenced operation.

6.3 Transformation of TOSCA Script Tasks

TOSCA Script Tasks need to be transformed into BPMN service tasks as conceptually shown in Sect. 5.3. A Script Task references the script and the node on which it has to be performed. During plan transformation, the TOSCA Script Task is replaced by three BPMN service tasks which are executed in sequence: deploy script, run script, undeploy script. The TOSCA container binds these three tasks to the services offered by the implementation artifact of the corresponding node. These services implement the interface introduced in Sect. 5.3.

To deploy the script on the target node, the first task invokes the `deployScript` operation offered by the implementation artifact's service and passes the entire script and the type of the script, e. g., Ant script, to the service. The type of the script is required for selecting the corresponding script handlers, which define how each script type is processed. This is completely transparent to the plan. The required logic is implemented either in the implementation artifact or the node itself. For some script types, there possibly is a need for some kinds of agents installed directly on the node to interact with the implementation artifact, others are able to implement the whole script handling in the implementation artifact and the node remains totally unaware of executing scripts. This is script type specific implementation design and therefore out of scope: BPMN4TOSCA defines only the interfaces, not how to deal with scripts. Thus, for each script type, there is a specific implementation, i. e., a script handler, for dealing with this type. Referenced scripts have to be resolved by the TOSCA container before passing the script to the service. After successful deployment, the `deployScript` operation returns a unique identifier used to identify the deployed script for further steps, i. e., execution and undeployment of the script.

The second BPMN service task invokes the `runScript` operation and passes the script identifier and corresponding input parameters which are contained in the input data object associated with the TOSCA Script Task to the service. The implementation artifact uses the identifier to find the deployed script handler, passes the input parameters, and executes it. After successful execution, the operation returns the output parameter which is in turn returned from the script via its script handler and writes the values to the associated output data object. Passing data from data objects to data input and from data output is done using the BPMN way using `dataInputAssociation` and `dataOutputAssociation` [2, p. 224]. Defining these associations is out of scope and has to be done by the modeler.

The third BPMN service task invokes the `undeployScript` operation and passes the script identifier. The operation undeploys the executed script.

6.4 Transformation of TOSCA Data Objects

The transfer of property information data from the TOSCA container to plans happens transparently to the modeler. TOSCA Data Objects are converted to BPMN data objects. For reading TOSCA data, additional BPMN service tasks

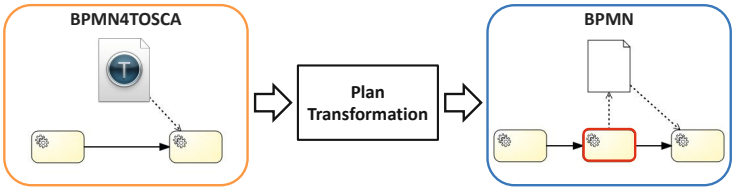

Fig. 3. Injection of Service Tasks (framed in red color) for Accessing Data

are injected. They request information from the TOSCA container and write the information to the data objects as shown in Fig. 3. Vice versa, if BPMN tasks modify data in TOSCA Data Objects, the modified information is sent to the TOSCA container via additional service tasks, too. To achieve this in a coherent way, the TOSCA container offers standardized interfaces to access and modify property information of nodes and relationships.

7 Prototype

Valesca[4] is a modeling tool with full support for TOSCA. Valesca uses the Signavio Core Components[5], which are the commercially-supported enhancements of Oryx [10]. Besides creating service templates, Valesca supports creation of custom node types and relationship types. In the BPMN plan modeling component, the BPMN4TOSCA tasks and data object are added the palette.

When dragging a TOSCA Topology Management Task (cf. Sect. 5.1) from the palette into the plan, the modeling tool suggests a list of topology management operations offered by the container.

When dragging a TOSCA Node Management Task (cf. Sect. 5.2) from the palette into the plan, the modeling tool lists all node templates contained in the topology. After selecting one template, the tool lists all corresponding management operations to let the modeler select the appropriate one. Then, a TOSCA Node Management Task is created having the TOSCA attributes set accordingly. Management operations need input parameters and return values. BPMN foresees the usage of `dataInput` and `dataOutput` [2, pp. 213 and 231]. The data types of the input and output are corresponding to the input and output parameter types defined in the node template's management operation and are generated automatically by the modeling tool.

When dragging a TOSCA Script Task (cf. Sect. 5.3) from the palette into the plan, the modeling tool lists all node templates contained in the topology. After selecting one template, the tool lists all corresponding management operations with script operations as implementation to let the modeler select the appropriate one. The modeler may also choose to discard the choice to specify a script stored directly in the script task.

The modeling tool supports TOSCA Data Objects (cf. Sect. 5.4) in two ways: (i) It provides a separate TOSCA Data Object element in its palette and (ii) manages the corresponding schemas of the properties. When dragging a TOSCA Data Object from the palette into the plan, the modeling tools lists all node templates and relationship templates contained in the topology. One can be chosen to have their property data reflected via the data object. Properties are stored as XML documents defined by an XML schema document (XSD) [1, Sect. 4.2 and 5.2]. In the BPMN modeling tool the TOSCA Data Objects must be typed with the XSD to enable the modeler to extract and process information contained in the

[4] `http://www.cloudcycle.org/valesca/`
[5] `http://code.google.com/p/signavio-core-components/`

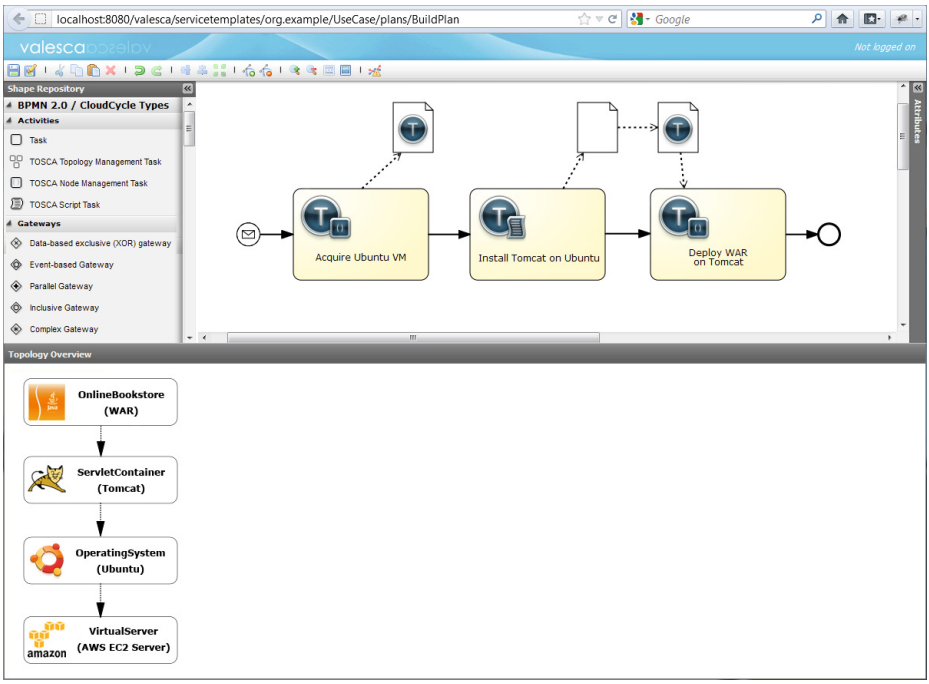

Fig. 4. Deploying the Example Bookstore Application: BPMN4TOSCA Plan in Valesca

properties. Therefore, the modeling tool supports recognizing TOSCA Data Objects, which are syntactically only identified by the additional TOSCA-related attributes, and injects the corresponding `itemDefinitions` whose `structureRef` attributes reference the corresponding XSD.

Besides offering a palette, the modeling tool offers drag'n'drop from the topology model to the BPMN plan model. At the drop of a node template, Valesca asks the modeler to whether he wants to add a TOSCA Data Object or a TOSCA Node Management Task. After the selection, Valesca continues as described in above. The dragging area is shown in Figure 4, which also presents the deployment plan for the use case.

8 Related Work

Extending modeling languages is a common technique to tailor them towards specific needs. For instance, there are at least 62 BPEL extensions including modeling and runtime extensions [9]. The classification in [9] shows that there are a number of design time BPEL extensions which are transformed into plain BPEL. Zor et al. [11] propose a BPMN extension for the manufactoring domain to explicitly handle products and resources. The inclusion of security aspects, such as access control or intrusion detection, into BPMN is described by

Rodríguez et al. [12] through a set of new annotations. However, the presented BPMN extensions do not address the execution of the extended BPMN processes and, therefore, not the transformation to an executable format.

Brucker et al. [13] present SecureBPMN, which is a methodology for secure and compliant business processes covering modeling and runtime. This includes a BPMN extension to add requirements such as access control or separation of duty into the process model. To address the business process runtime, Brucker et al. use a model-based approach to push the security and compliance requirements as configurations into existing systems. The tool chain was prototypically implementation based on Activiti[6], extending the Eclipse designer and process engine.

Discussions on business process transformations usually regard the business-IT-gap and transform high level processes into lower level, more technical, business processes. For instance, Stein et al. [14] survey on transformations to BPEL. Due to the fact that BPMN 2.0 process models are executable, they can be directly deployed to workflow engines for execution. Typically, BPMN 2.0 is transformed to a proprietary meta model of the workflow engine [15].

Before the standardization of TOSCA there have been different approaches in research and practice to model composite applications: Cafe [16] uses a declarative application model to deploy composite application which defines a `depends-on` and `deployed-on` relationship. Two approaches using UML [17] to describe the applications or architectures are presented by Machiraju et al. [18] and Arnold et al. [19]. An extensive overview on related work in the field of composite application and enterprise topology modeling is presented by Binz et al. [20].

9 Conclusions and Outlook

In this paper we motivated the upcoming OASIS standard TOSCA and showed how management plans enable the portability of services and their management. We argued that the integration between the service topology and management plans is important for the service creator modeling TOSCA service templates. To offer a tight integration, we propose a BPMN extension called BPMN4TOSCA adding four TOSCA-specific elements to BPMN: (i) The *TOSCA Topology Management Task* to access the TOSCA service topology from management plans, (ii) the *TOSCA Node Management Task* to invoke management operations of nodes, (iii) the *TOSCA Script Task* to execute scripts defined in the TOSCA service topology on nodes, and (iv) the *TOSCA Data Object* to read and write properties of nodes and relationships. We described the integration into a modeling tool and prototypically implemented our approach in the TOSCA modeling tool Valesca. Due to the fact that TOSCA containers use standard BPMN-compliant workflow engines, we showed how to transform BPMN4TOSCA into plain BPMN.

The set of TOSCA topology management operations is not yet fixed. In the context of the CloudCycle project we are working on an open source TOSCA

[6] `http://www.activiti.org`

container, which will provide new management operations to be included in Valesca. The industry partners of the CloudCycle project have started to use Valesca. We are going to use their feedback to further improve user's experience in modeling TOSCA with Valesca.

Acknowledgments. This work was funded by the BMWi project CloudCycle (01MD11023).

References

1. OASIS: Topology and Orchestration Specification for Cloud Applications Version 1.0 Working Draft 07 (June 2012), https://www.oasis-open.org/committees/download.php/46274/TOSCA-v1.0-wd07.zip
2. Object Management Group (OMG): Business Process Model and Notation (BPMN) Version 2.0, OMG Document Number: formal/2011-01-03 (2011)
3. OASIS: Web Services Business Process Execution Language Version 2.0 – OASIS Standard (2007)
4. Binz, T., Breiter, G., Leymann, F., Spatzier, T.: Portable Cloud Services Using TOSCA. IEEE Internet Computing 16(03), 80–85 (2012)
5. Kopp, O., Martin, D., Wutke, D., Leymann, F.: The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages. Enterprise Modelling and Information Systems 4(1), 3–13 (2009)
6. Cardoso, J.: How to Measure the Control-flow Complexity of Web Processes and Workflows. In: Workflow Handbook 2005, pp. 199–212 (2005)
7. Reijers, H.A., Mendling, J.: Modularity in Process Models: Review and Effects. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 20–35. Springer, Heidelberg (2008)
8. Weikum, G., Vossen, G.: Transactional Information Systems. Morgan Kaufmann Publishers (2002)
9. Kopp, O., Görlach, K., Karastoyanova, D., Leymann, F., Reiter, M., Schumm, D., Sonntag, M., Strauch, S., Unger, T., Wieland, M., Khalaf, R.: A Classification of BPEL Extensions. Journal of Systems Integration 2(4), 2–28 (2011)
10. Decker, G., Overdick, H., Weske, M.: Oryx – An Open Modeling Platform for the BPM Community. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 382–385. Springer, Heidelberg (2008)
11. Zor, S., Leymann, F., Schumm, D.: A Proposal of BPMN Extensions for the Manufacturing Domain. In: Proceedings of the 44th CIRP Conference on Manufacturing Systems (2011)
12. Rodríguez, A., Fernández-Medina, E., Piattini, M.: A BPMN Extension for the Modeling of Security Requirements in Business Processes. IEICE Transactions on Information and Systems 90(4), 745–752 (2007)
13. Brucker, A.D., Hang, I., Lückemeyer, G., Ruparel, R.: SecureBPMN: Modeling and Enforcing Access Control Requirements in Business Processes. In: ACM Symposium on Access Control Models and Technologies (2012)
14. Stein, S., Kühne, S., Ivanov, K.: Business to IT Transformations Revisited. In: Ardagna, D., Mecella, M., Yang, J. (eds.) BPM 2008 Workshops. LNBIP, vol. 17, pp. 176–187. Springer, Heidelberg (2009)

15. Leymann, F.: BPEL vs. BPMN 2.0: Should You Care? In: Mendling, J., Weidlich, M., Weske, M. (eds.) BPMN 2010. LNBIP, vol. 67, pp. 8–13. Springer, Heidelberg (2010)
16. Mietzner, R.: A Method and Implementation to Define and Provision Variable Composite Applications, and its usage in Cloud Computing. Dissertation, University of Stuttgart, Germany (August 2010)
17. OMG: Unified Modeling Language, UML (2011), http://www.omg.org/spec/UML
18. Machiraju, V., Dekhil, M., Wurster, K., Garg, P., Griss, M., Holland, J.: Towards generic application auto-discovery. In: IEEE/IFIP Network Operations and Management Symposium (2000)
19. Arnold, W., Eilam, T., Kalantar, M., Konstantinou, A.V., Totok, A.A.: Pattern Based SOA Deployment. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 1–12. Springer, Heidelberg (2007)
20. Binz, T., Fehling, C., Leymann, F., Nowak, A., Schumm, D.: Formalizing the Cloud through Enterprise Topology Graphs. In: Proceedings of 2012 IEEE International Conference on Cloud Computing (2012)

All links were last followed on June 29, 2012.