# Chapter 22
# TOSCA: Portable Automated Deployment and Management of Cloud Applications

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

**Abstract** Portability and automated management of composite applications are major concerns of today's enterprise IT. These applications typically consist of heterogeneous distributed components combined to provide the application's functionality. This architectural style challenges the operation and management of the application as a whole and requires new concepts for deployment, configuration, operation, and termination. The upcoming OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) standard provides new ways to enable portable automated deployment and management of composite applications. TOSCA describes the structure of composite applications as topologies containing their components and their relationships. Plans capture management tasks by orchestrating management operations exposed by the components.This chapter provides an overview on the concepts and usage of TOSCA.

## 22.1 Introduction

The increasing use of IT in almost any part of today's enterprises leads to a steadily increasing management effort, a challenge for enterprises as each new system or technology increases the degree of complexity [11]. This can be tackled by automation of

T. Binz (✉) · U. Breitenbücher · O. Kopp · F. Leymann
University of Stuttgart, IAAS,
Universitätsstr. 38, 70569 Stuttgart, Germany
e-mail: binz@iaas.uni-stuttgart.de

U. Breitenbücher
e-mail: breitenbuecher@iaas.uni-stuttgart.de

O. Kopp
e-mail: kopp@iaas.uni-stuttgart.de

F. Leymann
e-mail: leymann@iaas.uni-stuttgart.de

A. Bouguettaya et al. (eds.), *Advanced Web Services*,
DOI: 10.1007/978-1-4614-7535-4_22,

IT management or by outsourcing to external providers [18], which are both enabled and supported by cloud computing.

In recent years, cloud computing introduced a new way of using and offering IT software, platforms, and infrastructure services [21]. The "utility-like" offering of these services and flexible "pay-per-use" pricing are similar to how resources such as electricity and water are offered today [18]: Applications and other IT resources such as compute and storage must not be bought upfront and managed by the enterprise on its own, but can be simply requested when the respective functionality is actually needed—without dealing with the complexity of management, configuration, and maintenance. Therefore, enterprises move from a model of capital expenditure (CAPEX) to operational expenditure (OPEX) [1]. These approaches are expected to change the way how enterprises use and think about IT and may even relieve them from owning their own IT environment, which could be seen as the "next revolution in IT" [17]. Not only Gartner considers the efficient use of cloud computing as one of the key success factors for enterprises [12]. From a provider's perspective, automating the management of the offered services is of vital importance, because management and operation of IT is one of the biggest cost factors today—in terms of money and time. The ability to offer services which are elastic, self-serviced, rapidly provisioned, and priced based on actual consuption (pay-as-you-go) depends on the degree of automation of management. Thus, the management has to be organized in an industrial manner, i.e., shared throughout a number of customers and tenants [18].

Enterprise applications are typically complex composite applications, which consist of multiple individual components, each providing a clearly distinguishable piece of functionality. The functionality of the involved components is aggregated and orchestrated into a composite application providing a higher-level of functionality. Components typically have relationships to other components. For instance, a Web server component runs on an operating system component or an application connects to a database and external services. These composite enterprise applications typically rely on modular, component based architectures, which benefit from cloud technologies and properties such as elasticity, flexibility, scalability, and high availability [1, 5, 33, 34]. The different components involved need to be managed in terms of deployment, configuration, quality of service, and their communication to other components. The management becomes time-consuming and error-prone if the application structure, i.e., its components and relations, are not documented in a well-defined, machine-readable format. The management is often done manually by executing scripts or even completely manual work, which hinders automation, repeatability, and self-service.

To enable the creation of portable cloud applications and the automation of their deployment and management, the application's components, their relations, and management must be modeled in a portable, standardized, machine-readable format. This is where TOSCA—the Topology and Orchestration Specification for Cloud Applications [24]—proposes an XML-based modeling language tackling these issues by formalizing the application's structure as typed topology graph and capturing the management tasks in plans. In the scope of IT service management in general

and cloud computing in particular, three problems are addressed by TOSCA: (1) automated application deployment and management, (2) portability of applications and their management, and (3) interoperability and reuseability of components. An overview on TOSCA and how TOSCA addresses these challenges is provided in Sect. 22.2. After presenting the details of TOSCA in Sect. 22.3, we describe the supporting ecosystem in Sect. 22.4. In Sect. 22.5, we discuss how TOSCA achieves portability of composite cloud applications and what to do to improve portability of a TOSCA application. Finally, we close with our conclusions in Sect. 22.6.

## 22.2 Overview on TOSCA

TOSCA is an upcoming OASIS standard to describe composite (cloud) applications and their management. It provides a standardized, well-defined, portable, and modular exchange format for the structure of the application's components, the relationships among them, and their corresponding management functionalities. In this section, we provide a brief overview on the main concepts (Sect. 22.2.1) and which challenges in the are of cloud computing are addressed by TOSCA (Sect. 22.2.2).

### *22.2.1 Main Concepts of TOSCA*

TOSCA enables full automated deployment, termination, and further management functionalities, such as scaling or backing up applications through the combination of the two TOSCA main concepts: (1) Application topologies and (2) management plans. Application topologies provide a structural description of the application, the components it consists of and the relationships among them. Each node is accompanied with a list of operations it offers to manage itself. Thus, the topology is not only a description of the application's components and their relations, but also an explicit

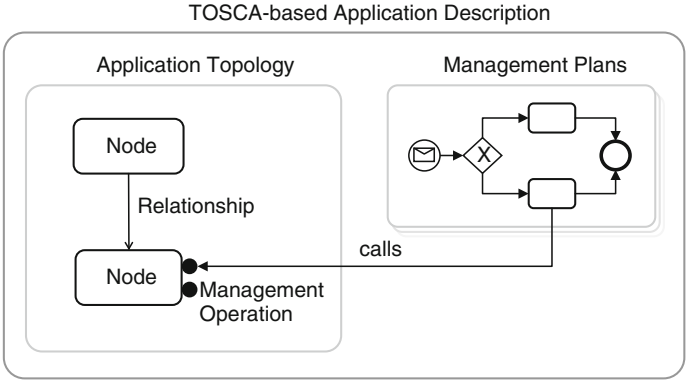


**Fig. 22.1** Relation of TOSCA concepts

declaration of its management capabilities. Management plans combine these management capabilities to create higher-level management tasks, which can then be executed fully automated to deploy, configure, manage, and operate the application. Figure 22.1 presents an abstract TOSCA-based application description, showing the two TOSCA main concepts and their relation: The application topology contains nodes, which are connected by relationships. Management plan are started by an external message and call management operations of the nodes in the topology.

### *22.2.2 Challenges Addressed by TOSCA*

In the area of cloud computing, there is a number of research challenges (cf. [9, 14]). This section discusses three major challenges and how TOSCA addresses them, namely ensure the portability of applications (Sect. 22.2.2.1), enable the automated management of applications (Sect. 22.2.2.2), and allow interoperability and re-usability of application components (Sect. 22.2.2.3).

#### 22.2.2.1 Automated Management

The management of applications plays an important role in enterprise IT (see Sect. 22.1). Especially external solutions impose the problem that the respective management knowledge must be acquired by each user, which usually results in slow and error prone manual management. TOSCA aims to formally capture the knowledge of the creator of the IT solution, who has all the knowledge of the solution's internals and proven best practices, in management plans [2]. These plans make the management of complex enterprise applications automated, repeatable, traceable, and less error prone. Users can easily fulfil management tasks without deep knowledge on how to manage the IT solution.

Management plans are portable between various environments and can be executed fully automated to support self-service management and rapid elasticity, both major requirements in cloud computing today. TOSCA enables these capabilities by using workflows to define management plans: Workflows provide the properties portability and fully automated execution [20].

#### 22.2.2.2 Portability of Applications

Current technologies and cloud providers usually define proprietary APIs to manage their services. Thus, moving an application based on these technologies to another provider requires rebuilding management functionalities and often even re-implementing parts of the application, if they use proprietary APIs offered only by the former provider. This is called vendor lock-in, which is the fact that the investment to switch from one provider to another provider is too expensive for a customer to be done economically. There is current research on technologies abstracting from

concrete APIs towards a unified interface for different APIs in order to reduce the problem of vendor lock-in, for example the work by Petcu et al. [29]. This may prevent vendor lock-in on the lower level but the user is then locked into this unified API, if, for instance, the unified API does not support certain providers. Research on this issue has already proposed solutions for supporting movability and migration of applications on a functional level, but especially application portability in terms their (automated) management is still a big problem [3, 19, 30]. TOSCA achieves portability by fomalizing the application topology as well as its management in a self-contained way. Each component defines and implements its management functionality in a portable way. How TOSCA achieves portability is discussed in detail in Sect. 22.5.

#### 22.2.2.3 Interoperability and Reusability of Application Components

TOSCA aims to enable the interoperability and reusability of application components such as Web servers, operating systems, virtual machines, and databases. These components are defined in a reusable manner by the developers, providers, or third parties together with their executables. Components of different providers do not stand on their own, as TOSCA enables combining them into new composite applications. Thus, TOSCA enables defining, building, and packaging the building blocks of an application in a completely self-contained manner. This allows a standardized way to reuse them in different applications.

## 22.3 TOSCA in Detail

TOSCA conceptually consists of two different parts: (1) The structural description of the application, called topology, and (2) the standardized description of the application's management by plans. These concepts are explained in Sects. 22.3.1 and 22.3.2 in detail. Instantiating the topology requires software files such as installables. In TOSCA, required software files, the topology and the management plans are packaged into one TOSCA archive. Section 22.3.3 describes this packaging. Section 22.3.4 describes an application topology example with a respective management plan for deploying the exemplary application.

### *22.3.1 TOSCA Application Topologies*

In TOSCA, the structure of a composite application is explicitly modeled by a colored graph called "application topology". Vertices represent the components of a composite application, edges represent different kinds of relations between these components. Relations may be, for example, one component is *hosted on*, *depends*

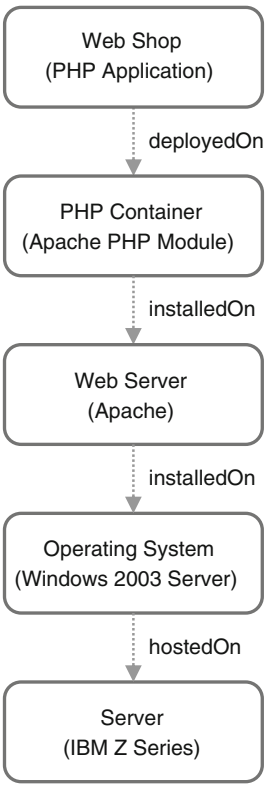


**Fig. 22.2** Conceptual layers of TOSCA-based applications

*on*, or *communicates with* another component. Figure 22.2 shows a PHP example topology delivering a PHP Web shop: A Windows 2003 Server operating system is hosted on an IBM server. Thereon, an Apache Web server is installed together with the PHP module on which the PHP application is deployed.

Vertices and edges in the topology may define additional properties, the management operations they offer, the artifacts required to run the component, or non-functional requirements. It is important to note that TOSCA does not only define the functional aspects of vertices and edges, i.e., providing a certain business functionality such as a Web service implementation, but in addition defines their management operations, for example, how to setup the component, establish a relation, deploy artifacts, scale-up, or backup. These management functionalities are reflected in the topology model and are the basis for the automated management concept of TOSCA, which is described in Sect. 22.3.2.

Figure 22.3 presents the structural elements of a Service Template: The Topology Template, Node Templates, Relationship Templates, and their types. The term *template* is used to indicate that it may be instantiated more than once and does not

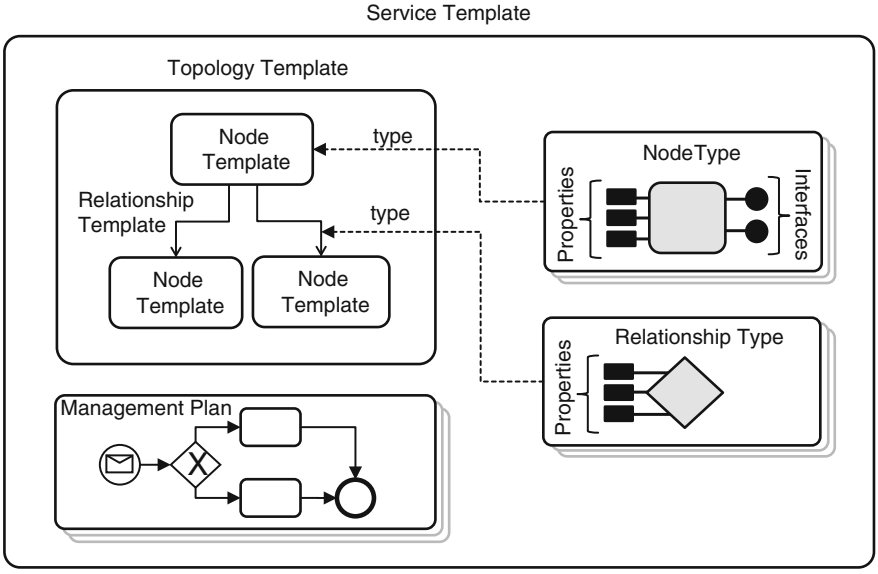


**Fig. 22.3** General structure of TOSCA service template (adapted from [24])

reflect the existing infrastructure. Each template is associated with a type, which defines the semantics of the template.

The layers of the topology are discussed in detail in Sect. 22.3.1.1. Section 22.3.1.2 details Node Types and Relationship Types. Node Templates and Relationship Templates are detailed in Sect. 22.3.1.3.

### 22.3.1.1 Conceptual Layers of TOSCA

To enable a clear understanding of TOSCA it is important to distinguish three conceptual layers as shown in Fig. 22.4: TOSCA defines a metamodel and exchange format for (1) types and (2) templates, which results in a third layer, the (3) instance layer, which depends on the TOSCA runtime (discussed in Sect. 22.4.2).

The metamodel layer defines Node Templates, which represent components, and Relationship Templates, representing the relations among the components, e.g., a *hosted on* relationship is used to define that a Web server component is hosted on an operating system component. These templates are typed with reusable types, i.e., Node Type for Node Templates and Relationship Type for Relationship Templates, respectively. These types are conceptually comparable to abstract classes in Java, whereas the templates are comparable to concrete classes extending these abstract classes.

The instance-layer represents the real instances of the components and relationships defined by templates. Thus, an instance of a Web server Node Template is a

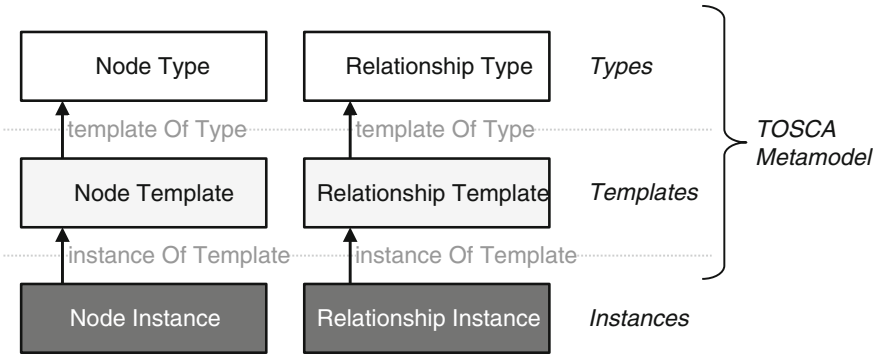


**Fig. 22.4** Conceptual layers of TOSCA-based applications

real existing instantiated Web server node, i.e., several instances may be created in "the real world".

#### 22.3.1.2 Node Types and Relationship Types

This section describes the information TOSCA offers to specify at Node Types and Relationship Types.

**Properties of Node Instances**. A node instance may have properties. Therefore, the respective Node Type references an XSD element (or type, [35]) declaring the schema for the actual property document. Properties are runtime information such as IP address, username, configuration, ports, and all other information required for deployment and management of the application. XSD supports lists and other complex structures, which basically allows to store all kind of information. In addition, XSD defines a strict schema for the resulting properties which can be used for validation. Templates are capable to define property defaults used at instantiation, for instance, the default port or username of the administrative interface. Support for reading and writing the properties is offered by a TOSCA container, which is explained in Sect. 22.4.2.

**Deployment Artifacts**. Deployment Artifacts specify the actual implementation of a Node Type. For example, an operating system type may have an image as Deployment Artifact and a Web server Node Type a Tomcat servlet container installable. During deployment of the application, the Deployment Artifacts are put onto the respective node. The concrete deployment procedure is not defined in the TOSCA topology. It is up to the management plans and management operations of the nodes.

**Lifecycle Definition**. Relationship instances and node instances may be in different states which aggregate the complex internal state of the instance. Example states are starting, running, stopping, and error. During runtime, each instance is in one of these states. The transition between the states is not described in a TOSCA model itself: The management plans and management operations trigger transitions

between the states. A TOSCA model defines, however, which states are possible in general: The possible states are defined as URIs in the respective types.

**Management and Implementation Artifacts**. Each hardware and software component offers explicit and implicit management capabilities. Explicit capabilities are startup parameters, configuration files, management interfaces, hardware buttons and so on. Implicit capabilities are descriptions of how to backup the application by copying a certain file, for instance. Offered operations include deployment operations, which are the deployment of an application on an application server or instantiating a new virtual machine, for instance. Further operations are offered for the *management* of an application, for example, upgrade, backup, scale up, and configure. A new concept introduced by TOSCA is that management capabilities of Node Types and Relationship Types are explicitly defined as REST-Service [10], WSDL-service [6], or scripts [26]. However, not all management capabilities of nodes are accessible that way. This is either because of technical reasons, such as incompatible protocols, or due to logical reasons, such as the operation being part of a composed operation. Management plans require standardized interface descriptions to be able to access management operations (see Sect. 22.3.2). Offering management capabilities not directly accessible by TOSCA plans is done by *Implementation Artifacts*. They are basically small management applications delivered together with the TOSCA application (cf. Sect. 22.3.3). Implementation Artifacts expose management capabilities of a Node Type via REST, WSDL, or script interfaces. Internally, they can do anything required to provide this functionality, including the invokation of management capabilities not compatible with TOSCA before. This ensures that all management operations are either offered by the node itself, an external service, a script, or an Implementation Artifact. Therefore, each Node Type or Relationship Type is self-contained with respect to its management. These basic management operations are then orchestrated by management plans into higher level management functionality spanning the whole application and, therefore, making the application self-contained with respect to its management.

**Policies**. TOSCA provides a generic container for attaching policies, for example, using WS-Policy [36] or the Rei Ontology [13], to nodes and relationships. The TOSCA specification does not state how and when policies are evaluated; it is only expected that a TOSCA-compliant environment respects these policies. Two examples for using policies are a connection (represented by a Relationship Template) with a policy that this connection must be encrypted and a server (represented by a Node Template) with a policy that a certain power consumption must not be exceeded during operation.

**Standardized and Derivation Types**. Node Types and Relationship Types can be refined through derivation [24, Sect. 4.3]. For instance, the Node Type `Tomcat` may be derived from Node Type `JavaApplicationServer` and the Relationship Type `JDBCConnection` may be derived from Relationship Type `connectsTo`. Each type may be derived from exactly one or no other type, which structures the types as trees.

Derivation enables groups of subject matter experts to standardize selected Node Types and Relationship Types. For instance, a generic virtual machine with its prop-

erties and operations may be offered as standardized Node Type. Vendors extend these standardized Node Types to offer their specific implementations. Besides offering standardized functionality, they might add proprietary functionality representing their competitive advantage. Offering different solutions under a common interface simplifies the creation of applications suitable for multiple environments and fosters portability.

From the ecosystem perspective (cf. Sect. 22.4), cloud and application providers may create and distribute libraries containing the Node Types and Relationship Types for their services and products to enable frictionless usage of them when building new applications.

#### 22.3.1.3 Node Templates and Relationship Templates

Node Templates and Relationship Templates, which are typed with exactly one Node Type or Relationship Type respectively, are composed to create the Topology Template of a TOSCA application. Templates define how the respective type is instantiated for use in the application. Templates allow defining the start values of the properties by specifying defaults for the properties. Deployment Artifacts, Implementation Artifacts, and policies may be overwritten and extended to adjust the types for the usage in the respective application, for example, an *Web Shop Application* Node Template of Node Type *PHP Application* defines a Deployment Artifact, which contains the respective PHP application files. Additionally, constraints may be put on properties to ensure that the properties fit to the overall application. For instance, the IP range of an application might be restricted to internal IPs of the company.

A Node Template may be instantiated multiple times. For instance, this is the case when there are multiple cluster nodes of an application or database cluster. Instead of requiring to put multiple Node Templates into the Topology Template, the properties `minInstances` and `maxInstances` are offered to set the range of the number of instances. This concept also supports Node Templates having a variable number of instances during runtime. For instance, the number of cluster nodes may be scaled up and down between 2 and 10. During runtime, for each instance of a Topology Template, each Node Template instance has its own identity and properties. This is obviously required, for example, to have multiple cluster nodes being equal besides the properties IP address and average load.

Grouping subgraphs of the Topology Template is possible by using Group Templates, which can be nested, but not overlapping. Group Templates can be used to separate nodes technically. For instance, a database cluster may be scaled independently of the other parts of the application. Either physically, e.g., by hosting all nodes of the database cluster in one dedicated data center, or logically, e.g., by assigning all database cluster nodes to a certain operations department.

### 22.3.2 TOSCA Management Plans

Section 22.3.1.2 showed how nodes and relationships offer their management capabilities. Based on the brief introduction to the concepts of management plans in Sect. 22.2, this section discusses details of the management plan concept. Management plans are not restricted to management operations of one node or relationship, but can also invoke a series of operations from different nodes, relationships, and also external services, including a human task interface [23]. Therefore, they are able to cover all kind of management tasks required by a TOSCA application.

Without TOSCA, the deployment and management of composite applications requires extensive, mostly manual, effort by the administrator, e.g., installing software on servers by using installation software provided on a DVD, logging onto servers updating applications, or creating backups. Each user has to learn on its own how to manage and operate the application, most of them making the same experiences and encounter the same difficulties, acquire management knowledge, and sometimes automate some management aspects through scripts. This is even more complicated for complex composite applications involving a large number of components by different vendors, which are combined to provide a certain business functionality. It requires significant knowledge and effort to provision, deploy, configure, manage, and, finally, terminate the components and their relationships [31]. TOSCA tackles these issues by enabling application developers and operators to capture reoccurring management tasks as management plans, which can be executed fully automated and thus decrease manual effort for application management and operation. Plans formalize the management knowledge and best practices implicitly for everyone to reuse. The management cost of applications described using TOSCA, including management plans, is significantly lower, especially because enterprises executing these management plans must not know all the details behind the management best practices. Figure 22.5 presents a simplified management plan used to deploy a PHP-based application: The plan installs an Apache Web server on a Windows operating system, installs the PHP module on that Web server, and finally deploys the PHP application thereon.

Automation of application management is a prerequisite to realize key cloud properties. Most important are self-service and rapid elasticity. Self-service means that a customer can instantiate and manage his application instance himself, e.g., add a new email account. Rapid elasticity enables on demand growing and shrinking of resources depending on the user needs, e.g., extending the storage of an email account. When going beyond cloud computing, automation has always been a key goal in IT service and application management. We want to stress that, despite its name, TOSCA is by no means restricted to cloud applications.

TOSCA does not introduce a new language for modeling and executing plans. Instead, TOSCA includes plans by using existing workflow languages such as the Business Process Model and Notation (BPMN, [25]) or the Business Process Execution Language (BPEL, [22]). By using workflow technology to automate management tasks, TOSCA benefits from all the capabilities and properties of workflow languages and workflow execution environments. These properties include parallel

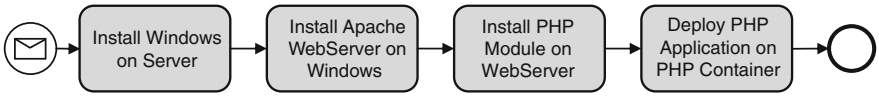


**Fig. 22.5** Example management plan for deploying a PHP-based application

execution, monitoring, compensation, recovery, auditing, and tracing functionalities [20]. In addition, established workflow languages and environments also support human tasks to include manual work into the management plans. A typical example for a manual task, which cannot be executed automatically without human intervention, is installing physical infrastructure such as servers, network components, or storage as basis for virtualized environments. Using workflow technology moves the low level management tasks onto business processes level and makes them accessible to people or software not aware of the technical management details.

To ensure portability of management plans, TOSCA relies on the portability of standardized workflow languages such as BPEL and BPMN. The recommended workflow language for TOSCA management plans is BPMN. However, TOSCA allows plans to be defined in any workflow language providing clear execution semantics required for automated execution. Unfortunately, not all existing languages are suitable as many process modeling languages focus either on modeling or on execution [16, 28, 32].

#### 22.3.2.1 Scripts and Plans

Today, many tasks in systems and operations management of applications are already automated by using scripts. These scripts are typically—often manually—copied to the target system on which they are executed. In comparison to plans, these scripts can be seen as microflows: small isolated pieces of work which can be executed fast and do not require transactional support, called micro script stream without transactions by Leymann and Roller [20]. In TOSCA, scripts are used for small management tasks such as setting up databases on a single component, whereas plans are used for large management tasks typically involving multiple components such as deploying a Tomcat servlet container on a Linux operating system followed by the configuration of both components. Of course, both concepts can be combined to provide the ability of specifying management operations on different layers of granularity. Then, plans represent workflows orchestrating several microflows represented by scripts.

A main benefit of this separation of concerns is provided by the combination of both concepts: Plans can use scripts to do more fine grained work directly on the target components while all problems of script handling such as data passing from and to other tasks, error handling, compensation, and recovery can be done by the workflow technology, which is on a much more coarse grained layer. Thus, wrapping script handling through workflow technology increases the level of abstraction for the operators as they do not have to deal with the deep technical details of script

handling [15]. This is in line with the programming-in-the-large idea by DeRemer and Kron [8], which is applied by the workflow technology, too [20].

#### 22.3.2.2 Plan Usage of the Application Topology Model

Management plans may inspect the application topology to retrieve nodes and relationships in order to manage them. This may be necessary for flexible plans not developed for one specific application topology to manage, but for multiple different topologies consisting of similar structures, or at least similar components. Thus, the plan needs information about the concrete structure of the considered topology to find the respective components and relationships therein the plan is supposed to manage. One example is a large topology consisting of multiple software stacks and a plan which updates the operating system components of each stack.

Management plans are executed on external workflow engines and may do various kinds of manipulations on the node and relationship instances. During operation, the state of node and relationship instances may change. For instance, the patch level of an operating system changes after installing a patch on an operating system node. To transfer this state information between different management plans, they need to store this information externally of the workflow context to make them accessible by various stakeholders. Therefore, the possible properties of nodes are explicitly defined by a schema to standardize accessing them. This information is included in the application topology model (see Sect. 22.3.1.2) and plans may read and write these service instance state information [2].

### *22.3.3 Packaging*

A TOSCA Service Template defines application topologies and corresponding management plans. The physical associated files such as Implementation Artifacts and Deployment Artifacts, scripts, or XML schema files are packaged together with the actual Service Template into a TOSCA archive, called "Cloud Service Archive (CSAR)" [24, Sect. 3.3]. This standardized archive format provides a way to package applications fully self-contained, with all required management functionalities into one single file used for installing the application. Thus, the archive can be seen as single installable for complex composite applications including their management. A TOSCA archive can be deployed on a TOSCA runtime environment (see Sect. 22.4.2) which is responsible for installing the application package, i.e., processing the archive. TOSCA archives follow a standardized format ensuring portability between different TOSCA runtime environments and thus provide an exchange format for complex composite applications including their management functionalities. Figure 22.6 shows the conceptual structure of a TOSCA archive.

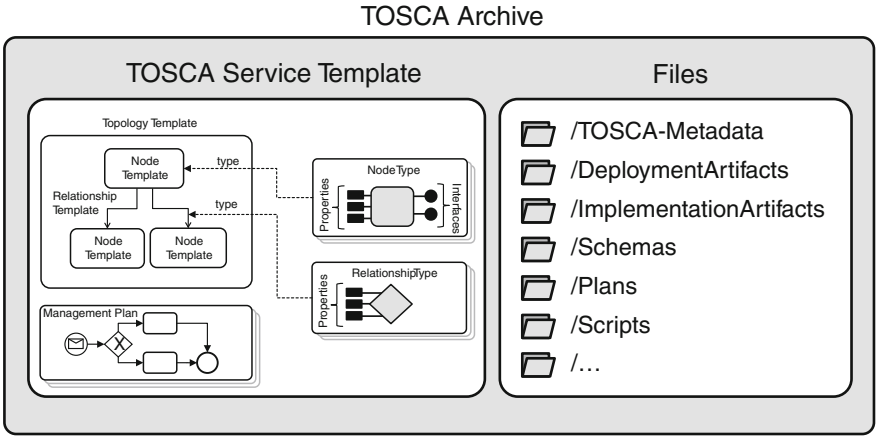


**Fig. 22.6** Conceptual structure of TOSCA archives

### *22.3.4 TOSCA-Based Example Application*

In this section we describe a TOSCA application example and a corresponding build plan, which deploys and instantiates the application. The example implements an online Web shop which consists of two functionally different software stacks: The first stack provides a Web-based GUI for the Web shop application, the second stack provides product information data stored in a MySQL database accessible through a REST API which is called by the Web-based GUI.

The complete application topology is presented in Fig. 22.7. The stack providing the GUI is presented on the left side of the figure. The infrastructure layer of this stack consists of a *Server* Node Template of Node Type IBM Z Series. This represents a physical server node. Thereon runs a Windows operating system represented by an Operating System Node Template of Node Type Windows 2003 Server. On this OS, a Web server Node Template of Node Type Apache runs with an installed PHP Module, which in turn is represented as a PHP Container Node Template of Node Type Apache PHP Module. This container is able to run PHP-based applications. The Web Shop Node Template of Node Type PHP Application implements the GUI of the Web shop software and is hosted on the PHP Container.

The infrastructure layer of the second topology stack providing product data for the Web-based GUI consists of a Virtual Server Node Template of Node Type AWS EC2 Server. This is an Infrastructure-as-a-Service (IaaS, [21]) offering provided by Amazon.[1] On this virtualized infrastructure an operating system of Node Type Windows 7 runs in a VM which is represented by an Operating System VM Node Template. On this operating system, there are two components hosted on: A Servlet Container Node Template of Node Type Tomcat and the Product Database Node

[1] http://aws.amazon.com/ec2/

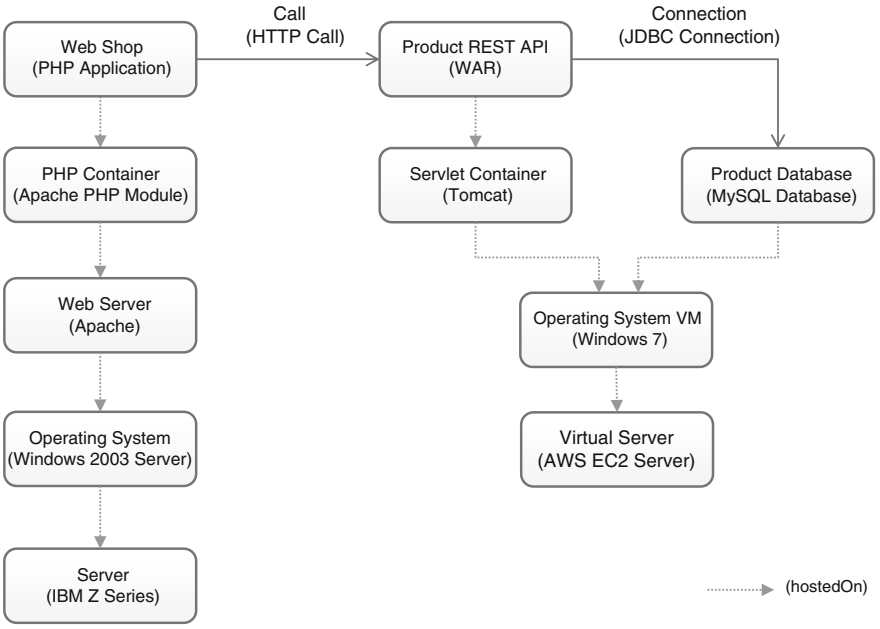


**Fig. 22.7** Example TOSCA application topology

Template of Node Type MySQL Database, which represents the database in which the product data are stored. On the servlet container, there is a REST API providing access to the product data stored in the database. The API is implemented as Java application, which is deployed as Web Archive (WAR) file. This Java application is hosted on the Tomcat servlet container and is represented as Product REST API Node Template of Node Type WAR hosted on the servlet container node. For simplification reasons, we modeled all runsIn, deployedOn, and installedIn relations as hostedOn relation, which is the parent Relationship Type for all these Relationship Types.

The Build Plan shown in Fig. 22.8 is responsible for deploying both software stacks. We simplified the plan in some points to reduce the degree of complexity. For instance, handling of security issues (e.g., password generation and storage) are hidden. BPMN supports parallel execution of tasks. Therefore, the two software stacks are deployed in parallel. First, the deployment of the Web-based GUI is described. The first activity installs the Windows 2003 Server operating system on a physical server whose IP-address is given by the input message of the Build Plan. Thus, for executing the plan, the IP-address of the server has to be known by the operator and written into the input message. After the OS is installed, the subsequent activity configures the operating system such as setting the correct firewall rules. After that, the Apache Web server is installed on the Windows 2003 operating system, the PHP Module is installed on the Apache Web server and the PHP Application is deployed into the PHP Container. The second software stack is deployed in the

parallel branch. First, an activity acquires a Windows 7 VM on Amazon EC2. The required credentials are contained in the input message of the plan, i.e., the operator has to know the credentials and put them into the input message for executing the plan. After the operating system VM is provisioned, installing the Tomcat servlet container followed by the deployment of the WAR file on it are done in parallel with installing the MySQL database server. The REST API Java application has to know the endpoint of the database. The last activity in this parallel branch sets this endpoint to the Java application. After both application stacks are deployed, the Web-based GUI needs to know the endpoint of the REST API. This is done by the last activity of the workflow which sets this endpoint to the PHP application.

## 22.4 Supporting Ecosystem

TOSCA specifies an exchange format for application topologies and their management plans. The TOSCA standard does not live on its own, but requires a supporting ecosystem to exploit its full potential. This section presents three key parts which are important for a viable TOSCA ecosystem: (1) Topologies and their management plans have to be modeled properly (Sect. 22.4.1). (2) After a TOSCA model is created, it has to be interpreted by a TOSCA-compliant runtime environment to enable automatic deployment and management (Sect. 22.4.2). (3) Finally, Sect. 22.4.3 presents how an application marketplace could benefit from the new possibilities enabled by TOSCA.

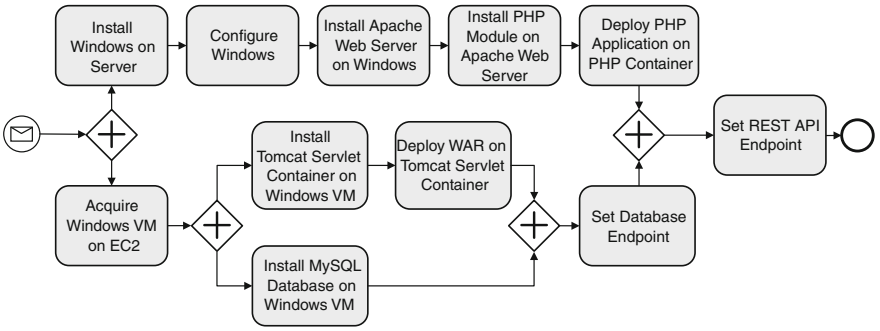


**Fig. 22.8** Build plan for the example application

### *22.4.1 Modeling Tool Support*

As TOSCA's representation format is XML, modeling TOSCA-based applications and their management plans, typically also having a textual XML representation, may be time-consuming when using text editors only. XML editors may be helpful

as TOSCA defines a schema which can be used by the editor to provide features such as auto-completion and tag-proposals. These tools might help avoiding syntactic errors and improve the speed of creating models compared to pure text editors. Nevertheless, they are still uncomfortable as manual typing is error prone and semantical dependencies are hard to recognize textually by the user or the tool.

Therefore, graphical modeling tools tailored towards TOSCA could reduce the effort significantly as topologies as well as plans can be represented visually easily. For example, modeling topologies can be enriched with graphical details, such as icons for nodes, which supports a faster recognition of the semantics. Thus, semantic errors, such as wrong hostedOn-relationships, can be recognized faster by the user. In addition, the speed of modeling increases noticeably as a lot of unnecessary typing is spared. As modeling of topologies as well as modeling of plans can be done graphically, some modeling tools combine both activities. This is an important advantage as bringing together modeling of topologies and corresponding management plans might be cumbersome—especially for annoying frequently reoccurring tasks such as copying IDs, creating boilerplate code, and so on. Enhancing a BPMN modeling tool to provide a tight integration with TOSCA has been presented by Kopp et al. [15]. TOSCA-tailored graphical modeling tools also may support reusability of Node Types by providing existing Node Types in a palette for dragging them into the topology, for example. Automated management of a variety of artifacts and exporting them into a TOSCA archive as described in Sect. 22.3.3 additionally reduces the complexity and assists the user.

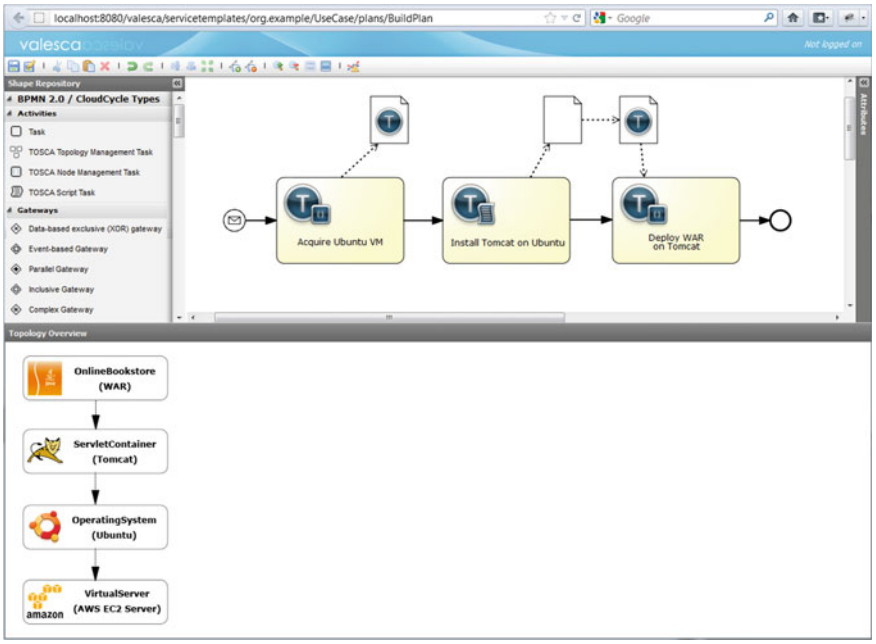


**Fig. 22.9** Screenshot of Valesca

One open source implementation of a TOSCA-tailored graphical tool combining the modeling of TOSCA application topologies and associated management plans is "Valesca".[2] It implements "Vino4TOSCA" [4], a visual notation for TOSCA topologies. Valesca uses the Signavio Core Components,[3] which are the commercially-supported enhancements of Oryx [7]. Figure 22.9 shows a screenshot. Valesca supports all the advantages mentioned above and is provided under the Apache 2.0 license.

### *22.4.2 TOSCA Container*

To use all the features of TOSCA—especially automation of application management—a TOSCA-compliant runtime is required. Without such a container, TOSCA could be used as pure exchange format and manually operated according to the definitions in the Service Template. However, a bare-minimum TOSCA container stores and serves the files contained in the TOSCA archive, installs and operates Implementation Artifacts and Management Plans, and manages the instance data of the application: The container is the glue between these functionalities. During modeling the management plans are written without knowing the exact location of the Implementation Artifacts, only referencing the abstract service description (port type and operation in case of WSDL services). Implementation Artifacts are deployed by the TOSCA container to the respective runtime, for example, Java Web services to an Apache Tomcat known and managed by the TOSCA container. Knowing the runtime and the location of the deployed Implementation Artifact, the TOSCA container is able to set the location information when deploying the Management Plans onto a workflow engine. The container is also responsible for managing the properties assigned to node and relationship instances. Therefore, the container offers a standardized API which may be used by Implementation Artifacts and the workflow engine to work on the properties.

To increase convenience, other functionalities, such as a user interface for starting the management plans, identity management, integrated monitoring and auditing, can be supported by the container, which exceeds the scope of this chapter.

### *22.4.3 Marketplace and Catalog*

TOSCA enables new business models in terms of application exchange, offering, and trading. Due to the fact that TOSCA applications are portable between different TOSCA-compliant providers, moving application flexibly between providers avoids vendor lock-in: Customers have the ability to choose applications independent from

[2] http://www.cloudcycle.org/valesca/

[3] http://code.google.com/p/signavio-core-components/

the cloud provider which hosts the application later on. This enables a new kind of marketplaces for trading manageable and portable applications which can be hosted by any TOSCA-compliant provider, as shown in Fig. 22.10. Inside enterprises, the TOSCA ecosystem enables offering self-service catalogs which allow flexible and rapid deployment of business applications.

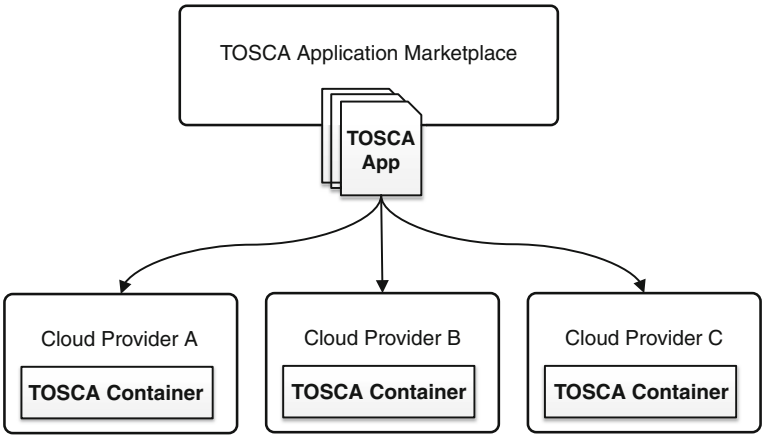


**Fig. 22.10** Sketch of an TOSCA application marketplace

## 22.5 Portability

As portability is a central goal of TOSCA, this section discusses in detail how TOSCA supports portability (Sects. 22.5.1 and 22.5.2) and propose how modelers can increase the portability of their TOSCA applications (Sect. 22.5.3).

### *22.5.1 Portability of Applications*

TOSCA addresses the portability of application descriptions and their management, not the portability of the application components themselves. That means, TOSCA does not make Deployment Artifacts portable, e.g., to run a .net application on Apache Tomcat or to migrate one flavor of relational database system to another.

The application topology may have some prerequisites concerning the environment it is deployed on. For instance, it might require an external service such as Amazon EC2 or inhouse infrastructure such as VMware. However, there are ways to abstract from concrete providers and increase portability between different environments, for example, by using software such as Deltacloud,[4] which unifies the APIs of

[4] http://deltacloud.apache.org/

different cloud infrastructure providers into a common interface or by using a generic and standardized virtual machine Node Type as described in Sect. 22.3.1.2, which can be bound to different implementations. The remaining parts of the application topology are built on top of these lower-level infrastructure and, therefore, are basically self-contained inside this application topology. Thus, they only depend on the lower-level infrastructure components. If these are portable, the whole application is portable. The application topology's main purpose is to be an information source and description of the component's management aspects for the management plans. By concerning the existence of standardized Node Types as lower-level components and that higher-level components can depend on these lower-level ones, we conclude that the application topology can be modeled in a portable way. Based on this we must have a look on the portability of management plans.

### *22.5.2 Portability of Management*

Management plans are written in certain workflow languages and it is the TOSCA container's responsibility to execute them on a compatible workflow engine. Therefore, TOSCA container support for the workflow language is the first precondition for TOSCA portability, which is, however, softened to some extend by the fact that BPMN is the recommended workflow language in TOSCA. Management plans are orchestrations of three types of services: (1) External services, which are portable, because services are, by definition, accessible from everywhere [6], (2) management operations offered by Node Types and Relationship Types, and (3) APIs of the TOSCA container, for example, to access the instance data of the application instance. As discussed in Sect. 22.3.1.2, the management operations can be provided as Implementation Artifacts, whose execution is also the TOSCA container's responsibility. The API of the TOSCA container will be standardized. Therefore, support for the language of the Implementation Artifact by the TOSCA container is the second precondition for TOSCA portability.

Consequently, the portability of TOSCA applications only fails if the type of management plan or Implementation Artifact is not known and supported by the TOSCA container. We want to highlight that both of them, management plans and Implementation Artifacts, represent the management part of TOSCA and are not the actual application. Moreover, we expect most TOSCA containers to provide some kind of extensibility mechanism to add plugins supporting additional plan types and additional Implementation Artifact types. A couple of basic types will then be offered by most of the TOSCA containers, which will provide a solid basis for portable TOSCA applications.

### *22.5.3 Improving Portability of TOSCA Applications*

The conclusions of the previous two sections lead to the following recommendations on how to increase the portability of a TOSCA application: For Implementation Artifacts, the goal is to provide them in programming languages, which are widely supported by TOSCA containers. Due to the fact that Implementation Artifacts are bound to Node Types and Relationship Types, they are widely reused so it may be worth the effort to do multiple implementations. Management plans are tied to the actual application and, therefore, their level of reusability is lower than reusability for Implementation Artifacts. Providing them in multiple workflow languages would also increase their portability, but doing this manually might not be worth the effort. Fortunately, there are existing approaches to transform workflow languages [32], for example, transforming BPMN to BPEL by using the approach by Ouyang et al. [27]. In addition, due to the fact that BPMN is the recommended workflow language for management plans, a wide variety of TOSCA containers will presumably support BPMN.

## 22.6 Conclusions

This chapter presented the upcoming OASIS standard Topology and Orchestration Specification for Cloud Applications (TOSCA). We highlighted that TOSCA distinguishes between the application topology and management plans. The application topology declares the components of the application and their relationships as a graph. We discussed that each vertex in the graph represents a Node Template, which has a Node Type and is instantiated as node instance. The management plans invoke management operations on these node instances.

TOSCA is a standard not providing any software and, therefore, requires an ecosystem. We gave a short overview on possible modeling tool support and runtime support. TOSCA packages may be distributed directly by a software vendor or available through dedicated marketplaces.

At the point of writing this chapter, the TOSCA specification was not finally released. However, we expect no fundamental changes going beyond what we described. One can follow the current development of the TOSCA specification on the OASIS TC website[5] and the development of the OpenTOSCA ecosystem of the University of Stuttgart on the OpenTOSCA website[6].

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

[5] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

[6] http://www.opentosca.org

## References


1. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: Above the Clouds: A Berkeley View of Cloud Computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley (2009)
2. Binz, T., Breiter, G., Leymann, F., Spatzier, T.: Portable Cloud Services Using TOSCA. IEEE Internet Computing **16**(03), 80–85 (2012). doi:10.1109/MIC.2012.43
3. Binz, T., Leymann, F., Schumm, D.: CMotion: A Framework for Migration of Applications into and between Clouds. In: Proceedings of the 2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA). IEEE Computer Society Conference Publishing Services (2011). doi:10.1109/SOCA.2011.6166250
4. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Schumm, D.: Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA. In: Proceedings of the 20th International Conference on Cooperative Information Systems (CoopIS 2012), Lecture Notes in Computer Science. Springer-Verlag (2012) doi:10.1007/978-3-642-33606-5_25
5. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems **25**(6), 599–616 (2009). doi:10.1016/j.future.2008.12.001
6. Curbera, F., Leymann, F., Storey, T., Ferguson, D., Weerawarana, S.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall PTR (2005).
7. Decker, G., Overdick, H., Weske, M.: Oryx - An Open Modeling Platform for the BPM Community. In: Proceedings of the 6th International Conference on Business Process Management (2008). doi:10.1007/978-3-540-85758-7_29
8. DeRemer, F., Kron, H.: Programming-in-the-Large Versus Programming-in-the-Small. Software Engineering, IEEE Transactions on **SE-2**(2), 80–86 (1976). doi:10.1109/TSE.1976.233534
9. Dillon, T., Wu, C., Chang, E.: Cloud Computing: Issues and Challenges. In: Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on, pp. 27–33 (2010). doi:10.1109/AINA.2010.187
10. Fielding, R.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California (2000)
11. Garbani, J., Mendel, T., Radcliffe, E.: The Writing on ITs Complexity Wall (2010). Forrester Research
12. Gartner: Gartner Identifies the Top 10 Strategic Technologies for 2011 (2010). Press Release
13. Kagal, L.: Rei Ontology Specifications, Ver 2.0 (2012). http://www.csee.umbc.edu/~lkagal1/rei/
14. Khajeh-Hosseini, A., Sommerville, I., I, S.: Research Challenges for Enterprise Cloud Computing. Tech. rep., LSCITS (2010)
15. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications. In: 4th International Workshop on the Business Process Model and Notation. Springer (2012) doi:10.1007/978-3-642-33155-8_4
16. 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)
17. Leymann, F.: Cloud Computing: The Next Revolution in IT. In: Proc. 52th Photogrammetric Week, pp. 3–12. Wichmann Verlag (2009)
18. Leymann, F.: Cloud Computing. it - Information Technology **53**(4) (2011) doi:10.1524/itit.2011.9070
19. Leymann, F., Fehling, C., Mietzner, R., Nowak, A., Dustdar, S.: Moving Applications to the Cloud: An Approach based on Application Model Enrichment. International Journal of Cooperative Information Systems (IJCIS) **20**(3), 307–356 (2011). doi:10.1142/S0218843011002250

20. Leymann, F., Roller, D.: Production Workflow - Concepts and Techniques. Prentice Hall PTR (2000)
21. Mell, P., Grance, T.: Cloud Computing Definition. National Institute of Standards and Technology (2009)
22. OASIS: Web Services Business Process Execution Language Version 2.0 - OASIS Standard (2007). https://www.oasis-open.org/committees/wsbpel/
23. OASIS: WS-BPEL Extension for People (BPEL4People) Specification Version 1.1 (2010). http://docs.oasis-open.org/bpel4people/bpel4people-1.1.html
24. OASIS: Topology and Orchestration Specification for Cloud Applications Version 1.0 Committee Specification Draft 03 (2012). http://docs.oasis-open.org/tosca/TOSCA/v1.0/csd03/TOSCA-v1.0-csd03.html
25. OMG: Business Process Model and Notation (BPMN) Version 2.0 (2011). http://www.omg.org/spec/BPMN/2.0/. OMG Document Number: formal/2011-01-03
26. Ousterhout, J.: Scripting: Higher level programming for the 21st century. Computer 31(3), 23–30 (1998)
27. Ouyang, C., Dumas, M., ter Hofstede, A., van der Aalst, W.: Pattern-based Translation of BPMN Process Models to BPEL Services. International Journal of Web Services Research **5**(1), Idea Group Publishing (2008)
28. Palmer, N.: Understanding the BPMN-XPDL-BPEL Value Chain. Business Integration Journal **November/December**, 54–55 (2006)
29. Petcu, D., Craciun, C., Rak, M.: Towards a Cross Platform Cloud API - Components for Cloud Federation. In: CLOSER. SciTePress (2011)
30. Petcu, D., Macariu, G., Panica, S., Crciun, C.: Portable Cloud applications–From theory to practice. Future Generation Computer Systems (2012). doi:10.1016/j.future.2012.01.009
31. Rus, I., Lindvall, M.: Knowledge management in software engineering. Software, IEEE 19(3), 26–38 (2002)
32. Stein, S., Kühne, S., Ivanov, K.: Business to IT Transformations Revisited. In: 1st International Workshop on Model-Driven Engineering for Business Process Management (2008). doi:10.1007/978-3-642-00328-8_18
33. Varia, J.: Architecting for the Cloud: Best Practices. Tech. rep., Amazon (2010). http://media.amazonwebservice.com/AWS_Cloud_Best_Practices.pdf
34. Varia, J.: Cloud Architectures. Tech. rep., Amazon (2010). http://jineshvaria.s3.amazonaws.com/public/cloudarchitectures-varia.pdf
35. W3C: XML Schema Part 1: Structures Second Edition (2004). http://www.w3.org/TR/xmlschema-1/
36. W3C: Web Services Policy 1.5 - Framework (2007). http://www.w3.org/TR/ws-policy/