Institute of Architecture of Application Systems

Standards-based Function Shipping – How to use TOSCA for Shipping and Executing Data Analytics Software in Remote Manufacturing Environments

Michael Zimmermann, Uwe Breitenbücher, Michael Falkenthal,
Frank Leymann, Karoline Saatkamp

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{lastname}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{Zimmermann2017_FunctionShipping,
  author    = {Michael Zimmermann and Uwe Breitenb{\"u}cher and Michael
               Falkenthal and Frank Leymann and Karoline Saatkamp},
  title     = {Standards-based Function Shipping – How to use TOSCA for
               Shipping and Executing Data Analytics Software in Remote
               Manufacturing Environments},
  booktitle = {Proceedings of the 2017 IEEE 21st International Enterprise
               Distributed Object Computing Conference (EDOC 2017)},
  year      = {2017},
  pages     = {50--60},
  doi       = {10.1109/EDOC.2017.16},
  publisher = {IEEE Computer Society}
}
```


Universität Stuttgart
Germany

Standards-based Function Shipping – How to use TOSCA for Shipping and Executing Data Analytics Software in Remote Manufacturing Environments

Michael Zimmermann, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, Karoline Saatkamp
Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany
Email: [lastname]@iaas.uni-stuttgart.de

***Abstract*—The increasing amount of gathered sensor data in Industry 4.0 allows comprehensive data analysis software that creates value-adding opportunities. As companies often cannot implement such software by themselves and as they typically don't want to give their data to external scientists, they commission them to build the required software in order to execute it locally. However, installing, configuring, and running complex third party software on another company's infrastructure and linking them to local data sources challenges the responsible administrators due to an immense technical complexity. Moreover, standards-based approaches for automation are missing. In this paper, we present three TOSCA-based deployment modelling approaches for function shipping that enable modelling data analysis software in a way that enables (i) its automated deployment and execution in a remote, foreign IT infrastructure including (ii) the wiring with the data sources that need to be processed in this environment. We validate the practical feasibility of the presented modelling approaches by a case study from the domain of manufacturing, which is based on the open-source TOSCA ecosystem *OpenTOSCA*, which provides a modelling tool, a runtime, as well as a self-service portal for TOSCA.**

I. Introduction

The increasing amount of gathered sensor data in the fields of the Internet of Things [1] and Industry 4.0 [2] allows comprehensive data analyses that create value-adding opportunities, for example, predictive maintenance of cyber-physical manufacturing systems. Unfortunately, the analysis of gathered data typically requires complex and domain-specific algorithms, which often cannot be developed by a company itself without employing expensive experts. Therefore, many companies commission external data scientists and third parties to analyze their data, for example, for implementing machine learning techniques [3]. However, this collaboration quickly becomes complex due to political issues as the data to be analyzed is often of vital importance for the company and must not be distributed to their competitors. As a result, third party data analysis software often has to be executed in the local infrastructure of the company in order to prevent that critical data about their business leaves their sovereignty [4].

One problem of this approach is that installing, configuring, and running complex third party software on another company's infrastructure challenges the responsible administrators as detailed expertise is required: (i) the middleware required by the software needs to be provisioned, (ii) the software has to be deployed, and (iii) the software needs to be configured in order to obtain the required data. Thus, the responsible administrators not only need to understand the required middleware but also the software itself, which is a serious problem if the software and its deployment are complex. Even if the developers of the software support the administrators, different physical hardware, employed virtualization technologies, and application platforms lead to the same kind of deployment problems. Moreover, a manual deployment approach is error-prone, time-consuming, and, therefore, not efficient [5]. Thus, *shipping* specialized software into a remote and foreign IT infrastructure of another company is a highly complex challenge that comes with technical as well as organizational deployment difficulties.

In this paper, we tackle these issues. We present three *deployment modelling approaches* that enable describing data analysis software in a way that enables (i) its automated deployment and execution in a remote, foreign IT infrastructure including (ii) the wiring with the data sources that need to be processed in this environment. We call such software that is shipped to the data it has to process a *function*. The presented approaches tackle different levels of arrangement between the third party developer and the ordering company. We first describe these approaches in an abstract manner followed by a detailed explanation how the TOSCA standard [6]–[8] can be used to realize them. Thereby, deploying and running third party software that processes locally stored data becomes significantly simplified as no experts are required on the side of the company to deploy, configure, and run the analysis software. We validate the practical feasibility of the presented deployment modelling approaches by a case study from the domain of manufacturing, which is based on the open-source TOSCA ecosystem *OpenTOSCA* [9]–[11], which provides a modelling tool, a runtime, as well as a self-service portal.

The remainder of this paper is structured as follows. In Sect. II, we present a motivation that is used throughout this paper to explain the presented concepts. Afterwards, we briefly describe the TOSCA standard in Sect. III. In Sect. IV, we present different approaches for modelling shippable functions in the form of automatically deployable models based on the TOSCA standard. To validate the practical feasibility, we discuss in Sect. V how the TOSCA ecosystem *OpenTOSCA* can be used to automatically deploy shipped functions. In Sect. VI, we present related work in terms of function shipping. Section VII concludes the paper and discusses future work.

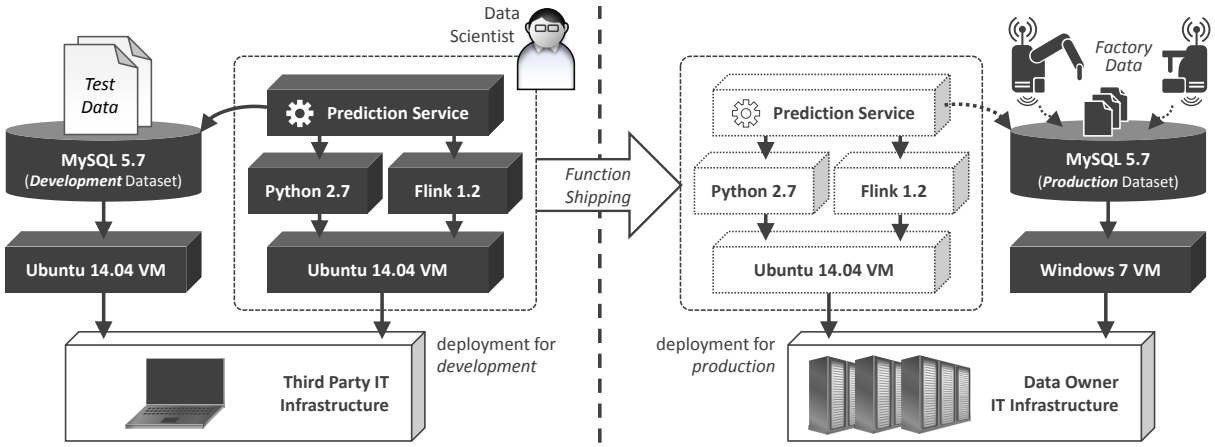

Fig. 1. Motivating Scenario: Analytics functionality is developed on the basis of an analytics stack deyployed on a development notebook (left) and shipped for production to an on-premise IT infrastructure in a manufacturing environment (right).

II. Background & Motivation

Developing services for analyzing data in the context of Industry 4.0 is often an activity that requires immense expertise about analytics techniques and algorithms as well as the structure and characteristics of the data to be analyzed [12]. Further, also appropriate analytics platforms and analytics stacks have to be configured and adapted in order to be utilized for running such applications. Since analytics applications are typically not part of the core business of manufacturing companies, their development and implementation is often sourced out to external data scientist, who specialize in analytics aspects like the afore-mentioned. Thus, in such situations manufacturing companies are faced with the problem to (i) enable the development of valuable analytics algorithms by sharing analysis data about their production processes while (ii) preserving data security due to the fact that this data can contain business critical information that has to be protected and must not leave the company. Finally, also (iii) protecting information about their employed IT infrastructure is often an obstacle for efficiently cooperating with data scientists.

To resolve this dilemma, companies tend to only provide parts of such analysis data, often in aggregated and anonymized form to externals, besides closing strict non-disclosure agreements with the data scientists. Thereby, external data scientists are enabled to develop analytics algorithms in IT environments that are not controlled by the manufacturing companies. However, the developed analytics software requires access to the actual data set from the manufacturing environment. Thus, the analytics software and often the entire analytics stack need to be shipped to the company's IT environment to process the real and continuously generated data there due to the policy that the data must not leave the data-owning company.

This typical Industry 4.0 scenario is abstractly depicted in Figure 1, which illustrates the connection of gathered machine-data with the functionality to analyze them. In this scenario, analytics algorithms are developed and provided by a *Data Scientist* via a *Prediction Service* as depicted on the left side. While developing the Prediction Service, the analytics stack based on Apache *Flink* and further *Python* libraries are both hosted on an *Ubuntu* virtual machine in the IT environment of the data scientist. Flink is an analytics platform enabling the integration of data sources, such as a *MySQL* database containing a *Development Dataset*, along with capabilities for batch as well as stream processing. This might be a development notebook as depicted in Figure 1, a private cloud infrastructure, some legacy system that can run the analytics stack for development purposes but could also be a public cloud offering capable of running the depicted stack. On the right side of Figure 1, a MySQL database running on a virtual machine hosted on OpenStack is shown, which is used to store analysis data generated by the machines, which are operated in a factory of the manufacturing company. The scenario assumes, that the OpenStack is operated within the manufacturing environment due to data security and privacy reasons as described above.

Since the machine-data often must not leave the company, the Prediction Service needs to be shipped to the data for going into production, i.e., it has to be deployed and executed close to the data in the IT environment of the manufacturing company. Therefore, particular components or even the whole analytics stack, as depicted in Figure 1, have to be firstly shipped to the IT environment of the data owner and then, secondly, be wired with the data source containing the *Production Dataset*. This can be generalized and grasped as *function shipping* to deliver the Prediction Service to the data to be analyzed.

Such function shipping scenarios can strongly profit from automated deployment mechanisms, which have evolved in the domain of cloud computing over the last years. Independently from the actually used deployment technology, the wiring of the analytics functionality, i.e., the analytics algorithm or the whole analytics stack, with the data to be analyzed needs to be specified and modelled. Thereby, the amount of information that can be added to such deployment models varies based on circumstances, as for example, the mutual confidence of the data scientist and the data owner. Thus, the amount of information the data owner is willing to share with externals fundamentally

influences the design and configuration of the deployment model. One option for the data owner is to share all required information with the data scientist. Such required information are, for example, how the externally developed analytics stack can be wired with the locally operated production database. However, while this means that endpoint addresses, usernames, passwords, further required keys and certificates, etc. can be integrated aprior in the deployment model, sharing this information with third parties can cause critical security issues and is, therefore, not suitable for many companies.

To prevent from such issues, the deployment model can also be parametrized in a way that these data are only provided at deployment time of the analytics stack in the manufacturing environment. This approach ensures that, on the one hand, a deployment model can be designed, which can be reused for the automated deployment of the analytics stack in the environment of the data owner and, on the other hand, that the security-relevant data required for the provisioning and wiring does not have to be shared with externals but can be provided by the data owner themself when starting the provisioning.

Finally, a deployment model of the analytics stack can also be elaborated in a way that only requirements are defined, which specify mandatory capabilities the components in the manufacturing environment have to fulfill or provide in order to be wired with the analytics stack. For example, the analytics stack can specify the requirement that the data to be analyzed has to be served by a specific database system and must be available in a specific schema. This option provides the opportunity that the deployment model does not have to be completely modelled by the third-party developer. This means that the data scientist only models the analytics software and all requirements that have to be fulfilled during deployment and overhand this model to the data owner. Then the data owner can complete the model by adding components, such as the MySQL database containing the production dataset as depicted in Figure 1, along with other required information, such as username and password required for the deployment.

These three exemplary scenarios illustrate that a wide spectrum of realizations of a deployment model is possible to ship analytics functionality close to the data to be processed. In this paper, we present how this can be realized using TOSCA.

III. The TOSCA Standard

In this section, we briefly explain the *Topology and Orchestration Specification for Cloud Applications (TOSCA)* [6], [8], [13], which is an OASIS standard for automating the deployment and management of cloud applications. We use TOSCA in the next section to explain how the presented modelling approaches for function shipping can be realized.

TOSCA enables the description of the components of an application, their dependencies, as well as required infrastructure resources in a portable manner. The structure of a cloud application can be modelled as *Topology Template*. A Topology Template is a directed graph consisting of typed nodes and weighted edges. The nodes represent the components of the application, for example, a MySQL-Database, an Apache Tomcat, or an Ubuntu virtual machine, and are called *Node Templates*. The edges represent the relationships between these components and are called *Relationship Templates*. Relationship Templates enable to model, e.g., that a PHP Application is "*hostedOn*" an Apache 2 Web Server. For reusability purposes, the semantics of Node Templates and Relationship Templates are defined by *Node Types* and *Relationship Types*. Node Types define, for example, *Properties*, e.g., the username and password of a database or the port of a web server. Moreover, Node Types may also define parameterizable *Management Operations* that can be invoked to manage instances of this Node Type. For example, a cloud provider or hypervisor Node Type usually provides a management operation to create ("*createVM*") and terminate ("*terminateVM*") a virtual machine.

Figure 2 depicts an example Topology Template. Here, a virtual machine with an Ubuntu 14.04 operating system is hosted on OpenStack. On this virtual machine, Python 2.7 as well as Apache Flink 1.2 is installed. A prediction service implemented using Python is hosted on the Apache Flink processing framework. Furthermore, it has dependencies on the Python 2.7 installation. The Topology Template also shows some exemplary modelled Properties, such as the port of the Flink instance or username and password of OpenStack. Thus, this model is a deployment model for our motivation scenario.

A. Artifacts, Management Plans, and CSARs

TOSCA defines two kinds of artifacts: (i) *Implementation Artifacts (IAs)* and (ii) *Deployment Artifacts (DAs)*. Implementation Artifacts implement the Management Operations defined by Node Types and can be implemented using various technologies. For example, as a web service packaged as WAR file, a simple shell script, or by using a configuration management technology such as Ansible [14] or Chef [15]. Deployment Artifacts, on the other side, implement the business functionality of a Node Template. For example, a Python file implementing the Prediction Service is a Deployment Artifact.

The creation and termination of instances of the modelled application is done by *TOSCA Runtimes*. They either consume a Topology Template *declaratively* and derive the actions to be executed on their own, or invoke *Management Plans* that are associated with the Topology Template. A Management Plan is an automatically executable workflow model that *imperatively* specifies the management operations to be executed for executing a certain management functionality, for example, to provision a new instance of the application or to scale it. TOSCA allows to use any process modelling language, but recommends to use workflow languages such as the *Business Process Execution Language (BPEL)* [16] or the *Business Process Model and Notation (BPMN)* [17]. Moreover, there is a BPMN extension called *BPMN4TOSCA* [18], [19] that is explicitly tailored for describing TOSCA-based deployment and management plans. Thus, using imperative Management Plans, arbitrary management functionality can be realized.

For packaging all the mentioned artifacts, type definitions, templates, plans, and additional files, TOSCA defines a portable and self-contained packaging format, which is called *Cloud*

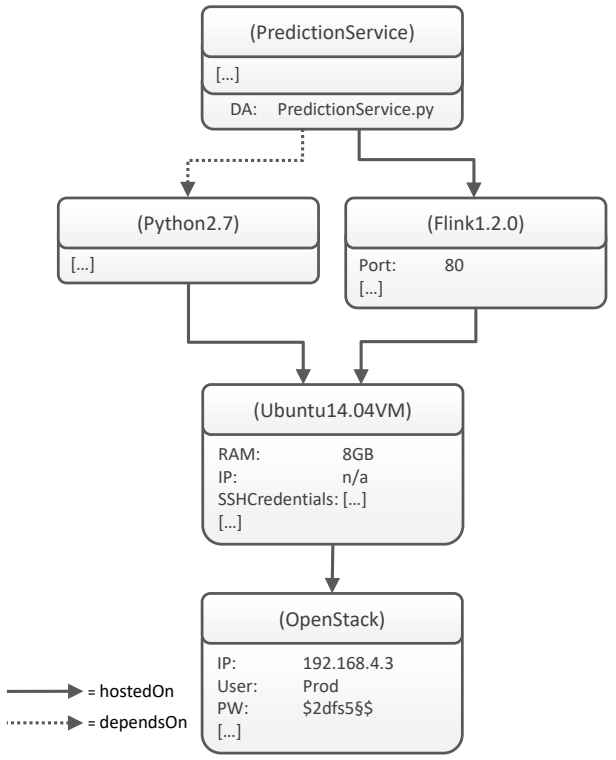

Fig. 2. Simplified TOSCA Topology Template describing the deployment of the motivation scenario.

Service ARchive (CSAR). Thus, such a CSAR contains all TOSCA model files as well as further software required to enable the automated provisioning and management of the modelled application. Furthermore, Cloud Service Archives can be executed using a standard-compliant TOSCA Runtime.

B. Using TOSCA for Function Shipping

TOSCA's capability to package the entire deployment model including all required artifacts into a self-contained CSAR optimally supports third-party developments in terms of deployment automation: The CSAR just has to be executed by a TOSCA Runtime to deploy the application, thus, neither experts nor the developers are required to deploy the software.

However, there is no guideline how the TOSCA standard can be used to ship functions into another environment that have to process data sources available in this environment. For example, if a CSAR contains an application stack including all files, how to configure the deployment in a way that the implemented function finds the data to be analyzed? Manually adapting the CSAR by an administrator of the company for injecting this information is not appropriate as it requires knowledge about the application, which we have to avoid. Moreover, some companies have tight arrangements with their third-party developers, which possibly enables hard-coding such data source endpoints directly in the TOSCA model. However, also this breaks if the endpoint of the data source changes. Thus, a guideline how to model the application in a way that it is able to access the data source to be processed is missing.

IV. Deployment Modelling Approaches for Shipping and Executing Functions

In this section, we present three deployment modelling approaches that enable to ship data analysis software into arbitrary environments including their fully automated deployment and wiring with local data sources that have to be processed by the shipped software. The three approaches are: (i) *Complete Deployment Model*, (ii) *Configurable Deployment Model*, and (iii) *Variable Deployment Model*. Furthermore, each modelling approach consists of a declarative and an imperative variant. Therefore, in the following we provide a conceptual guideline on how the TOSCA standard can be used for shipping functions into remote manufacturing environments, also under consideration of the respective existing requirements.

In the following, a *declarative deployment model* denotes a model that specifies the components, their relationships, and all properties of the desired deployment. A TOSCA Topology Template is an example for such a model. In contrast, an *imperative deployment model* is a process model that specifies tasks to be executed as well as their order. An SH script or a workflow model belongs to this kind, thus, also TOSCA Management Plans are imperative deployment models. The presentation of the modelling approaches is structured as follows: We first present the main idea of the modelling approach in an abstract manner, regarding the declarative and the imperative variant of this approach. Afterwards, we describe forces that have to be tackled when applying it. Subsequently, we summarize the traits of the modelling approach. Finally, we show how the approach can be realized using TOSCA.

A. Complete Deployment Model

The idea of this deployment modelling approach is to specify every detail of the deployment in the model so that the deployment model does not have to be parametrized.

1) Declarative Deployment Model: If a declarative deployment model is used, this means that (i) every component, (ii) every relationship between the components, as well as (iii) every property of components and relationships are exactly specified in the deployment model. For example, if the function to be shipped is implemented as Java application that has to be executed within a Java Virtual Machine that runs on an Ubuntu operating system hosted on an OpenStack cloud management system, every detail must be specified in the declarative model. This means that, for example, the exact version of the Ubuntu operating system must be specified as well as the IP-address, username, and password of the OpenStack in order to enable instantiating the required virtual machine. Moreover, the model must exactly specify the data to be processed by the software. For example, the model must contain a database component including all properties required to retrieve data from this database, e. g., IP-address, username, password, name of the table, etc. In Figure Fig. 3, the motivating scenario is illustrated as declarative deployment model fulfilling these characteristics.

2) Imperative Deployment Model: If an imperative deployment model is used, i. e., a process model such as a workflow, this means that every task of the deployment must be

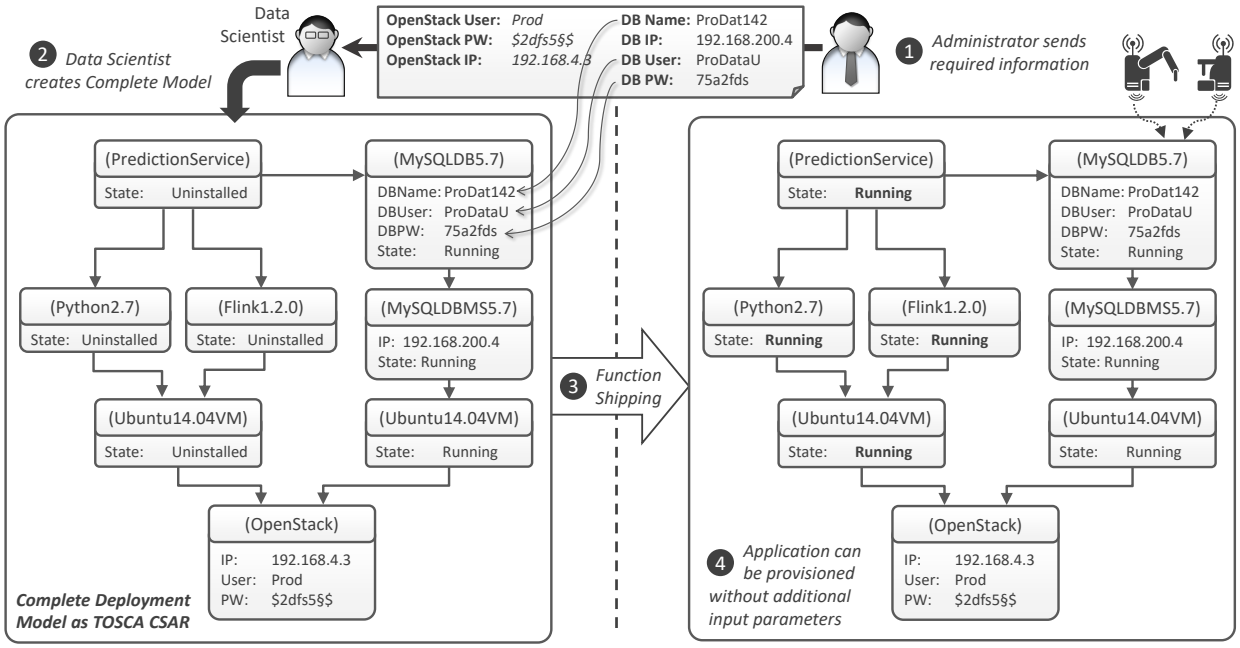

Fig. 3. Overview of the Complete Deployment Model approach using TOSCA illustrating the motivation scenario.

modelled in a self-contained manner. Thus, the process model has no input parameters and can be executed without any further information. For example, if the imperative deployment model is implemented as script, the script must contain every information: The IP-address of the OpenStack as well as username and password. In addition, the script has to invoke the respective OpenStack API to create a virtual machine, has to log into this virtual machine via SSH to install the JVM, Python, etc. Moreover, the script has to exactly know (i) where the data to be processed is stored and (ii) how to access it.

3) Forces: A Complete Deployment Model, either imperative or declarative, requires all information of the desired deployment. Thus, as the software, for example, the function, has to be shipped into a remote environment, every required information must be contained in the model. For example, the IP-address, username, and password of the OpenStack must be known by the third party developers for enabling them to build such a model. However, this means that the company that commissions these external developers must trust that they do not misuse this information. Thus, the Complete Deployment Model approach is only suited for trustworthy collaborations.

4) Result: By using a Complete Deployment Model, the third-party developer can ship the application in a fully self-contained manner. For executing the deployment, no expertise is required as all information is completely contained in the model. Thus, the company that wants to use this software only has to execute the corresponding model, no parametrization or further information is required. However, a Complete Deployment Model is tightly coupled to the target infrastructure. Thus, e.g., if the username or password of the OpenStack account to be used or its IP change, the deployment model has to be adapted. As deployment quickly become very complex, this probably requires the third party developer and causes additional cost.

5) Complete Deployment Model in TOSCA: The Complete Deployment Model approach is natively supported by TOSCA in both flavors: TOSCA Topology Templates can be used to model the deployment declaratively while properties of Node Templates and Relationship Templates can be used to precisely specify all required information and desired configurations as shown in Fig. 3. The illustrated Complete Deployment Model shows the analytics stack on the left side, containing Flink and the Prediction Service, as well as the MySQL database stack on the right side. Both stacks are connected using a Relationship Template. Thus, the relation between the Prediction Service and the MySQL database can be specified, for example, by defining a "connectsTo" relation. Furthermore, all specified Properties, such as the IP-address of the OpenStack, the IP-address of the MySQL database management system, and username and password of the MySQL database are already filled in the Complete Deployment Model. Thus, the application can be provisioned without adding additional parameters, as it is shown on the right side of Fig. 3. In contrast, as TOSCA supports Management Plans, imperative deployment models can be created as well and packaged as a CSAR. Of course, these Management Plans also need to contain all information required for the provisioning and wiring of the application with the database, such IP-addresses, usernames, and passwords.

B. Configurable Deployment Model

The idea of this deployment model approach is to model the deployment of all components, but without specifying all details required for the provisioning and wiring. Therefore, resulting in a deployment model that needs to be parametrized.

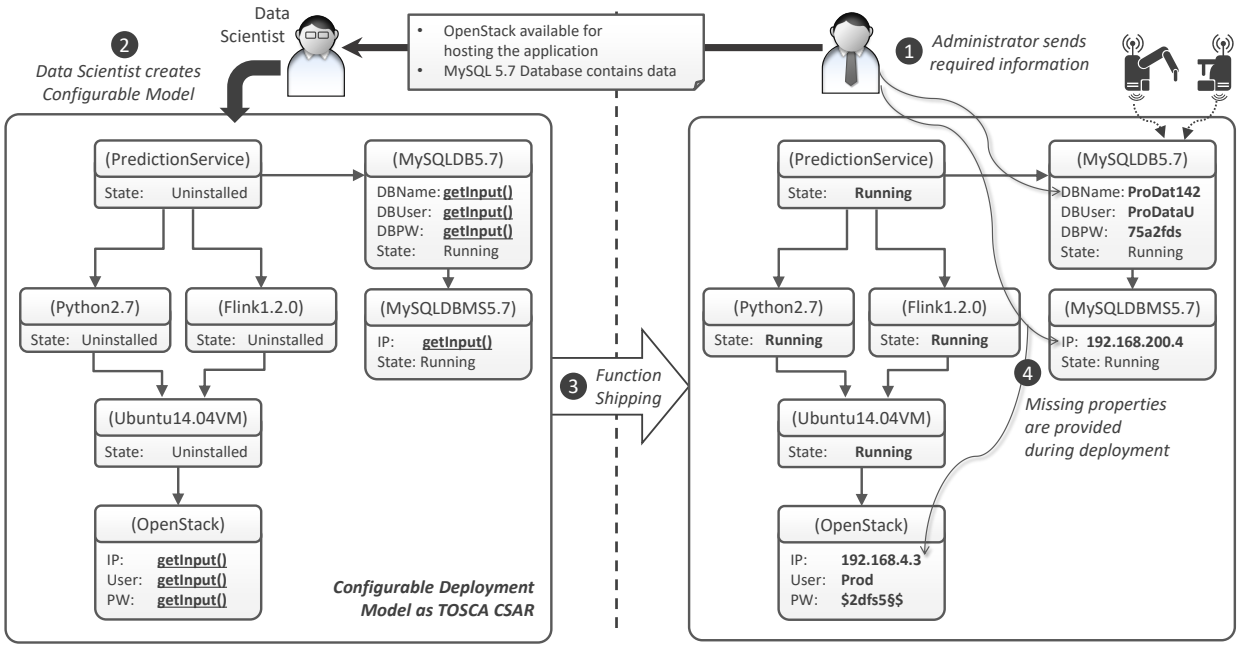

Fig. 4. Overview of the Configurable Deployment Model approach using TOSCA illustrating the motivation scenario.

1) Declarative Deployment Model: When using a declarative deployment model, similar to the first approach, (i) the components of the application and (ii) the relationships between these components are specified in the deployment model. However, in this approach, not all properties are specified in the model. For example, instead of modelling a MySQL database with all its properties, such as IP-address, username, and password, only the components themselves without specifying the concrete property values are modelled. These properties are filled during the deployment time of the application either (i) by the person initiating the deployment or (ii) the runtime environment. Furthermore, since only the information to connect to the database are required, it is not necessary to define, for example, on which version of the Ubuntu operating system the database is hosted. In Figure Fig. 4 a declarative deployment model of this approach is depicted, illustrating the motivating scenario.

2) Imperative Deployment Model: If an imperative deployment model is used for this approach, this model, for example, a workflow, must be parameterized by defining input parameters. In order to get the missing information, these input parameters can be filled either (i) by the user initiating the provisioning or (ii) the runtime by using available instance data. For example, before starting the provisioning of the application, the administrator can enter the missing information such as the IP-address of the locally operated OpenStack or the username and password for accessing the available MySQL database.

3) Forces: A Configurable Deployment Model does not require all information of the desired deployment. Instead, it needs to be designed in a way to be is configurable and reusable. For example, only the components of the application, such as a MySQL5.7 database, must be specified in the deployment model. Additional information, e.g., the IP-address or the username and password required for accessing the database, are set during the deployment time. Thus, in this approach information about the available hosting infrastructure, e.g., OpenStack, and already running components, e.g., a MySQL5.7 database need to be exchanged between the third party and the company as shown in Fig. 4 in order to enable external developers creating a Configurable Deployment Model. But in contrast to the first approach, no detailed or credential information about the target IT environment are required. Thus, the Configurable Deployment Model approach fits best, when concrete information about the available target IT infrastructure can be shared with third parties, but detailed information, e.g., IP-addresses, usernames, and passwords should be kept secret.

4) Result: This approach allows the creation of a flexible and reusable deployment model. Furthermore, for creating the Configurable Deployment Model, no credential information about the target environment infrastructure, such as IP-addresses or usernames and passwords are required. For this deployment model approach, only the available components in the target environment need to be known. However, for executing the deployment model, additional information are required.

5) Configurable Deployment Model in TOSCA: TOSCA supports both, the declarative as well as the imperative processing of a Configurable Deployment Model. Within the TOSCA Topology Template, the properties of Node Templates and Relationship Templates can be left open or marked with "getInput()" [13] as shown in Fig. 4. Thus, the TOSCA Runtime Environment operated in the target IT environment understands that additional information, such as the IP-address of OpenStack or the username and password of the MySQL database are required and requests the user to enter them. For example, in the shown Configurable Deployment Model, for privacy

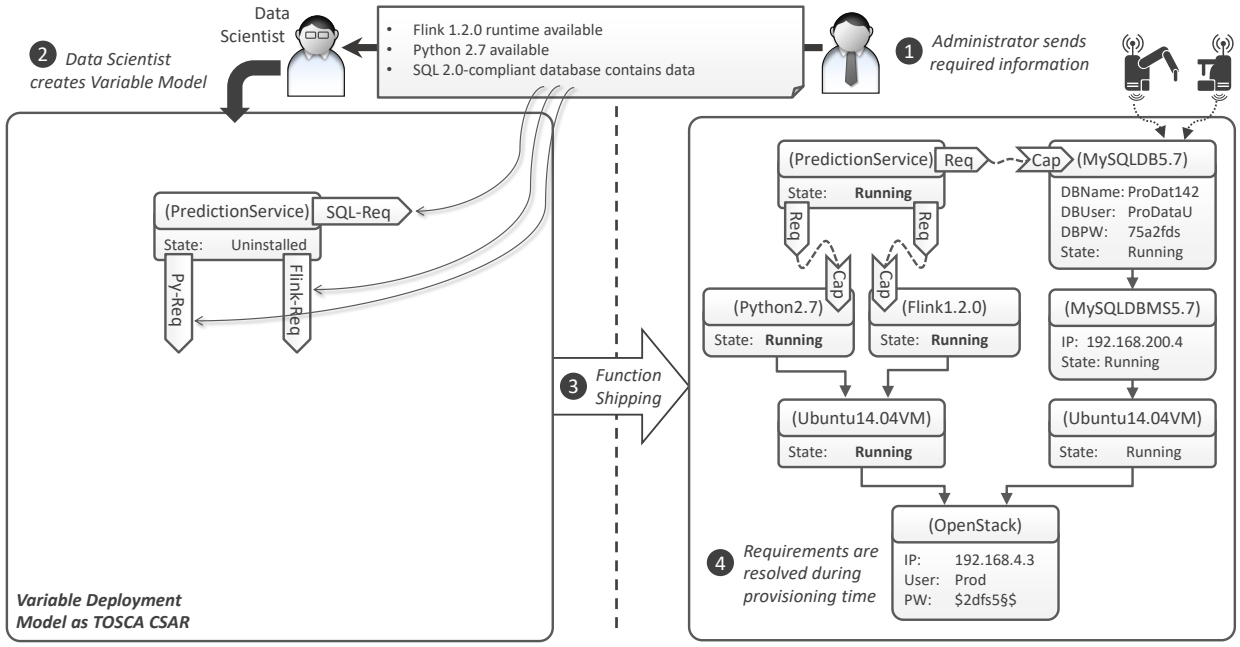

Fig. 5. Overview of the Variable Deployment Model approach using TOSCA illustrating the motivation scenario.

reasons, all Properties of the components operated in the target IT environment, such as the OpenStack or the MySQL database are marked with "getInput()". Thus, the third party developer can create the deployment model, although parameters required for the provisioning are missing. Furthermore, as shown in the figure illustrating the motivation scenario, for example, the information on which operating system the database is hosted is not required for provisioning the analytics stack and connecting the Prediction Service to the database. Thus, such information do not need to be shared between the data-owning company and the developer of the analytics software. However, since the Properties are defined within the Node Types, the Node Types of the components available in the target IT environment, such as the MySQL 5.7 database or the OpenStack, are required for modelling the Configurable Deployment Model. For the imperative variant of the Configurable Deployment Model, input parameters can be specified for Management Plans in order to make them parametrizable. Thus, before the Management Plans are executed in the target environment, the missing parameters can be entered, e.g., by the administrator.

C. Variable Deployment Model

The idea of this deployment modelling approach is that as little as possible needs to be modelled in order to get a Variable Deployment Model. For example, only the function that should be shipped and its requirements are modelled in this approach.

1) Declarative Deployment Model: In case of a Variable Deployment Model, only the components that should be shipped may be specified in the deployment model. Moreover, in order to be able to provision the shipped components later on, requirements of this components need to be specified in the model. For example, if a prediction service needs to be hosted on Apache Flink, has a dependency to Python, and requires to be connected to a MySQL database, these requirements need to be specified in the deployment model. The specified requirements are used to find components providing matching capabilities in the target IT environment. In Fig. 5 the motivating scenario is illustrated as TOSCA-based declarative deployment model fulfilling the described characteristics.

2) Imperative Deployment Model: If the imperative deployment model variant is used for the Variable Deployment Model approach, this model can be shipped as incomplete model. Before the provisioning, this incomplete deployment model needs to be completed by the data-owning company itself. For example, if a workflow is used as deployment model, abstract activities, for instance "createVM", "installFlink", and "deployApp" can be defined. However, these abstract activities need to be implemented manually by the data-owning company.

3) Forces: The Variable Deployment Model only contains the components to be shipped as well as requirements for the provisioning of them. Thus, no detailed information about the target infrastructure are required. For example, the developer of the prediction service only needs the information that an Apache Flink instance is available, but neither on which operating system it is hosted nor if it is running on a local OpenStack or any other hypervisor or cloud provider. Thus, the Variable Deployment Model approach is suitable if as little as possible information should be shared with other parties.

4) Result: By using the Variable Deployment Model approach, the third party developer can create a deployment model without the requirement of getting all information about the target IT infrastructure. Therefore, the development of components as well as the modelling of the deployment model can be done mostly independently of the target IT environment,

resulting in a highly flexible and reusable deployment model. Only some basic information need to be shared for this modelling approach. However, in order that the requirements and capabilities match, this approach requires that the definition of requirements and capabilities is done in a consistent manner.

5) Variable Deployment Model in TOSCA: For realizing the declarative approach, TOSCA allows the definition of *Requirements* and *Capabilities* for Node Types [6], [7]. Again, *Requirement Types* and *Capability Types* can be modelled as reusable entities. Requirements and Capabilities can be used to define, for example, that a component requires a feature that is provided by another component. For example, the Prediction Service shown in Fig. 5, specify that it requires a SQL endpoint for accessing the stored data. Furthermore, it also requires a Python installation and a Flink runtime. Thus, components fulfilling these defined Requirements by providing matching Capabilities can be found in the target IT environment. For example, if the Prediction Service specifies a "SQLEndpoint"-Requirement, the TOSCA Runtime can find the MySQL database providing a "SQLEndpoint"-Capability. Furthermore, for Relationship Types a valid source and a valid target can be defined. As a valid source, a Requirement Type can be specified and as a valid target, a Capability Type can be specified. Thus, a concrete Relationship Type for connecting the components having Requirements and providing matching Capabilities can be defined indirectly. For our motivation scenario, for example, a "SQLConnection" can be indirectly specified between the Prediction Service and the MySQL database by defining the "SQLEndpoint"-Requirement as valid source and the "SQLEndpoint"-Capability as valid target of the Relationship Type. As shown in Fig. 5, using this approach, only the function that needs to be shipped has to be modelled in the Variable Deployment Model. Thus, the Variable Deployment Model is an incomplete deployment model, that can be completed by a TOSCA Runtime in the target environment. The declarative approach can be realized by defining Management Plans containing abstract activities. However, since there is no approach for automatically completing an imperative Variable Deployment Model available yet, these abstract activities need to be implemented manually by the data-owning company.

V. Validation & Prototype

In this section, we present our implemented prototype supporting the different function shipping approaches presented in the previous section. The prototype validates the practical feasibility of our proposed function shipping modelling concepts. Furthermore, in this section, we show how our implemented prototype can execute the deployment models automatically.

The prototype is based on the *OpenTOSCA Ecosystem*, a standards-based TOSCA Runtime Environment. OpenTOSCA consists of three main components: (i) *Winery*[1] [10], a graphical tool for modelling TOSCA models, (ii) *OpenTOSCA container*[2] [9], the provisioning and management engine for TOSCA models, and (iii) the graphical self-service portal *Vinothek*[3] [11] for starting the provisioning. The source code of these three components can be obtained from GitHub.

Winery enables the graphical modelling of TOSCA Topology Templates of the application that should be deployed. Furthermore, the topology as well as all required files can be packaged into a CSAR. Also, the matching of requirements and capabilities and the resulting auto-completion of the topology model can be done using Winery. The OpenTOSCA container is able to process the resulting CSAR, interprets the contained TOSCA model, deploys required Implementation Artifacts and Management Plans and finally provisions the modelled application. Furthermore, the OpenTOSCA container is able to process declarative as well as imperative deployment models. For the end user, the self-service portal Vinothek is provided. Vinothek is a graphical user interface allowing to choose an available application and start the provisioning of it. If required, the user initiating the provisioning of an application can insert missing information required for the provisioning here.

The OpenTOSCA container is implemented using the programming language Java 1.7. Furthermore, it is based on the OSGi Framework Equinox[4], which is a Java-based runtime for building modular and dynamic applications. Management Plans implemented using BPEL are deployed by the OpenTOSCA container on a local workflow engine namely the WSO2 Business Process Server (BPS)[5] to make them executable. Winery is implemented using the programming language Java 1.8. Winery is available as Web Application Archive (WAR) and thus, can be easily deployed on a web container such as Tomcat. From an architectural perspective, Winery is split into the graphical front end for modeling the topologies, called Topology Modeler and Winery Repository, which is basically the back end of Winery. The graphical web interface of Winery is implemented using Javascript and Java Server Pages (JSP). Moreover, the self-service portal Vinothek is also implemented using Java Server Pages and packaged as WAR.

Since all these components of the OpenTOSCA ecosystem are available as open-source implementations, with our prototype, we implemented an open-source end-to-end toolchain for TOSCA-based cloud and IoT applications. The prototype enables the modelling, provisioning, management, orchestration, and communication of these applications, as well as the shipping of functions into remote manufacturing IT environments, by using the proposed function shipping modelling approaches. Besides the TOSCA Specification [6] and the TOSCA Primer [7], explaining the concepts of the specification in more detail, there is also a TOSCA Simple Profile [13] available. While in the Specification and the Primer XML is used for rendering TOSCA, in the Simple Profile YAML is used instead. Also both version provide different concepts. The concept of Management Plans, for example, is only specified in the TOSCA Specification. The concept of getting properties by

[1]https://github.com/OpenTOSCA/winery
[2]https://github.com/OpenTOSCA/container
[3]https://github.com/OpenTOSCA/vinothek
[4]http://www.eclipse.org/equinox/
[5]http://wso2.com/products/business-process-server/

the user at deployment time by using a "getInput" function originates from the Simple Profile. However, our prototype fully supports the XML TOSCA Specification as well as a small selected subset of concepts from the Simple Profile in YAML. How the different approaches are supported by our prototypical implementation is explained in the following.

A. Complete Deployment Model

In the Complete Deployment Model all required information for provisioning the modelled function as well as its required components are described. Since a plan generator [20] is part of our prototype, we are able to generate a Management Plan based on the declarative TOSCA Topology Template. As workflow language we use the Business Process Execution Language (BPEL). Furthermore, since our prototype is able to deploy BPEL workflows and execute them, of course, it is also possible to create imperative deployment models manually using BPEL, which contain all required information for the provisioning.

B. Configurable Deployment Model

The Configurable Deployment Model does not contain all information required for provisioning the modelled function and its components. For example, the IP-address and the username and password for the OpenStack running in the target IT environment may be omitted in the deployment model. In case of the declarative deployment model, these information are defined within the properties of Node Templates. The plan generator is able to detect the missing properties and to generate a parametrized provisioning plan. Thus, if properties are kept empty, the prototype requests the user who is initiating the provisioning to enter missing information. Furthermore, the prototype is also able to use existing instance data to find missing information. In case of an imperative deployment model, a parametrized BPEL workflow can be used. Again, before executing the workflow, the missing information can be entered by the user or are determined by the TOSCA Runtime.

C. Variable Deployment Model

The Variable Deployment Model only defines the function that should be shipped as well as its requirements. In this case the specified requirements are resolved by the OpenTOSCA Runtime by finding existing components specifying matching capabilities, therefore, resulting in a completed TOSCA Topology Template. Based on this, the plan generator is used to generate an executable provisioning plan again. But, since requirements and capabilities need to be matched in this approach, our prototype does not support the imperative variant of this modelling approach yet.

D. Comparison of the 3 Deployment Models and Discussion

In order to discuss the characteristics of the three presented deployment models, in this section we compare these three function shipping modelling alternatives and describe their advantages as well as their impact on the deployment regarding aspects, such as time, complexity, and the required skill of the persons responsible for the deployment.

Since in the Variable Deployment Model not the entire stack but only the function that should be shipped as well as its requirements need to be modelled, the Variable Deployment Model is the fastest modelling variant regarding the time required for creating the model. In both, the Complete Deployment Model as well as the Configurable Deployment Model the entire stack needs to be modelled, thus, there is no big difference between these two alternatives regarding the time required for modelling. However, the Complete Deployment Model contains all required information for the deployment, thus it can be directly provisioned in the target environment, whereas in the Configurable Deployment Model required information, for example IP-addresses, usernames, or passwords, need to be inserted by the administrator first. Moreover, in case of the Variable Deployment Model, the components matching the specified requirements additionally need to be determined first.

Regarding the complexity of the different presented deployment model alternatives, again, the Variable Deployment Model only requires the function as well its requirements to be modelled. However, in order that required components can be determined during provisioning time, the specified requirements need to be defined in a way that they can be matched with capabilities provided by components already available in the target environment. Thus, in order to use this function shipping modelling alternative the definition of requirements and capabilities need to be managed thoroughly. The declarative variants of the Complete Deployment Model as well as the Configurable Deployment Model is pretty much straight-forward since all components as well as their relations are known. However, if not only software should be provisioned in the target environment, but also, for example, a device needs to be physically connected to a machine first, this can only be modelled using an imperative deployment model. Thus, using the imperative variants, the complexity of the deployment models can vary significantly depending on the respective use case. For example, if data of a machine should be analyzed, but that machine is missing any possibility for transferring the data, some more complex tasks are required. One possibility to handle this problem would be to physically connect a separate device, such as a Raspberry Pi with the machine and configure it in a way that it is working as a proxy. Thus, the analytics algorithm can retrieve the data by requesting the device instead of the machine itself. Another solution would be to manually copy the data to a location where the data can be retrieved from. These both tasks can not be described within a declarative deployment model, but could be realized by modelling human tasks within the imperative deployment model.

Although all components required for the Complete Deployment Model as well as the Configurable Deployment Model are known, however, implementing the management operations of the components requires specific knowledge about the components contained in the model, for example, how they can be installed and managed. Furthermore, the composition of the components itself in order to create, for example, a functioning analytics stack requires domain-specific knowledge.

VI. Related Work

The principle of function shipping is already applied in different research fields in various ways. Thus, in this section, different approaches regarding this principle are presented.

Regarding databases, for example, there is the concept of stored procedures [21]. These procedures are stored on the server, thus, they can be executed near to the data. Similar to our approach, it is also possible to develop a procedure and send it to another company. However, this approach is limited to certain databases supporting the stored procedures approach.

There is also a self-extensible database middleware system for integrating distributed data sources, called MOCHA (**M**iddleware Based **O**n a Code S**H**ipping **A**rchitecture) [22]. MOCHA enables the automatically shipping of functionality implemented in Java to the data site where the functionality should be executed. The goal of MOCHA is to reduce the amount of data that needs to be moved in order to be processed. However, MOCHA enables only the shipping of functions implemented using Java and is limited to those functions that are stored in the central code repository. In contrast, our approach is standards-based by using TOSCA, extensible regarding the implementation of the functions, and supports various application stacks as well as various data source types.

In the area of cloud computing, there is also the concept of serverless architectures [23], [24]. The idea behind the serverless architecture approach is that the developer of a function does not need to setup the system or runtime environment the function is executed on. Instead, the cloud provider is responsible to provide a runtime environment that ensures all required capabilities for executing the function and also to manage these resources. This principle is also called Function as a Service (FaaS) [23]. However, with this concept the functions can only be shipped to a cloud provider, that is providing such a serverless architecture. Thus, to ship functions near to the data, for example, into a manufacturing environment, a cloud provider independent approach is required.

Regarding distributed data processing, cluster computing frameworks such as Apache Hadoop [25] or Apache Storm [26] are making use of the function shipping principle as well. In Apache Hadoop, for example, the functions implementing the MapReduce [27] functionality are shipped to other nodes within the cluster. In Apache Storm, the topology describing the processing logic is analyzed by the master node, which distributes the tasks to worker nodes where they are executed.

All these different approaches have in common that only the function itself can be shipped into the target environment and not an entire stack consisting of the function as well as additional required components. Furthermore, these concepts are dependent on a specific target environment, for example, a specific database, a middleware, or a cloud provider, where the function can be shipped in. Thus, using the TOSCA standard enables a more flexible and generic realization of the function shipping principle.

In distributed computing, Remote Procedure Calls (RPC) are used in order to execute a procedure in a different address space. There are different implementations available such as Java RMI [28] or CORBA [29]. However, here only the request to invoke an operation is transfered and not the function itself.

Mobile agents [30] are another related paradigm in the field of distributed computing. They are enabling the autonomously migration of software from a node to another node within a network, thus reducing the network bandwidth consumption. However, from a data security and privacy point of view, autonomously moving code is often not possible in production environments as described by our motivating scenario.

In the area of microarchitecture, function shipping is also a research field. With near-data processing (NDP) [31], [32], for example, the data movement between processor chips and memory is reduced by processing the data as near as possible to the location of the data. However, this mainly is a hardware-dependent solution and not a general approach for enabling function shipping from a software perspective of view.

While function shipping is often used due to data privacy reasons, there may be other scenarios where the data have to be shipped to the function that processes the data, called *data shipping*. In the manufacturing domain, the data shipping principle is used, for example, when the IT infrastructure near the data has not enough computing power for processing the data. In this case, the data needs to be transferred to a more powerful execution environment, e.g., in a public cloud. Thus, the best suitable shipping paradigm is depending on the concrete scenario. In the area of simulation workflows, data provided by different data sources is required. SIMPL (SimTech - Information Management, Processes, and Languages) [33] is an extensible framework providing a generic abstraction for data provisioning activities in simulation workflows. Therefore, they defined extraction, transformation, and load operations (ETL) to access arbitrary external data in simulation workflows in a unified way. However, since this is a workflow specific solution, this is no general approach for realizing data shipping.

In [4] the key challenges for realizing the both paradigms function and data shipping in manufacturing environments are stated in the context of the project *SePiA.Pro (Service Platform for intelligently optimizing Applications in Production and Manufacturing)*. In this work, we discuss organizational as well as technical challenges. Furthermore, in [34] we demonstrate how an analytics stack based on Apache Flink can be automatically provisioned using OpenTOSCA.

VII. Conclusion

In this paper, we presented three different deployment modelling approaches for function shipping, which enable (i) the automated deployment and execution of these functions in a remote infrastructure as well as (ii) the wiring of these functions with data sources available in the target IT environment. For each modelling approach, we first described the main idea of the approach in an abstract manner, regarding the declarative as well as the imperative variant of this approach. Furthermore, we presented the forces that have to be tackled when applying one of these function shipping modelling approaches. In order to enable the selection of a modelling approach suitable for a

concrete scenario, we also described the benefits and drawbacks for each of the presented modelling approach. We also showed how the TOSCA standard can be used for realizing our concepts. Thus, we provide a conceptual guideline on how function shipping can be enabled in remote manufacturing environments. To validate the practicable feasibility of the presented approaches, we also implemented a prototype, which is able to process the declarative as well as the imperative variants of the shown modelling approaches. In future work we plan to find an approach for completing imperative deployment models. Since there are scenarios where the data needs to be shipped to another location, we also plan in future work to describe modelling concepts for realizing data shipping as well. In the context of the BMWi-funded project *SePiA.Pro*, together with our industrial partners we are going to evaluate the three presented function shipping modeling concepts in real world production scenarios. This way, we want to gain more insights about which use-cases benefit most from which modeling concept.

ACKNOWLEDGMENT

This work is partially funded by the BMWi project *SePiA.Pro* (01MD16013F).

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A Survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] M. Hermann, T. Pentek, and B. Otto, "Design Principles for Industrie 4.0 Scenarios," in *Proceedings of the 49th Hawaii International Conference on System Sciences (HICSS)*. IEEE, 2016, pp. 3928–3937.

[3] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi, "Machine Learning for Predictive Maintenance: A Multiple Classifier Approach," *Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 812–820, 2015.

[4] M. Falkenthal, U. Breitenbücher, M. Christ, C. Endres, A. W. Kempa-Liehr, F. Leymann, and M. Zimmermann, "Towards Function and Data Shipping in Manufacturing Environments: How Cloud Technologies leverage the 4th Industrial Revolution," in *Proceedings of the 10th Advanced Summer School on Service Oriented Computing*, ser. IBM Research Report. IBM Research Report, Sep. 2016, pp. 16–25.

[5] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, and J. Wettinger, "Integrated Cloud Application Provisioning: Interconnecting Service-Centric and Script-Centric Management Technologies," in *On the Move to Meaningful Internet Systems: OTM 2013 Conferences (CoopIS 2013)*. Springer, Sep. 2013, pp. 130–148.

[6] OASIS, *Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0*, Organization for the Advancement of Structured Information Standards (OASIS), 2013.

[7] ——, *Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0*, Organization for the Advancement of Structured Information Standards (OASIS), 2013.

[8] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, *TOSCA: Portable Automated Deployment and Management of Cloud Applications*, ser. Advanced Web Services. Springer, Jan. 2014, pp. 527–549.

[9] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, and S. Wagner, "OpenTOSCA - A Runtime for TOSCA-based Cloud Applications," in *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013)*. Springer, Dec. 2013, pp. 692–695.

[10] O. Kopp, T. Binz, U. Breitenbücher, and F. Leymann, "Winery – A Modeling Tool for TOSCA-based Cloud Applications," in *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013)*. Springer, Dec. 2013, pp. 700–704.

[11] U. Breitenbücher, T. Binz, O. Kopp, and F. Leymann, "Vinothek - A Self-Service Portal for TOSCA," in *Proceedings of the 6th Central-European Workshop on Services and their Composition (ZEUS 2014)*. CEUR-WS.org, Feb. 2014, Demonstration, pp. 69–72.

[12] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management*, vol. 35, no. 2, pp. 137–144, 2015.

[13] OASIS, *TOSCA Simple Profile in YAML Version 1.0*, Organization for the Advancement of Structured Information Standards (OASIS), 2015.

[14] M. Mohaan and R. Raithatha, *Learning Ansible*. Packt Publishing, Nov. 2014.

[15] Opscode, Inc., "Chef Official Site," Feb. 2016. [Online]. Available: http://www.opscode.com/chef

[16] OASIS, *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*, Organization for the Advancement of Structured Information Standards (OASIS), 2007.

[17] OMG, *Business Process Model and Notation (BPMN) Version 2.0*, Object Management Group (OMG), 2011.

[18] O. Kopp, T. Binz, U. Breitenbücher, and F. Leymann, "BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications," in *Proceedings of the 4th International Workshop on the Business Process Model and Notation (BPMN 2012)*. Springer, Sep. 2012, pp. 38–52.

[19] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann, and T. Michelbach, "A Domain-Specific Modeling Tool to Model Management Plans for Composite Applications," in *Proceedings of the 7th Central European Workshop on Services and their Composition, ZEUS 2015*. CEUR Workshop Proceedings, May 2015, pp. 51–54.

[20] U. Breitenbücher, T. Binz, K. Képes, O. Kopp, F. Leymann, and J. Wettinger, "Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA," in *International Conference on Cloud Engineering (IC2E 2014)*. IEEE, Mar. 2014, pp. 87–96.

[21] G. Harrison and S. Feuerstein, *MySQL Stored Procedure Programming*. O'Reilly Media, Mar. 2006.

[22] M. Rodríguez-Martínez and N. Roussopoulos, "MOCHA: a Self-Extensible Database Middleware System for Distributed Data Sources," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD 00)*. ACM, 2000, pp. 213–224.

[23] M. Roberts, "Serverless Architectures," Aug. 2016. [Online]. Available: http://martinfowler.com/articles/serverless.html

[24] AWS, "AWS Serverless Multi-Tier Architectures: Using Amazon API Gateway and AWS Lambda," Nov. 2015. [Online]. Available: https://d0.awsstatic.com/whitepapers/AWS_Serverless_Multi-Tier_Architectures.pdf

[25] The Apache Software Foundation, "Welcome to Apache Hadoop!" [Online]. Available: http://hadoop.apache.org/

[26] ——, "Apache Storm." [Online]. Available: http://storm.apache.org/

[27] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[28] Oracle, "Java Remote Method Invocation - Distributed Computing for Java," http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html, Oracle, 2010.

[29] OMG, *CORBA 3.3*, Object Management Group (OMG), 2012.

[30] V. A. Pham and A. Karmouch, "Mobile Software Agents: An Overview," *IEEE Communications Magazine*, vol. 36, no. 7, pp. 26–37, 1998.

[31] M. Gao, G. Ayers, and C. Kozyrakis, "Practical Near-Data Processing for In-memory Analytics Frameworks," in *Proceedings of the 2015 International Conference on Parallel Architecture and Compilation (PACT)*. IEEE, 2015, pp. 113–124.

[32] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-Data Processing: Insights from a MICRO-46 Workshop," *IEEE Micro*, vol. 34, no. 4, pp. 36–42, Aug. 2014.

[33] P. Reimann, M. Reiter, H. Schwarz, D. Karastoyanova, and F. Leymann, "SIMPL-A Framework for Accessing External Data in Simulation Workflows," in *Datenbanksysteme für Business, Technologie und Web (BTW 2011): 14. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS)*. Gesellschaft für Informatik (GI), 2011, pp. 534–553.

[34] M. Falkenthal, U. Breitenbücher, K. Képes, F. Leymann, M. Zimmermann, M. Christ, J. Neuffer, N. Braun, and A. W. Kempa-Liehr, "OpenTOSCAfor the 4th Industrial Revolution: Automating the Provisioning of Analytics Tools based on Apache Flink," in *Proceedings of the 6th International Conference on the Internet of Things*. ACM, 2016, pp. 179–180.

All links were last followed on May 20, 2017.