Institute of Architecture of Application Systems

Decentralized Cross-Organizational Application Deployment Automation: An Approach for Generating Deployment Choreographies Based on Declarative Deployment Models

Karoline Wild, Uwe Breitenbücher, Kálmán Képes,
Frank Leymann, Benjamin Weder

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{wild, breitenbuecher, kepes, leymann, weder}@iaas.uni-stuttgart.de

BibTeX:

```
@article {Wild2020_DeploymentChoreographies,
    Author    = {Karoline Wild and Uwe Breitenb{\"u}cher and K{\'a}lm{\'a}n
                 K{\'e}pes and Frank Leymann and Benjamin Weder},
    Title     = {{Decentralized Cross-Organizational Application Deployment
                 Automation: An Approach for Generating Deployment
                 Choreographies Based on Declarative Deployment Models}},
    Booktitle = {Proceedings of the 32nd Conference on Advanced Information
                 Systems Engineering (CAiSE 2020)},
    Publisher = {Springer International Publishing},
    Series    = {Lecture Notes in Computer Science},
    Volume    = {12127},
    Pages     = {20--35},
    Month     = jun,
    Year      = 2020,
    doi       = {10.1007/978-3-030-49435-3_2}
}
```

Decentralized Cross-Organizational Application Deployment Automation: An Approach for Generating Deployment Choreographies Based on Declarative Deployment Models

Karoline Wild[0000−0001−7803−6386], Uwe Breitenbücher[0000−0002−8816−5541], Kálmán Képes[0000−0002−1392−9789], Frank Leymann[0000−0002−9123−259X], and Benjamin Weder[0000−0002−6761−6243]

Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany
{wild,breitenbuecher,kepes,leymann,weder}@iaas.uni-stuttgart.de

Abstract. Various technologies have been developed to automate the deployment of applications. Although most of them are not limited to a specific infrastructure and able to manage multi-cloud applications, they all require a central orchestrator that processes the deployment model and executes all necessary tasks to deploy and orchestrate the application components on the respective infrastructure. However, there are applications in which several organizations, such as different departments or even different companies, participate. Due to security concerns, organizations typically do not expose their internal APIs to the outside or leave control over application deployments to others. As a result, centralized deployment technologies are not suitable to deploy cross-organizational applications. In this paper, we present a concept for the decentralized cross-organizational application deployment automation. We introduce a global declarative deployment model that describes a composite cross-organizational application, which is split to local parts for each participant. Based on the split declarative deployment models, workflows are generated which form the deployment choreography and coordinate the local deployment and cross-organizational data exchange. To validate the practical feasibility, we prototypical implemented a standard-based end-to-end toolchain for the proposed method using TOSCA and BPEL.

Keywords: Distributed Application · Deployment · Choreography · TOSCA · BPEL.

1 Introduction

In recent years various technologies for the automated deployment, configuration, and management of complex applications have been developed. These *deployment automation technologies* include technologies such as Chef, Terraform, or Ansible to name some of the most popular [27]. Additionally, standards such as the Topology and Orchestration Specification for Cloud Applications

(TOSCA) [20] have been developed to ensure portability and interoperability between different environments, e.g., different cloud providers or hypervisors. These deployment automation technologies and standards support a declarative deployment modeling approach [9]. The deployment is described as *declarative deployment model* that specifies the desired state of the application by its components and their relations. Based on this structural description a respective deployment engine derives the necessary actions to be performed for the deployment. Although most of these technologies and standards are not limited to a specific infrastructure and able to manage multi-cloud applications, they all use a central orchestrator for the deployment execution. This central orchestrator processes the declarative deployment model and either forwards the required actions in order to deploy and orchestrate the components to agents, e.g., in the case of Chef to the Chef clients running on the managed nodes, or executes them directly, e.g., via ssh on a virtual machine (VM), as done by Terraform [25].

However, today's applications often involve multiple participants, which can be different departments in a company or even different companies. Especially in Industry 4.0 the collaboration in the value chain network is of great importance, e.g., for remote maintenance or supply chain support [7]. All these applications have one thing in common: They are *cross-organizational applications* that composite distributed components, whereby different participants are responsible for different parts of the application. The deployment and management of such applications cannot be automated by common multi-cloud deployment automation technologies [22], since their central orchestrators require access to the internal infrastructure APIs of the different participants, e.g., the OpenStack API of the private cloud, or their credentials, e.g., to login to AWS. There are several reasons for the involved participants to disclose where and how exactly the application components are hosted internally: new security issues and potential attacks arose, legal and compliance rules must be followed, and the participant wants to keep the control over the deployment process [17]. This means that common centralized application deployment automation technologies are not suitable to meet the requirements of new emerging application scenarios that increasingly rely on cross-organizational collaborations.

In this paper, we address the following research question: *"How can the deployment of composite applications be executed across organizational boundaries involving multiple participants that do not open their infrastructure APIs to the outside in a fully automated decentralized manner?"* We present a concept for the decentralized cross-organizational application deployment automation that (i) is capable of globally coordinating the entire composite application deployment in a decentralized way while (ii) enabling the involved participants to control their individual parts locally. Therefore, we introduce a *global multi-participant deployment model* describing the composite cross-organizational application, which is split into local parts for each participant. Based on the local deployment models a deployment choreography is generated, which is executed in a decentralized manner. Based on the TOSCA and BPEL [19] standards the existing OpenTOSCA ecosystem [6] is extended for the proposed method and validated prototypically.

2 Declarative and Imperative Deployment Approaches

For application deployment automation two general approaches can be distinguished: declarative and imperative deployment modeling approaches [9]. For our decentralized cross-organizational application deployment automation concept both approaches are combined. Most of the deployment automation technologies use *deployment models* that can be processed by the respective deployment engine. Deployment models that specify the actions and their order to be executed, e.g., as it is done by workflows, are called *imperative deployment models*, deployment models that specify the desired state of an application are called *declarative deployment models* [9]. We explain the declarative deployment models in a technology-independent way based on the *Essential Deployment Meta Model (EDMM)* that has been derived from 13 investigated deployment technologies in previous work [27]. The meta model for declarative deployment models presented in section 3 is based on the EDMM and is the basis for the declarative part of the presented concept.

In EDMM an application is defined by its *components* and their *relations*. For the semantic of these components and relations reusable *component* and *relation types* are specified. For example, it can be defined that a web application shall be hosted on an application server and shall be connected to a queue to publish data that are processed by other components. For specifying the configuration of the components *properties* are defined, e.g., to provide the credentials for the public cloud or to set the name of the database. For instantiating, managing, and terminating components and relations executable artifacts such as shell scripts or services are encapsulated as *operations* that can be executed to reach the desired state defined by the deployment model. The execution order of the operations is derived from the deployment model by the respective deployment engine [5].

In contrast, imperative deployment models explicitly specify the actions and their order to be executed to instantiate and manage an application [9]. Actions can be, e.g., to login to a public cloud or to install the WAR of a web application on an application server. Especially for complex applications or custom management behavior imperative deployment models are required, since even if declarative models are intuitive and easy to understand, they do not enable to customize the deployment and management. Imperative deployment technologies are, e.g., BPMN4TOSCA [16], and general-purpose technologies such as BPEL, BPMN [21], or scripting languages. In general, declarative deployment models are more intuitive but the execution is less customizable, while imperative deployment models are more complex to define but enable full control of the deployment steps. Therefore, there are hybrid approaches for using declarative models that are transformed into imperative models to get use of the benefits of both approaches [5]. In this paper, we follow this hybrid approach by transforming declarative models to imperative choreography models. This means, the user only has to specify the declarative model and, thus, we explain the declarative modeling approach in section 4 using a motivating scenario. First, in the next section the meta model for declarative deployment models is introduced.

3 Meta Model for Declarative Deployment Models

Our approach presented in section 5 is based on declarative deployment models that are transformed into imperative choreographies. Based on EDMM and inspired by the Declarative Application Management Modeling and Notation (DMMN) [3], the GENTL meta model [1], and TOSCA, a definition of declarative deployment models $d \in D$ is introduced:

Definition 1 (Declarative Deployment Model). *A declarative deployment model $d \in D$ is a directed, weighted, and possibly disconnected graph and describes the structure of an application with the required deployment operations:*

$$d = (C_d, R_d, CT_d, RT_d, O_d, V_d, type_d, operations_d, properties_d)$$

The elements of the tuple d are defined as follows:

- C_d: Set of components in d, whereby each $c_i \in C_d$ represents one component.
- $R_d \subseteq C_d \times C_d$: Set of relations in d, whereby each $r_i = (c_s, c_t) \in R_d$ represents the relationship and c_s is the source and c_t the target component.
- CT_d and RT_d: The set of component and relation types in d, whereby each $ct_i \in CT_d$ and $rt_i \in RT_d$ describes the semantics for components and relations having this type.
- $O_d \subseteq \wp(V_d) \times \wp(V_d)$: The set of operations in d, whereby each operation $o_i = (Input, Output) \in O_d$ specifies an operation that can be applied to components or relations with its input and output parameters.
- $V_d \subseteq \wp(\Sigma^+) \times \wp(\Sigma^+)$: Set of data elements, whereby Σ^+ is the set of characters in the ASCII table and $v_i = (datatype, value) \in V_d$.

Let the set of deployment model elements $DE_d := C_d \cup R_d$ be the union of components and relations of d. Let the set of deployment model element types $DET_d := CT_d \cup RT_d$ be the union of component types and relation types of d.

- $type_d$: The mapping assigning each $de_i \in DE_d$ to a component or relation type $det_i \in DET_d$: $type_d : DE_d \rightarrow DET_d$.
- $operations_d$: The mapping assigning each $de_i \in DE_d$ to a set of operations that can be applied to it: $operations_d : DE_d \rightarrow \wp(O_d)$.
- $properties_d$: The mapping assigning each $de_i \in DE_d$ to a set of data elements which are the properties of the element: $properties_d : DE_d \rightarrow \wp(V_d)$. ■

4 Research Method and Motivating Scenario

Following the design cycle by Wieringa [26], we first examined the current situation in various research projects with industrial partners, namely in the projects IC4F[1], SePiA.Pro[2], and SmartOrchestra[3]. With regard to horizontal integration through the value chain network in the context of Industry 4.0, we focused

[1] https://www.ic4f.de/
[2] http://projekt-sepiapro.de/en/
[3] https://smartorchestra.de/en/

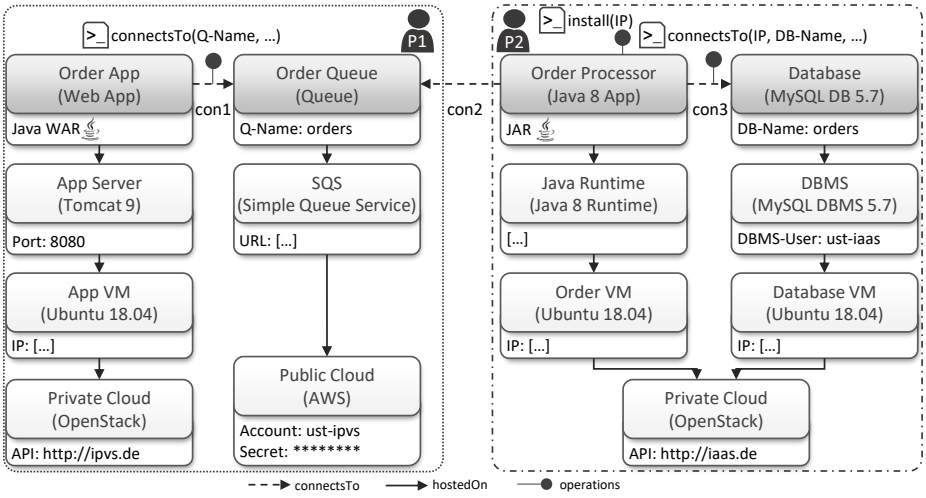

Fig. 1. Declarative deployment model specifying all details of the desired application. The notation is based on Vino4TOSCA with components as nodes, relations as edges, and the types in brackets [4]. In addition, sample operations are shown as dots.

on the requirements and challenges of collaboration between different companies [7]. Based on our previous research focus, the deployment and management of applications, the following research problems have emerged:

(a) How can the deployment of composite applications across organizational boundaries be automated in a decentralized manner?
(b) What is the minimal set of data to be shared between the involved participants to enable the automated decentralized deployment?

In fig. 1 a declarative deployment model according to the meta model introduced in section 3 is depicted for an order application to motivate the research problem. The application consists of four components: a web application *Order App* sending orders to the *Order Queue* and an *Order Processor* that processes the orders and stores them in the *Database*. These four components, depicted in dark gray with their component types in brackets, are the so-called *application-specific components*, since they represent the use case to be realized. For the Order Queue and Database, e.g., properties are specified to set the name of the queue and database, respectively. In addition, three operations are exemplary shown: a *connectsTo* to establish a connection to the queue, a *connectsTo* to connect to the database, and an *install* operation to install the JAR artifact on the *Order VM*. The other properties and operations are abstracted.

Assuming that a single organization is responsible for deploying the entire application and has full control over the OpenStacks and AWS, the common deployment automation technologies examined by Wurster et al. [27] fit perfectly. However, in the depicted scenario two participants, *P1* and *P2*, who may be different departments or companies, intend to realize a cross-organizational

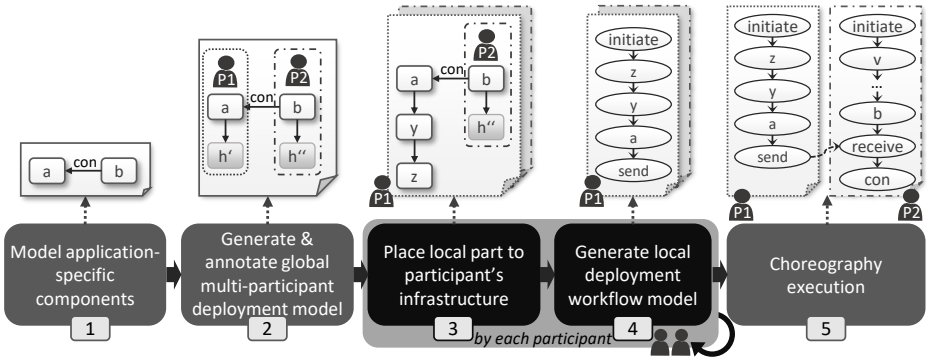

Fig. 2. Decentralized cross-organizational application deployment automation.

application so that common deployment automation technologies are no longer applicable. While all participants must agree on the application-specific components, the underlying infrastructure is the responsibility of each participant. For security reasons, participants typically do not provide access to internal APIs, share the credentials for AWS, or leave the control over deployment to others.

To address the research problems, we propose a decentralized concept to enable the cross-organizational application deployment automation ensuring that (i) only as little data as necessary is exchanged between participants and (ii) each participant controls only his or her local deployment while the overall deployment is coordinated. The proposed solution is described in detail in the following section and in section 6 the implementation and validation is presented. The motivating scenario in fig. 1 serves as use case for the validation.

5 A Concept for Decentralized Cross-Organizational Deployment Automation

For the decentralized cross-organizational application deployment automation with multiple participants, it has to be considered that (i) the participants want to exchange as little data as necessary and (ii) each participant controls only his or her local deployment while the global coordination of the deployment of the entire application is ensured. Taking these requirements into account, we have developed the deployment concept depicted in fig. 2. In the first step, the application-specific components are modeled representing the use case to be realized. They typically include the business components such as the Order App, storage components such as the Database component, and communication components such as the Order Queue in fig. 1. In the second step, the *global multi-participant deployment model (GDM)* is generated, a declarative deployment model containing all publicly visible information that is shared between the participants. This publicly visible information contains also data that must be provided by the respective infrastructure. For example, to execute the operation

to establish a connection between Order Processor and Database in fig. 1, the IP of the Database VM is required as input. Subgraphs, so called *local parts* of the GDM, are then assigned to participants responsible for the deployment of the respective components. The GDM is then processed by each participant. First, in step three, for each application-specific component a hosting environment is selected and the adapted model stored as *local multi-participant deployment model (LDM)*. In the motivating scenario in fig. 1 participant P1 selected AWS for the Order Queue and the OpenStack for the Order App. However, this individual placement decision is not shared. For the deployment execution we use an hybrid approach: Based on the LDM a local deployment workflow model is generated in step four that orchestrates the local deployment and cross-organizational information exchange activities. All local workflows form implicitly the deployment choreography which enables the global coordination of the deployment across organizational boundaries. Each step is described in detail in the following.

5.1 Step 1: Application-Specific Component Modeling

In the initial step, the application-specific components representing the use case to be realized have to be modeled. They typically include business components, storage components, and communication components. In the motivating scenario in fig. 1 the set of application-specific components contains the Order App, the Order Queue, the Order Processor, and the Database. In addition, the lifecycle operations, e.g., to install, start, stop, or terminate the components and relations, have to be defined for each of these components and their relations, since all input parameters of these operations must be provided as globally visible information in the GDM. Application-specific components are defined as follows:

Definition 2 (Application-Specific Components). *The set of application-specific components $C_s \subseteq C_d$ in d, where all $r_s = (c_s, c_t) \in R_d$ with $\{c_s, c_t\} \in C_s$ are of $type_d(r_s) = connectsTo$ and for each $c_i \in C_s : cap(type_d(c_i)) = \emptyset$, since they cannot offer hosting capabilities (see definition 3).* ■

In addition, it has to be expressed that a component can have a certain requirement and that a component that provides a matching capability can serve as host, i.e., the component is the target of a relation of type *hostedOn*:

Definition 3 (Hosting Requirements and Capability). *Let RC the set of hosting requirement-capability pairs. The mapping $req : C_d \rightarrow \wp(RC)$ and $cap : C_d \rightarrow \wp(RC)$ assign to each component the set of capabilities and requirements, respectively. The hosting capability of a component $c_y \in C_d$ matches the hosting requirement of a component $c_z \in C_d$, if it exists $rc \in RC$ with $rc \in req(c_z) \cap cap(c_y)$, then c_y can host c_z.* ■

5.2 Step 2: Global Multi-Participant Deployment Model Generation

To ensure that the application-specific components can be deployed across organizational boundaries, the GDM is generated in the second step which contains

the minimal set of necessary information that have to be globally visible, i.e., that have to be shared. Thus, the GDM is defined as follows:

Definition 4 (Global Multi-Participant Deployment Model). *A global multi-participant deployment model (GDM) is an annotated declarative deployment model that contains all globally visible information including the (i) application-specific components, (ii) necessary information about the hosting environment, and (iii) the participants assigned to parts of the GDM:*

$$g = (d, P_g, participant_g)$$

The elements of the tuple g are defined as follows:

- $d \in D$: Declarative deployment model that is annotated with participants.
- $P_g \subseteq \wp(\Sigma^+) \times \wp(\Sigma^+)$: Set of participants with $p_i = (id, endpoint) \in P$, whereby Σ^+ is the set of characters in the ASCII table.
- $participant_g$: The mapping assigning a component $c_i \in C_d$ to a participant $p_i \in P_g$ $participant_g : C_d \rightarrow P_g$. ■

The example in fig. 3 depicts a simplified GDM. The application-specific components, depicted in dark gray, specify requirements, e.g., the Order Queue requires a message queue middleware. These requirements have to be satisfied by the respective hosting environment. Furthermore, for these components as well as their connectsTo-relations operations with input parameters are defined. To establish a connection to the Order Queue the *URL* and *Q-Name* of the Queue are required. Either the target application-specific component provides respective matching properties such as the *Q-Name* property exposed by the Order Queue component or the environment has to provide it such as the input parameter *URL*. For this, in this step *placeholder host* components are generated that contain all capabilities and properties that have to be exposed by the hosting environment. Each placeholder host component c_h is generated based on the following rules:

- For all $c_j \in C_s : req(c_j) \neq \emptyset$ a placeholder host component $c_h \in C_h$ and a hosting relation $r_h = (c_j, c_h) \in R_d$ with $type_d(r_h) = hostedOn$ are generated. Thus, $|C_h| \leq |C_s|$ since external services do not require a host.
- For each operation $op_j \in operations_s(c_j)$ all input data elements $v_j \in \pi_1(op_j) \setminus properties_d(c_j)$ are added to $properties_d(c_h)$.
- For each $r_j \in R_d : \pi_2(r_j) = c_j$ with $type_d(r_j) = connectsTo$ and for each operation $op_r \in operations_s(r_j)$ all data elements $v_r \in \pi_1(op_r) \setminus properties_s(c_j)$ are added to $properties_d(c_h)$.

In the example in fig. 3 the *Host Order Queue* component provides the capability *MessageQueue* and exposes the property *URL*, which is required as input parameter for the *connectsTo* operations. Before the deployment model is processed by each participant, subgraphs of the GDM are assigned to the participants. This subgraph is called *local part* and indicates who is responsible for this part of the application. This is done by annotating the GDM as shown in fig. 3 on the right.

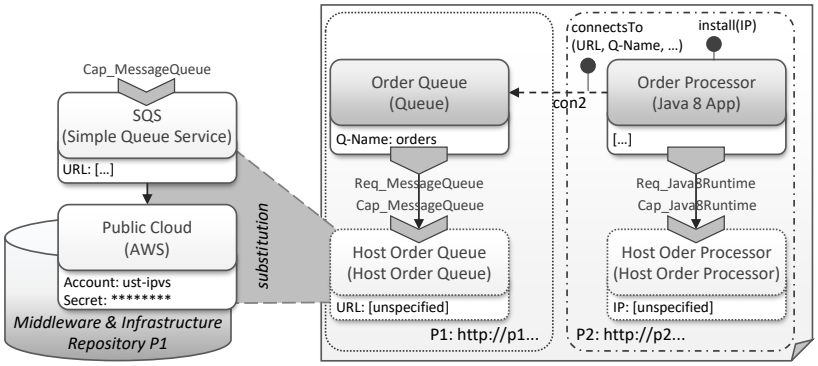

Fig. 3. Substitution of the database host from the motivating scenario.

5.3 Step 3: Local Part Placement

Since participants typically do not want to share detailed information about their hosting environment, the GDM is given to each participant for further processing. Each participant p_i has to select a hosting environment for all $c_s \in C_s$ with $participant_g(c_s) = p_i$. In fig. 3 the placement of the Order Queue component by substituting the placeholder host component *Order Queue Host* is shown. A placeholder host can be substituted by a stack that exposes the same capabilities and at least all properties of the placeholder host. These stacks are stored as declarative deployment models that contain middleware and infrastructure components available in the environment of the respective participant. In fig. 3 a *SQS* hosted on AWS is shown. The substitution is based on the substitution mechanisms of TOSCA [20,24]: A placeholder host component $c_h \in C_h$ can be substituted by a deployment model m if for each $prop_h \in properties_d(c_h)$ a component $c_f \in C_m : prop_h \in properties_m(c_f)$ and for each $cap_h \in cap(c_h)$ a component $c_k \in C_m : cap_h \in cap(c_k)$ exists.

The substitution in fig. 3 is valid because the property *URL* is covered and the *SQS* exposes the required capability *MessageQueue*. The substitution is automated by our prototype described in section 6. For the distribution of components and matching to existing infrastructure and middleware several approaches exist [24,12,23]. Soldani et al. [24] introduced the ToscaMart method to reuse deployment models to derive models for new applications, Hirmer et al. [12] introduced a component wise completion, and we presented in previous work [23] how to redistribute a deployment model to different cloud offerings. These approaches use a requirement-capability matching mechanism to select appropriate components. We extended this mechanism to match the properties as well. The resulting *local multi-participant deployment model (LDM)* is a partially substituted GDM with detailed middleware and infrastructure components for the application-specific components managed by the respective participant. Up to this point we follow a purely declarative deployment modeling approach.

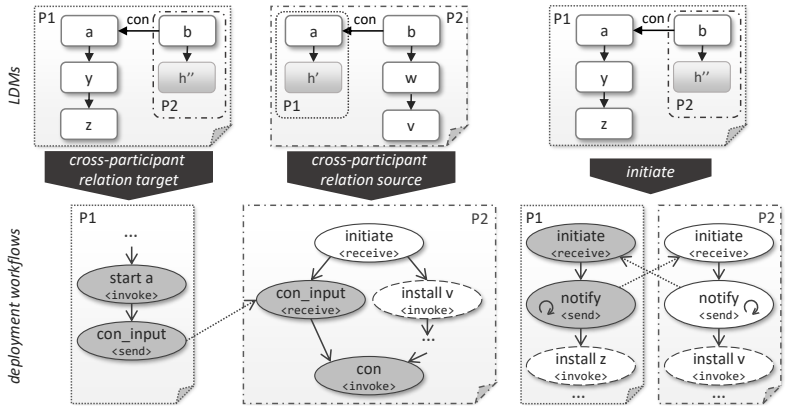

Fig. 4. Generated activities to (a) send information to relation source, (b) receive information from relation target, and (c) initiate the overall deployment.

5.4 Step 4: Local Deployment Workflow Generation

The core step of our approach is the generation of local deployment workflow models that form the deployment choreography. They are derived from the LDMs by each participant and (i) orchestrate all local deployment activities and (ii) coordinate the entire deployment and data exchange to establish *cross-participant relations*. While centralized deployment workflows can already be generated [5], the global coordination and data exchange are not covered yet.

Cross-participant relations are of type *connectsTo* and between components managed by different participants. To establish cross-participant relations, the participants have to exchange the input parameters for the respective connectsTo-operations. In the example in fig. 3 the relation *con2* establishes a connection from the Order Processor managed by P2 to the Order Queue managed by P1. The connectsTo-operation requires the *URL* and the *Q-Name* as input. Both parameters have to be provided by P1. Since this information is first available during deployment time, this data exchange has to be managed during deployment: For each cross-participant relation a sending and receiving activity is required to exchange the information after the target component is deployed and before the connection is established. In addition, the deployment of the entire application must be ensured. Independent which participant initiates the deployment, all other participants have to deploy their parts as well. This is covered by three cases that have to be distinguished for the local deployment workflow generation as conceptually shown in fig. 4. In the upper part abstracted LDMs and in the lower part generated activities from the different participants perspectives are depicted. On the left (a) activities from a *cross-participant relation target* perspective, in the middle (b) from a *cross-participant relation source* perspective, and on the right (c) activities generated to ensure the *initiation* of the entire deployment are depicted. First, a definition of local deployment workflow models based on the production process definition [14,18] is provided:

Definition 5 (Local Deployment Workflow Model). *For each participant $p_i \in P$ a local deployment workflow model w_i based on the LDM is defined as:*

$$w_i = (A_{w_i}, E_{w_i}, V_{w_i}, i_{w_i}, o_{w_i}, type_{w_i})$$

The elements of the tuple w_i are defined as follows:

- A_{w_i}: Set of activities in w_i with $a_y \in A_{w_i}$.
- $E_{w_i} \subseteq A_{w_i} \times A_{w_i}$: Set of control connectors between activities, whereby each $e_y = (a_s, a_t) \in E_{w_i}$ represents that a_s has to be finished before a_t can start.
- $V_{w_i} \subseteq \wp(\Sigma^+) \times \wp(\Sigma^+)$: Set of data elements, whereby Σ^+ is the set of characters in the ASCII table and $v_y = (datatype, value) \in V_{w_i}$.
- i_{w_i}: The mapping assigns to each activity $a_y \in A_{w_i}$ its input parameters and it is called the input container $i_{w_i} : A_{w_i} \rightarrow \wp(V_{w_i})$.
- o_{w_i}: The mapping assigns to each activity $a_y \in A_{w_i}$ its output parameters and it is called the output container $o_{w_i} : A_{w_i} \rightarrow \wp(V_{w_i})$.
- $type_{w_i}$: The mapping assigns each $a_y \in A_{w_i}$ to an activity type, $type_{w_i} : A_{w_i} \rightarrow \{\texttt{receive}, \texttt{send}, \texttt{invoke}\}$. ■

Based on this definition, local deployment workflow models can be generated based on specific rules. In fig. 4 the resulting activities are depicted:

(a) For each component $c_t \in C_d$ that is target of a cross-participant relation $r_c = (c_s, c_t)$ with $participant_g(c_t) = p_i$ and $participant_g(c_s) = p_j$, an activity $a_t \in A_{w_i} : type_{w_i}(a_t) = \texttt{invoke}$ is added that invokes the *start* operation of c_t. After a component is started, a connection to it can be established [5]. Thus, $a_c : type_{w_i}(a_c) = \texttt{send}$ is added to w_i that contains all input parameters of the connectsTo-operation of r_c provided by p_i in $o_{w_i}(a_c)$.

(b) For the component $c_s \in C_d$, the source of the cross-participant relation r_c, an activity $a_{c'} : type_{w_j}(a_{c'}) = \texttt{receive}$ is add to w_j of p_j. With the control connector $e(a_{init}, a_{c'})$ added to w_j it is ensured that the activity is activated after the *initiate* activity of p_j. After the input values are received and the *start* operation of c_s is successfully executed, the actual connectsTo-operation can be executed.

(c) Each workflow w_i starts with the *initiate* activity $a_{init} \in A_{w_i} : type_{w_i}(a_{init}) = \texttt{receive}$. To ensure that after a_{init} is called the entire application deployment is initiated, a notification is sent to all other participants. For each $p_j \in P \setminus \{p_i\}$ an activity $a_n : type_{w_i}(a_n) = \texttt{send}$ with a control connector $e(a_{init}, a_n)$ is added to w_i. Since each participant notifies all others, for n participants, each participant has to discard n-1 messages. Since the payloads are at most a set of key-value pairs this is not critical.

Each participant generates a local deployment workflow model, which together implicitly form the deployment choreography. As correlation identifier the GDM id and application instance id are sufficient. While the GDM id is known in advance, the application instance id is generated by the initiating participant. The approach enables a decentralized deployment while each participant controls only his or her deployment and shares only necessary information.

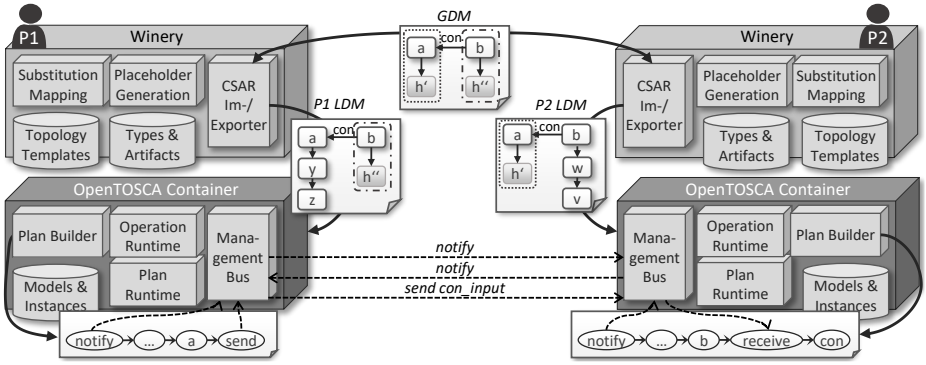

Fig. 5. System architecture and deployment choreography execution.

6 Implementation and Validation

To demonstrate the practical feasibility of the approach we extended the TOSCA-based open-source end-to-end toolchain *OpenTOSCA*[4] [6]. It consists of a modeling tool *Winery*, a deployment engine *OpenTOSCA Container*, and a self-service portal. In TOSCA, deployment models are modeled as *topology templates*, the components as *node*, and the relations as *relationship templates* with their *types*. The types define *properties*, *operations*, *capabilities*, and *requirements*. *Plans* are the imperative part of TOSCA, for which standard workflow languages such as BPMN or BPEL can be used. All TOSCA elements and executables, implementing operations and components, are packaged as *Cloud Service Archive (CSAR)*.

In fig. 5 the system architecture for two participants is depicted. Winery is extended by the *placeholder generation* and the *placeholder substitution*. Either P1 or P2 models the application-specific components and generates the GDM using the *Placeholder Generation* that generates node types with the respective properties and capabilities. The resulting GDM is then packaged with the *CSAR Im-/ Exporter* and sent to each participant. The *Substitution Mapping* detects the local part of managed by the respective participant in the GDM and selects topology templates from the repository to substitute the placeholder host components. The substituted topology template is then uploaded to the OpenTOSCA Container. The *Plan Builder* generates a deployment plan based on the declarative model. We use BPEL for the implementation. Either P1 or P2 can then initiate the deployment. The *Plan Runtime* instantiates the plan and invokes the operations. The actual operation, e.g., to create a VM, is executed by the *Operation Runtime*. The communication between the OpenTOSCA Containers is managed by the *Management Bus*. The Management Bus is the participant's endpoint in our setup. However, also arbitrary messaging middleware or any other endpoint that can process the messages can be used. We used the deployment model presented in fig. 1 with two and three participants for the validation.

[4] https://github.com/OpenTOSCA

7 Validity, Limitations, and Implications

In contrast to general workflow approaches [14,15], we do not have to deal with splitting workflows according to the participants, since we can completely rely on the declarative deployment model and only implicitly generates a choreography. However, a prerequisite is that each participant only uses the predefined interfaces so that the choreography can be executed. At present, we also limit ourselves to the deployment aspect and do not consider the subsequent management. While management functionalities such as scaling are often covered by the cloud providers themselves, other functionalities such as testing, backups, or updates are not offered. Management increases the complexity of automation, especially when local management affects components managed by other participants. We currently only support TOSCA as a modeling language and OpenTOSCA as a deployment engine. So far, we lack the flexibility to support technologies like Kubernetes, Terraform, or Chef, which are often already in use in practice. However, this is part of the planned future work.

8 Related Work

The research in the field of *multi-cloud*, *federated cloud*, and *inter-cloud* [22,10] focuses on providing unified access to different cloud providers, making placement decisions, migration, and management. All these approaches consider multiple cloud providers satisfying the requirements of a single user. The cloud forms differ in whether the user is aware of using several clouds or not. However, the collaboration between different users each using and controlling his or her environment, whether it is a private, public, or multi-cloud, is not considered, but this is highly important, especially in cross-company scenarios which arose with new emerging use cases in the fourth industrial revolution. Arcangeli et al. [2] examined the characteristics of deployment technologies for distributed applications and also considered the deployment control, whether it is centralized or decentralized. However, also the decentralized approaches with a peer-to-peer approach does not consider the sovereignty of the involved peers and the communication restrictions. In previous work [13], we introduced an approach to enable the deployment of parts of an application in environments that restrict incoming communication. However, the control is still held by a central orchestrator.

Kopp and Breitenbücher [17] motivated that choreographies are essential for distributed deployments. Approaches for modeling choreographies, e.g., with BPEL [8] or to split orchestration workflows into multiple workflows [15,14] have been published. However, most of the deployment technologies are based on a declarative deployment models [27], since defining the individual tasks to be performed in the correct order to reach a desired state are error-prone. Thus, instead of focusing on workflow choreographies we implicitly generated a choreography based on declarative deployment models. Breitenbücher et al. [5] demonstrated how to derive workflows from declarative deployment models. However, their approach only enables to generate orchestration workflows which cannot be used

for decentralized cross-organizational deployments. Herry et al. [11] introduced a planning based approach to generate a choreography. However, they especially focus on generating an overall choreography that can be executed by several agents. For us the choreography is only an implicit artifact, since we mainly focus on enabling the cross-organizational deployment by minimizing the globally visible information and obtaining the sovereignty of the participants.

9 Conclusion and Future Work

In this paper, we presented an approach for the decentralized deployment automation of cross-organizational applications involving multiple participants. A cross-organizational deployment without a central trusted third-party is enabled based on a declarative deployment modeling approach. The approach facilitates that (i) each participant controls the local deployment, while the global deployment is coordinated and (ii) only the minimal set of information is shared. A declarative global multi-participant deployment model that contains all globally visible information is generated and split to local deployment models that are processed by each participant. Each participant adapts the local model with internal information and generates an imperative deployment workflow. These workflows form the deployment choreography that coordinates the entire application deployment. We implemented the concept by extending the OpenTOSCA ecosystem using TOSCA and BPEL. In future work the data exchange will be optimized since each participant sends notification messages to all other participant and thus for n participants n-1 messages have to be discarded. We further plan not only to enable multi-participant deployments but also multi-technology deployments by enabling to orchestrate multiple deployment technologies.

Acknowledgments. This work is partially funded by the BMWi project *IC4F* (01MA17008G), the DFG project *DiStOPT* (252975529), and the DFG's Excellence Initiative project SimTech (EXC 2075 - 390740016).

References

1. Andrikopoulos, V., Reuter, A., Sáez, S.G., Leymann, F.: A GENTL approach for cloud application topologies. In: Service-Oriented and Cloud Computing. Springer Nature (2014)
2. Arcangeli, J.P., Boujbel, R., Leriche, S.: Automatic deployment of distributed software systems: Definitions and state of the art. Journal of Systems and Software **103** (2015)
3. Breitenbücher, U.: Eine musterbasierte Methode zur Automatisierung des Anwendungsmanagements. Dissertation, Universität Stuttgart, Fakultaet Informatik, Elektrotechnik und Informationstechnik (2016)
4. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Schumm, D.: Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA. In: CoopIS 2012. Springer (2012)

5. Breitenbücher, U., et al.: Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In: IC2E 2014. IEEE (2014)
6. Breitenbücher, U., et al.: The opentosca ecosystem - concepts & tools. In: European Space project on Smart Systems, Big Data, Future Internet - Towards Serving the Grand Societal Challenges - Volume 1: EPS Rome 2016,. SciTePress (2016)
7. Camarinha-Matos, L.M., Fornasiero, R., Afsarmanesh, H.: Collaborative networks as a core enabler of industry 4.0. In: Collaboration in a Data-Rich World. Springer International Publishing (2017)
8. Decker, G., Kopp, O., Leymann, F., Weske, M.: Bpel4chor: Extending bpel for modeling choreographies. In: ICWS 2007 (2007)
9. Endres, C., et al.: Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications. In: PATTERNS (2017)
10. Grozev, N., Buyya, R.: Inter-cloud architectures and application brokering: taxonomy and survey. Software: Practice and Experience **44**(3) (2012)
11. Herry, H., Anderson, P., Rovatsos, M.: Choreographing Configuration Changes. In: CNSM 2013. IEEE (2013)
12. Hirmer, P., et al.: Automatic Topology Completion of TOSCA-based Cloud Applications. In: GI-Jahrestagung, GI, vol. P-251. GI (2014)
13. Képes, K., Breitenbücher, U., Leymann, F., Saatkamp, K., Weder, B.: Deployment of Distributed Applications across Public and Private Networks. In: EDOC. IEEE (2019)
14. Khalaf, R.: Supporting Business Process Fragmentation While Maintaining Operational Semantics: A BPEL Perspective. Dissertation, Universität Stuttgart, Fakultaet Informatik, Elektrotechnik und Informationstechnik (2006)
15. Khalaf, R., Leymann, F.: E role-based decomposition of business processes using bpel. In: ICWS'06 (2006)
16. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications. In: BPMN 2012. Springer (2012)
17. Kopp, O., Breitenbücher, U.: Choreographies are Key for Distributed Cloud Application Provisioning. In: Proceedings of the 9$^{\text{th}}$ Central-European Workshop on Services and their Composition. CEUR-WS.org (2017)
18. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall PTR (2000)
19. OASIS: Web Services Business Process Execution Language Version 2.0 (2007)
20. OASIS: TOSCA Simple Profile in YAML Version 1.2 (2019)
21. OMG: BPMN Version 2.0. Object Management Group (OMG) (2011)
22. Petcu, D.: Multi-Cloud: expectations and current approaches. In: 2013 International Workshop on Multi-Cloud Applications and Federated Clouds. ACM (2013)
23. Saatkamp, K., Breitenbücher, U., Kopp, O., Leymann, F.: Topology splitting and matching for multi-cloud deployments. In: CLOSER 2017. SciTePress (2017)
24. Soldani, J., Binz, T., Breitenbücher, U., Leymann, F., Brogi, A.: ToscaMart: A method for adapting and reusing cloud applications. Journal of Systems and Software **113** (2016)
25. Weerasiri, D., Barukh, M.C., Benatallah, B., Sheng, Q.Z., Ranjan, R.: A Taxonomy and Survey of Cloud Resource Orchestration Techniques. ACM Computer Surveys **50**(2) (2017)
26. Wieringa, R.J.: Design science methodology for information systems and software engineering. Springer (2014)
27. Wurster, M., et al.: The Essential Deployment Metamodel: A Systematic Review of Deployment Automation Technologies. SICS (2019)