Institute of Architecture of Application Systems

Modeling and Execution of Data-aware Choreographies: An Overview

Michael Hahn, Uwe Breitenbücher, Oliver Kopp, Frank Leymann

[1]Institute of Architecture of Application Systems, University of Stuttgart, Germany
{hahnml,breitenbuecher,leymann}@iaas.uni-stuttgart.de
[2]Institute of Parallel and Distributed Systems, University of Stuttgart, Germany
kopp@ipvs.uni-stuttgart.de

BibTeX:

```
@Article{Hahn2017,
  author    = {Hahn, Michael and Breitenb{\"u}cher, Uwe and Kopp, Oliver and
               Leymann, Frank},
  title     = {Modeling and execution of data-aware choreographies:
               an overview},
  journal   = {Computer Science - Research and Development},
  year      = {2017},
  month     = {Sep},
  doi       = {10.1007/s00450-017-0387-y},
  issn      = {1865-2042},
  publisher = {Springer Berlin Heidelberg}
}
```


University of Stuttgart
Germany

Noname manuscript No.
(will be inserted by the editor)

Modeling and Execution of Data-aware Choreographies

An Overview

Michael Hahn · Uwe Breitenbücher · Oliver Kopp · Frank Leymann

Received: date / Accepted: date

Abstract Due to recent advances in data science and Big Data the importance of data is increasing. Although service choreographies provide means to specify complex conversations between multiple interacting parties from a global perspective and in a technology-agnostic manner, they do not fully reflect the paradigm shift towards data-awareness at the moment. In this paper, we discuss current shortcomings such as missing support for data flow across services and a choreography data contract all interacting parties agree on. This results in more complex and rigid choreography models, making them also less flexible regarding their data perspective during run time. The main contribution is our approach for modeling and execution of *data-aware service choreographies* towards increasing the level of data awareness in choreographies.

Keywords Service Choreographies · Data-awareness · Cross-Partner Data Flow · Transparent Data Exchange

M. Hahn
Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Germany
E-mail: michael.hahn@iaas.uni-stuttgart.de

U. Breitenbücher
Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Germany
E-mail: uwe.breitenbuecher@iaas.uni-stuttgart.de

O. Kopp
Institute for Parallel and Distributed Systems (IPVS)
University of Stuttgart, Germany
E-mail: oliver.kopp@ipvs.uni-stuttgart.de

F. Leymann
Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Germany
E-mail: frank.leymann@iaas.uni-stuttgart.de

1 Introduction

Service-oriented architectures (SOA) have seen wide spread adoption. The concept of composing small, self-contained units of functionality, services, over the network has found application in many research areas and application domains [26]. For example, in Business Process Management (BPM), Cloud Computing, the Internet of Things (IoT), eScience and in particular scientific workflows. To compose multiple existing services into service compositions multiple modeling languages evolved. These languages can be divided into two general groups each following a different paradigm: service orchestrations and service choreographies. Service orchestrations, also known as workflows or processes, are modeled from the viewpoint of one party that acts as a central coordinator [5]. The most prominent orchestration modeling languages are the Business Process Management Notation (BPMN 2.0) [20] and the Business Process Execution Language (BPEL) [19] which provide means to specify orchestration-based service compositions. In contrast, service choreographies provide a global perspective of the potentially complex conversations between multiple interacting services without relying on a central coordinator. Each party that takes part in the collaboration, as a so-called participant, is able to model its conversations with other involved parties by specifying corresponding message exchanges with them [5]. To model service choreographies modeling languages such as BPMN 2.0 or BPEL4Chor [12] can be used.

With advances in the fields of Big Data and IoT the importance and value of data is increasing significantly [17, 23]. However, current state of the art in service choreographies, despite some promising works trying to improve data-awareness [9, 13, 16], fails to provide an overall solution that allows data to assume

its deserved primary role for both modeling and execution of service choreographies. To tackle this problem and account for the crucial importance of data in service choreographies the traditional BPM life cycle was extended by data management capabilities [8]. The inherent goal is to introduce data already on the level of the choreography to enable the decoupling of data flow, data exchange and data management from the control flow in service choreographies and orchestrations.

In this work, we want to have a closer look at the proposed modeling extensions and discuss challenges regarding the data-awareness of current service choreography modeling languages as well as potential solutions and their implications towards the run time perspective of service choreographies. Therefore, in Sect. 2, we motivate our work followed by a discussion on shortcomings of current choreography modeling languages and run time environments. Based on that, in Sect. 3, we introduce our Transparent Data Exchange (TraDE) approach to tackle the identified shortcomings by introducing new modeling capabilities and a middleware towards our goal of supporting data-aware service choreographies. To exemplify the application of our concepts, we introduce a corresponding case study from the domain of eScience in Sect. 4. Furthermore, we outline the underlying architecture of a modeling and run time environment capable of supporting the proposed modeling extensions and their execution in Sect. 5 and our prototypical implementation in Sect. 6. Finally, the paper concludes with related work in Sect. 7, and a summary of our contributions together with an outlook on future work in Sect. 8.

2 Motivation and Discussion

To further illustrate and motivate the need to improve data-awareness in service choreographies we first present a motivation example followed by a discussion on shortcomings regarding the modeling and execution of service choreographies.

Figure 1 shows a choreography model with two interacting participants. The conversations between the participants are modeled by message intermediate events and message flows. Data is modeled by data objects and the reading and writing of data objects from tasks and events is specified through data associations. Using this example, we discuss shortcomings of the current state of the art in choreography modeling.

We use BPMN 2.0 as a basis for the following discussion and also to illustrate our concepts. However, our concepts are not bound to BPMN 2.0 and can be applied to other choreography modeling languages such as BPEL4Chor. The only assumption is, that the underlying choreography modeling language follows the *interconnected interface behavior* modeling approach [5]. This means, that the choreography model specifies control flow per participant and the interactions between participants through message links. Figure 1 shows an example of such a interconnected interface behavior model illustrated as BPMN 2.0 collaboration model. A choreography model specifies a public model of a collaboration which is then transformed into a collection of private process models specifying internal logic of each participant while implementing the conversations defined on the level of the choreography model [6].

As a result, not only the data-awareness and data modeling capabilities of the used choreography modeling languages are important, we also have to take process modeling languages into account. Exactly for this purpose, Meyer et al. [15] already evaluated business process modeling languages with respect to their level of data-awareness and data modeling capabilities. Therefore, in the context of this work, we reuse their results as a basis to discuss and identify further issues based on the introduced motivation example depicted in Fig. 1 with focus on the level of choreographies.

One of the major shortcomings, is that data flow across participants (*inter-participant* data flow) has to be modeled differently compared to the data flow inside a participant (*intra-participant* data flow). For example, in BPMN 2.0 inter-participant data flow can be expressed intuitively by drawing data associations between data objects and tasks as shown in Fig. 1 between the message start event s1 and data object D of participant P1. However, to model the exchange of data between two or more participants, data associations are not allowed anymore and message flow has to be introduced (e. g., x1 in Fig. 1). While from an execution point of view it is clear that data exchange across participant boundaries needs to be handled differently than local data exchange, the question is if this has to be explicitly specified in choreography models. Introducing message flow to exchange data between participants requires the specification of a whole set of additional modeling constructs and unnecessarily couples the data flow to the control flow in choreographies. First, a message has to be specified that will transport the data between the participants during run time. Second, the modeler has to add a corresponding message send task or event (e. g., t1 in Fig. 1) with data associations on data objects containing the data to exchange, in order to encapsulate data into a message and send it to another participant. On the side of the receiving participant, the modeler has to add a message receive task or event (e. g., s2 in Fig. 1) to consume the message and specify data asso-

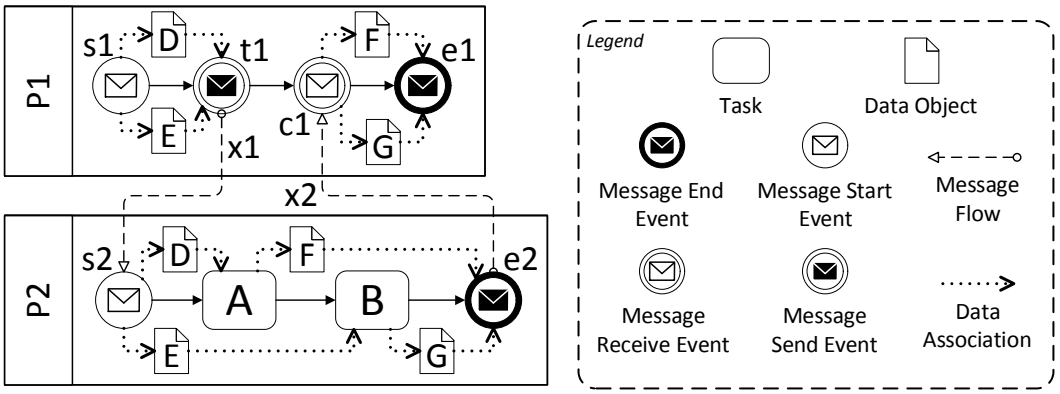

Figure 1 Example choreography illustrated as BPMN 2.0 collaboration model.

ciations to one or more data objects where the data contained in a message should be extracted to. Finally, the modeler has to connect the sending and receiving tasks or events with a message flow (e. g., x1 in Fig. 1), to specify the exchange of the previously introduced message. The result is that data produced by one participant is now also accessible at another participant both having their own local, independent copy of the data. We call this *message-based data exchange*. An example is shown in Fig. 1, where data objects D and E are exchanged through a message flow between participant P1 and P2. The key point to take away is that data cannot be exchanged between participants without introducing additional control flow and modeling constructs in order to exchange the data. As we can see in Fig. 1, what we discussed is a rather simple conversation scenario. The overhead of introducing additional modeling constructs in order to share data between participants will increase, e. g., if multiple participants require the same data or received data has to be routed to other participants.

Another drawback of message-based data exchange is the fact that the same data objects, or slight variations of them, need to be specified in the scope of each participant that interacts with the data. This is also directly visible in Fig. 1 where each of the participants defines the same four data objects. However, to technically enable the collaboration between different participants through a choreography they have to agree on the structure of the exchanged data. This leads to the question if the underlying choreography modeling language allows the specification of a common, globally consolidated and agreed set of data objects representing a data contract between the collaborating parties. Often, this is not supported and therefore modelers have to agree on structures without being able to specify the outcome explicitly and in a central, reusable manner within a choreography model. As a result, each party has to take care of specifying the required set of data objects in the context of all participants it is responsible for while being compliant with the agreed structures. The issue is that if these structures change over time, there is no central point in a choreography model to do this and changes have to be manually reflected at all affected data objects specified in the context of several participants interacting with this data. This makes changes error-prone and might lead to inconsistent specifications of data objects. Furthermore, the data required and produced by the choreography as a whole and of each participant individually has to be identified by analyzing the model instead of being directly visible through its graphical representation. Another potentially useful capability not supported at the moment, is that modelers should be able to express that multiple data objects (semantically) belong together. This will improve the visual expressiveness of the models regarding their data perspective [15].

Moreover, binding cross-partner data flow to message exchanges and therefore coupling it with the control flow disallows the independent evolution of data and control flow in choreography models. This coupling potentially also results in unnecessary routing of data and blocking of control flow of participants while data is exchanged. The former means that data might be passed through several consecutive message flows across participants instead of directly exchanging it in a peer-to-peer manner as soon it is available. The latter point addresses the fact that while the data is exchanged through message flow the receiving participants are blocked until the message arrives. This is actually the result of an inherent trade-off modelers have to make. On the one hand, they can improve the data flow by introducing additional message-based data exchanges to exchange the data in a peer-to-peer manner and as soon as possible. On the other hand, modelers can try to keep the number of message-based data exchanges minimal and only pass all required data to another participant at once, if the exchange of a message is anyhow required. The first choice results in more complex and conversation-

intensive models but reduces the time participants have to wait for required data. The second choice results in less complex and less conversation-intensive models but increases the time participants have to wait for required data because data is only exchanged in blocks at certain points in time.

To sum up our discussion, when following the message-based data exchange approach the exchange of data during run time has to be specified completely upfront at modeling time. Therefore, during run time it is hard to optimize data exchange between participants since all data exchanges are strictly expressed through message flows. The only way to realize (more) dynamic data capabilities using the message-based approach is to introduce corresponding control flow logic already on the level of the choreography models. The main drawback of this approach is that the models are polluted with data management functionality that is not relevant from a business perspective. Therefore, we are arguing that such generic data management and exchange functionality should be provided transparently by the run time environment without the necessity to be explicitly modeled in a choreography model.

3 The TraDE Approach

In this section, we introduce our concepts for modeling and execution of data-aware choreographies through TraDE. Therefore, we first have a look at the modeling perspective and introduce the notion of cross-partner data objects and data flow and how they tackle the shortcomings discussed in Sect. 2. Based on that, we will introduce our proposal for required run time support for these new modeling constructs through a new TraDE middleware component.

3.1 Modeling

Figure 2 shows our motivation example choreography model with our concepts already applied. To enable the specification of choreography data, representing a data contract between the participants within a choreography model, we introduce a *choreography data model* (CDM). A CDM provides the foundation for data-awareness and data-related capabilities in choreographies. It enables to specify required data and its structures in a self-contained and consolidated manner at a central location in a choreography model. Therefore, a CDM consists of a set of *cross-partner data objects* that express the commonly agreed data of a choreography shared by and accessible from all participants. To avoid confusion between BPMN 2.0 data objects and cross-partner

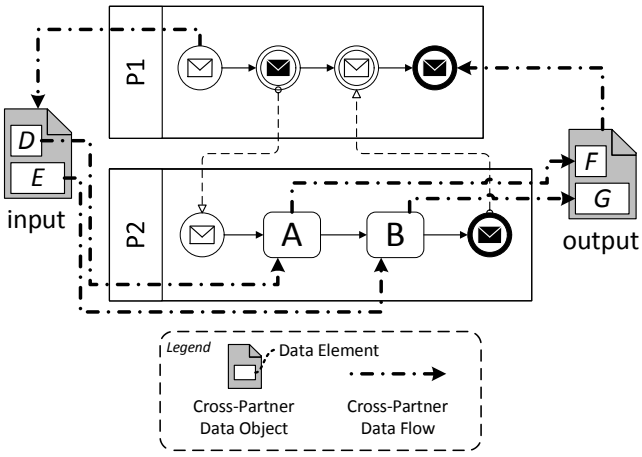

Figure 2 Example choreography with TraDE concepts applied.

data objects as a general concept, we use the term *data container* as a language-independent name for a modeling construct that allows the specification of data, e. g., BPMN 2.0 data objects or BPEL variables. The model of Fig. 2, makes it explicit which data the overall choreography as well as each of its participant requires or produces. The major advantage is that data required by multiple participants of a choreography can be itself modeled less redundantly. Moreover, its exchange can be expressed more intuitively using cross-partner data flow instead of specifying messages and corresponding tasks or events to process them. Although each participant can still have its own local data containers, cross-partner data objects allow to model shared data which is associated to the whole choreography instead of a single participant. By using cross-partner data objects, data containers have to be modeled only once which normally need to be modeled multiple times in the scope of different participants on the level of the choreography.

A cross-partner data object has a unique identifier and contains one or more *data elements*. A data element has a unique name, from the scope of its surrounding data object, and a reference to a definition of its structure, e. g., using a simple, build-in type system or XML Schema Definitions[1]. The idea of this single level of nesting is that cross-partner data objects can be seen as named envelopes for a collection of typed data containers, namely data elements, which semantically belong together. Therefore, data elements hold the corresponding data values during run time. We distinguish between cross-partner data objects that can be instantiated only once (single-instance) or that hold a collection of values (multi-instance) during run time. Where multi-instance cross-partner data objects are useful to hold a collection of identically structured values that can be processed

[1] W3C, XML Schema Definition Language (XSD) 1.1 Part 1: Structures. Online: http://www.w3.org/TR/xmlschema11-1/

with the same sequence of tasks through a loop, while the number of loop iterations is dynamically bound to the size of the collection during run time.

Based on top of the notion of cross-partner data objects, we introduce cross-partner data flow. The idea is that modelers should be able to intuitively specify data flow within and across participants. Furthermore, cross-partner data flow allows decoupling the exchange of data from the exchange of messages and therefore from choreography control flow as shown in Fig. 2. As participant internal data flow, cross-partner data flow also supports the specification of transformation logic, selective queries and mappings, e. g., to read or write only parts of a data element or cross-partner data object or multiple data elements at once. The only difference between intra-partner and cross-partner data flow is how the specified data flow is conducted during run time, i. e., if the process engine executing a participants private process or TraDE middleware is handling the data exchange.

Regarding the modeling perspective, these two extensions, i. e., cross-partner data objects and data flow, are enough to provide us the basis for an easier and more intuitively specification and handling of data in choreographies.

3.2 Refinement

At the end of the choreography modeling phase, we are transforming modeled choreographies to a collection of interconnected process models in order to execute them [5]. The idea is to translate the introduced cross-partner data objects into standard data containers on the level of the private process models again. For example, using data objects in BPMN 2.0 or variables in BPEL. Using the specified choreography data model as input, this can be done in an automated manner. Based on that, we are able to reduce manual refinement efforts on the level of the resulting private process models by leveraging provided data-related knowledge to generate more complete process models. The overall goal is that modelers refining the private processes should not need to know nor distinguish between local or globally shared data containers. From the viewpoint of each private process model, there is no difference between a data container that is defined locally or globally. As a result, during refinement modelers can extend the process model with additional private control flow and data flow as usual.

To reflect the extensions to model the context to which a data container belongs (data element of a cross-partner data object) and to express if a data container is defined locally or linked to a cross-partner data object on the level of a process model, the modeling construct for data containers of the underlying process language has to be extended. In the case of BPMN 2.0 and BPEL the extensibility of the languages can be used to introduce new attributes and elements that can be associated with the corresponding modeling constructs for data objects or variables, respectively. These language extensions can then be used by the process engine in order to communicate with the TraDE middleware to realize the modeled cross-partner data flow.

Figure 3 presents the refined example for the choreography model from our motivation scenario. In the *Choreography Modeling Environment* we can see the data-aware choreography model which specifies two cross-partner data objects and corresponding data flow. The transformed and refined private process models of the two participants are deployed to corresponding process engines as depicted in the middle of Fig. 3. Each of the two private process models contains corresponding BPMN data objects that contain a link to the cross-partner data object they represent (indicated by *). Whenever the process engine reads or writes data from or to these data objects, it invokes corresponding functionality exposed by the TraDE middleware to query or forward shared data from or to the middleware, respectively. This allows us to share and exchange data across multiple interacting parties as modeled through the choreography but completely independent and decoupled from message flow. The resulting data containers are only placeholders referring to the actual cross-partner data objects managed by the TraDE middleware outside of the process engine. Therefore, the process engines have to be extended in order to integrate with the TraDE middleware and to support such placeholder data containers.

3.3 Middleware

By the TraDE middleware we want to introduce new degrees of freedom regarding the data perspective of service choreographies. In general, TraDE acts as a middleware layer supporting an easier management, exchange, and provisioning of shared data independent of its processing within a service choreography or orchestration. Therefore, all cross-partner data objects are exposed in a web-accessible manner through a REST API. The major advantage is that each data object is represented as a resource and can therefore be easily accessed, referenced, and shared with others through a Uniform Resource Locator (URL).

This is especially important for scientists who should be supported with an accepted way to share data of their simulation choreographies independent of the life

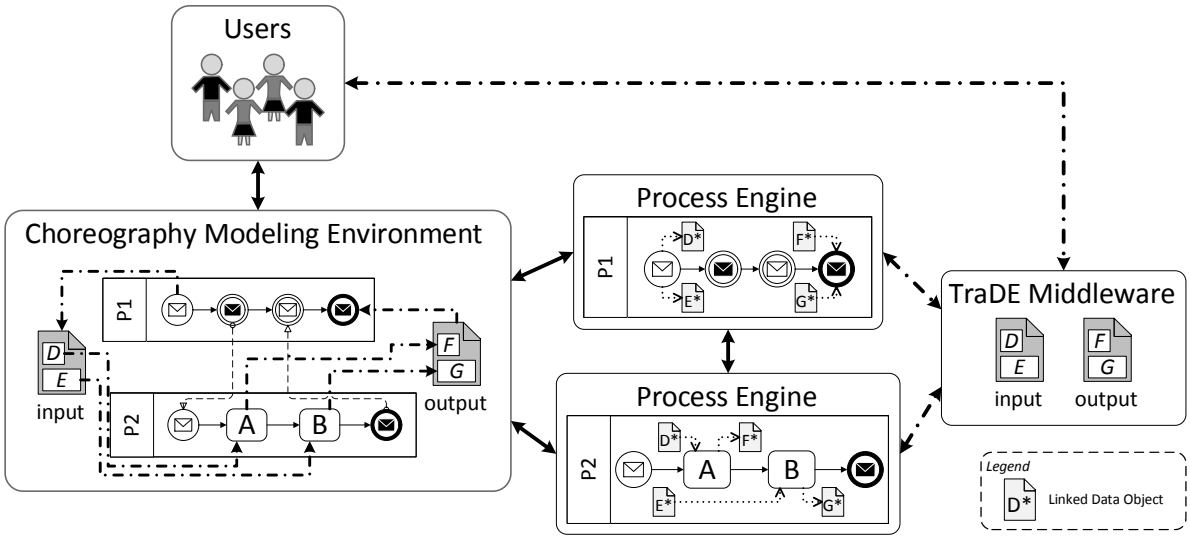

Figure 3 Run time environment for data-aware choreographies with deployed motivation example choreography from Fig. 2.

time of a simulation instance. The TraDE middleware will enable scientists to upload and provide simulation input data, inspect and observe intermediary results during the execution of a simulation (depicted in Fig. 3 by the arrow between the users and the middleware) or enable the reuse of data from previous simulation runs by simply provide a reference in form of a URL to it.

Outsourcing the data to the TraDE middleware decouples the life time of the data from the underlying process instances and from the availability of the process engines. This allows an easier reuse of data across multiple instances of the same choreography model or even across different choreography models that require common data. Especially the fact that we are able to share the data not only in the context of a choreography and its interacting services, opens up completely new ways of how data can be processed, provided and managed in service choreographies. For example, using other data-centric tools and systems to process or transform the data in parallel to the execution of a choreography to allow their use for other purposes. The logic can therefore be triggered in an event-based manner using the TraDE middleware as coordinator.

In contrast to the previously outlined positive effects and advantages of introducing cross-partner data flow and decoupled data exchange through the TraDE middleware, this causes also some negative side effects. Whenever something is shared in a distributed context, concurrency issues will arise. Since in our case data and control flow are running in parallel, the probability for concurrent access of shared data from different, potentially not synchronized participants is much higher than in classical scenarios. Therefore, modelers have to pay attention to concurrency issues when specifying cross-partner data flow. In future work, we will conduct a thorough analysis of potential concurrency issues and cases for which we have to provide corresponding mechanisms based on the state of the art.

4 Case Study

This section presents a case study from the domain of eScience as an example how to apply our concepts to an existing choreography model.

Figure 4 shows a choreography model of a Kinetic Monte Carlo (KMC) simulation using the custom-made simulation software *Ostwald ripening of Precipitates on an Atomic Lattice* (OPAL) [4]. OPAL simulates the formation of copper precipitates, i. e., the development of atom clusters, within a lattice due to thermal aging. The simulation consists of five major building blocks which are reflected as participants of the choreography depicted in Fig. 4. Following the choreography paradigm, the conversations between the five participants are specified in a peer-to-peer manner through the exchange of messages. The consumed and produced data is represented through BPMN data objects (DO). To exchange data between participants the corresponding message flow is specified. The labels on the message flows are added to visualize which data objects of a participant are exchanged during a conversation through a message. The invocations of the different modules of the OPAL simulation software are modeled through corresponding service tasks. Since the OPAL simulation software was originally programmed in Fortran using file-based data exchange and not having service orientation in mind, the software is provided as a set of executables each representing one module of OPAL. To provide these

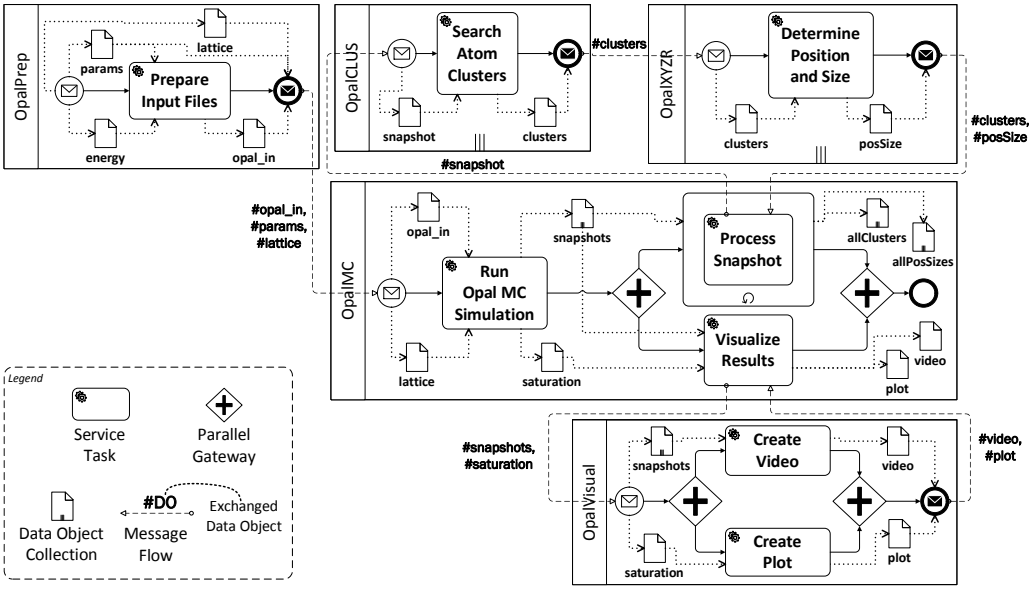

Figure 4 Choreography conducting a thermal aging simulation from material science domain.

executables as services, they are manually wrapped to enable their service-based invocation [24].

Whenever the *OpalPrep* participant receives a new request message, a new simulation instance is created. The initial request contains a set of parameters (*params* DO), e. g., the number of snapshots to take, an initial energy configuration (*energy* DO) and a lattice (*lattice* DO). The *Prepare Input Files* service task sends the parameters and the energy configuration to a service which takes the data to consolidate and transform it into the right input format (*opal_in* DO) for the KMC simulation. After that, a message containing the *opal_in*, *params* and *lattice* data is send to the *OpalMC* participant to start the actual KMC simulation.

The *Run Opal MC Simulation* service task invokes the corresponding service which conducts the KMC simulation based on the provided data. According to the specified number of snapshots in *params*, the service saves the current state of the atom lattice at a particular point in time as a snapshot and replies all snapshots together (*snapshots* collection DO) as well as saturation data (*saturation* DO). After that, the snapshots are analyzed and visualized in parallel. Based on the number of snapshots, multiple instances of the *OpalCLUS* and *OpalXYZR* participants are created through the *Process Snapshots* service task. Each instance of the two participants is analyzing one particular snapshot. The *OpalCLUS* participant takes a snapshot and invokes a corresponding service (*Search Atom Clusters* task) which replies identified clusters (*clusters* DO) of the snapshot. This cluster information is then forwarded to the *OpalXYZR* participant which invokes a service (*Determine Position and Size* task) capable of identifying the position and size of each cluster (*posSize* DO). After that, the resulting cluster and position data is replied back to the *Process Snapshot* task of the *OpalMC* participant which collects the instance results into collection data objects, i. e., *allClusters* and *allPosSizes* DOs.

In parallel to analyzing the snapshots, the *Visualize Results* service task triggers the visualization of the snapshot and saturation data through the *OpalVisual* participant. The required data is passed through a message and then used by the *Create Video* and *Create Plot* service tasks to invoke corresponding services. Based on the collection of snapshots, a video of animated 3D scatter plots (*Create Video* task) is created and the saturation data is used to create a 2D plot of the saturation function of the precipitation process (*Create Plot* task). Finally, the resulting media data is replied back to the *OpalMC* participant which then completes the execution of the simulation instance.

Figure 5 presents the same choreography using the TraDE paradigm. All data relevant for the choreography model is specified independently of any participant and in a shared and reusable manner through cross-partner data objects. This makes the data required and produced by the choreography and each of its participants more visible and easier to identify by human readers. Furthermore, this also reduces the amount of data objects and participant internal data flow leading to less complex graphical models. The grouping of data elements directly visually reflects data that is semantically related and belongs together, as for example the *sim_input* data object containing all data elements providing simulation input data. Another benefit of this grouping is that the whole collection of data elements

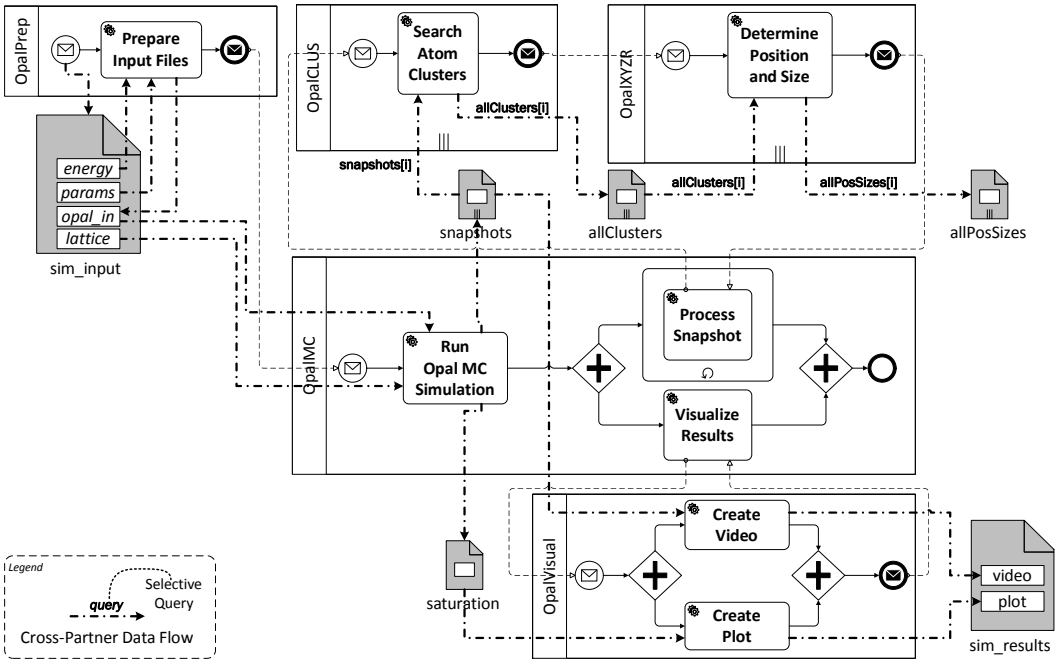

Figure 5 Data-aware OPAL simulation choreography after applying TraDE concepts.

can be specified as the source or target of a data flow. For example, the data flow between the start activity of the *OpalPrep* participant and the *sim_input* data object specifies that the request message contains data for multiple data elements. The data flow therefore specifies a set of mappings how the data contained in the message should be extracted and assigned to the respective data elements. Furthermore, selective queries can be attached to the cross-partner data flow in order to specify that only parts of a cross-partner data object or even data element are required. For example, each instance of the *OpalCLUS* participant processes one snapshot from the whole collection of snapshots (*snapshots* cross-partner data object) which is specified through the selective query attached to the data flow (*snapshots[i]*).

The routing of data through consecutive participants is improved compared to the classical version of the model shown in Fig. 4. Instead of routing cluster data from the *OpalCLUS* participant through the *OpalXYZR* participant to the *OpalMC* participant, with the data-aware paradigm it is directly stored in the globally shared *allClusters* cross-partner data object. The advantage is that other services requiring this data can directly retrieve it from the shared data object instead of waiting for a corresponding message. Furthermore, by using the introduced capabilities of the data-aware paradigm, scientists are able to inspect each of the identified clusters as soon as they are added to the *allClusters* data object while the processing of the remaining snapshots is still running. Instead of sending all required input data encapsulated in the initial request message to trigger a new simulation instance, scientists are also able to upload data such as the initial lattice to the TraDE middleware beforehand and then only pass a reference to this data within the request message. This is especially useful if the same data is used for multiple simulation runs, e. g., in the context of a parameter study where it does not make sense to send the same data multiple times over the network, if this is not really required in terms of other reasons. The same applies the other way around, e. g., for the resulting video. Instead of routing data through several participants, the resulting video will be directly stored from the *OpalVisual* participant to the shared *sim_results* data object from where it can then be downloaded through the TraDE middleware even after the simulation instance is completed.

5 System Architecture

Figure 6 presents the overall architecture of the software system enabling the modeling and enactment of data-aware service choreographies. Each participant has a *Choreography and Orchestration Modeling Environment* to model his part of the overall choreography. The modeling environment supports the transformation of the choreography model to a set of private process models where each of them is further refined by a modeler to an executable process model. The introduced TraDE facilities are integrated into the modeling environment and enable the specification of a choreography data model

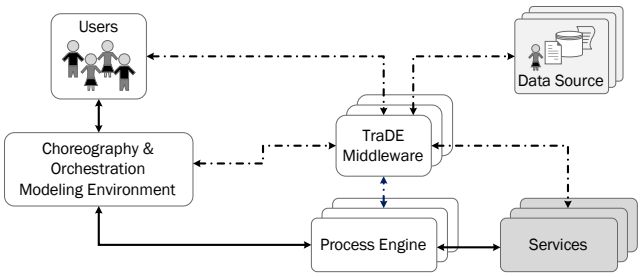

Figure 6 System architecture of a modeling and run time environment for data-aware choreographies supporting the TraDE approach.

with cross-partner data objects and data flow and their transformation to the level of the process models.

To enable the execution of a choreography its private process models are deployed to process engines. The choreography model is in addition deployed to the TraDE middleware to provide the shared data of the choreography. The TraDE middleware is therefore not a single centralized component, but rather a collection of multiple independent and decentralized nodes forming a distributed system where the whole network of nodes is experienced as one single coherent middleware. The overall goal is to efficiently place and provide data and to use the TraDE middleware as a platform to apply for further optimizations on data flow across participants during run time. For example, data can be pro-actively transferred, as soon as it is available, between participants based on available knowledge extracted from the models (e. g., analyzing data dependencies) or auditing information from previous choreography executions. The idea is that data are already available as near to the target participant as possible and before the corresponding control message is exchanged between the participants.

During choreography execution, the process engines responsible for the participant processes, conduct the modeled conversations through the exchange of messages that transport data or invoke corresponding services. In addition, the process engines are also communicating with the TraDE middleware in terms of handling the modeled cross-partner data flow between participants. Furthermore, as depicted by the dashed lines in Fig. 6, the TraDE middleware also allows users and other services to access and provision data through its API. In terms of services, this allows a process engine to pass data by reference in a request message, in case the corresponding service is aware of the TraDE middleware. Users will be able to upload their data for its processing within a choreography or process instance as well as to inspect data available at the middleware. To support the retrieval and storing of data of cross-partner data objects from and to heterogeneous external data sources, existing concepts and mechanisms [16, 22] can be integrated into the TraDE middleware. The integration of corresponding functionality into the TraDE middleware improves the provisioning and persistence of shared data since data can then be automatically retrieved from and stored into various data sources.

6 Prototypical Implementation

We are building on existing tools: The *Chor Designer* [25] and an extended version of the *BPEL Designer*[2] both being Eclipse-based graphical editors for BPEL4Chor and BPEL, respectively. The resulting Eclipse-based modeling environment also provides required model transformation logic [21] in order to generate a collection of BPEL process models out of a given BPEL4Chor choreography model.

An extended version of the open source BPEL engine Apache *Orchestration Director Engine* (ODE)[3] is used as process engine. To support the reading and writing of cross-partner data objects, we extended the underlying implementation of Apache ODE and integrated it with our TraDE middleware.

The TraDE middleware itself is a Java-based web server which exposes its functionality through a REST API. At the moment, we support a single-node deployment of the TraDE middleware, but for future work, we are planing to support also multi-node deployments.

7 Related Work

Our focus is on existing works following also the goal of improving data-awareness and data-related capabilities for the modeling and execution of choreographies. Therefore, we follow the arguments of Meyer et al. [15] that data-related aspects or data-awareness in general should only be supported to the degree actually required in the domain or scope in which a modeling language is used. Since our focus is on improving and extending the role of data in classical control flow driven choreography and process modeling languages such as BPMN, BPEL4Chor, and BPEL, we focus on corresponding related work following the same paradigm. This means the specification of control flow remains the main part of choreography and process modeling while further support for data and data flow modeling is added. However, for the sake of completeness, we also shortly outline

[2] The Eclipse Foundation, BPEL Designer Project: `http://www.eclipse.org/bpel/`

[3] The Apache Software Foundation, Apache ODE: `http://ode.apache.org`

other potentially related approaches yet following another paradigm and therefore not discussed in detail in the context of this work.

Modeling languages following the *artifact-centric* modeling paradigm consider business data and how it evolves and is changed within a process as the main driver. Therefore, the focus changes from modeling control flow with associated data to modeling data with associated control flow, specifying the actions performed on data. Examples for corresponding artifact-centric modeling approaches are *business artifacts* [18] or *case handling* [1].

Lohmann and Wolf [14] apply the paradigm of business artifacts on the level of choreographies to model collaborations from a data perspective. Therefore, they provide a systematic approach to specify relevant data as artifacts and a set of agents that are operating on that data in the context of a collaboration. By enhancing artifacts with location and remote access information they are then able to derive an overall interaction model. Furthermore, Lohmann and Nyolt [13] investigate to what extend BPMN can be used or requires extensions in order to support modeling of artifact-centric processes and choreographies. Although we support the author's arguments to make data more prominent and increase modeling support in choreography and process models, we still rely on control flow as the main driver for processes and related established standards such as BPMN and BPEL.

Knuplesch et al. [9, 10] introduce the notion of data-aware process interaction models as a means to model data-aware choreographies. Their goal is to enrich interaction models with a data perspective as a means to explicitly consider data being exchanged through messages between participants and also used for routing decisions while ensuring correctness of the resulting models. Therefore, they define a formal framework and specific correctness criteria for *Data-Aware Choreographies* (DAChor) while their behavior is described using elements of *Interaction Petri Nets* and *Workflow Nets with Data*. Although the authors concentrate on the modeling and correctness of data-aware choreographies, our focus is more on the improvement of data-awareness during run time. Similar to our cross-partner data objects, the authors introduce so-called *virtual data objects*. However, the exchange of actual business data is still message-based and virtual data objects only allow to share states across participants to support data-based routing decisions.

The model-driven approach by Meyer et al. [16] supports the modeling and enactment of data exchange in choreographies using messages. The authors propose an extension of the BPMN modeling language by introducing annotations on BPMN data objects which are then automatically transformed into SQL queries to specify and enact message extraction from and message storage to local databases. This enables the complete automation of data exchange between participants and the enrichment of model transformations with data-related aspects. We fully support the authors arguments that the collaborating partners should specify the exchanged data and its structure in a commonly agreed global data model already on the level of a choreography model. However, instead of directly binding data objects to databases on the level of the models, our approach introduces the TraDE middleware as an abstraction layer. This allows us to support also other domains, e. g., eScience where data is commonly stored and exchanged through files in different formats. Moreover, we decouple the exchange of data from the exchange of messages to simplify modeling and improve run time flexibility of choreographies regarding their data perspective.

Barker et al. [2] define MAP as new language for executable service choreographies. For the enactment an open source framework and the concept of *peers* is introduced. A peer provides extra functionality that enables web services to participate in a choreography without requiring to adapt the underlying service implementations. The main difference is the introduction of a new modeling language and run time environment instead of building on top of existing standardized languages and tools. Furthermore, Barker et al. [3] introduce the *Circulate approach* that combines the advantages of both paradigms: orchestration and choreography. Although the control flow remains orchestration-based, the data flow is conducted in a choreography-based manner. In order to enable services to transfer data between each other, *proxies* are introduced to provide the required functionality. Therefore, a proxy acts as an intermediary between the process engine and the actual services during service invocations. Consequently, the process engine triggers the invocation of services and the exchange of data between services through a proxy. In general, we are following a similar approach by applying the public-to-private technique and introducing the TraDE middleware to decouple the data flow from the control flow. However, instead of explicitly modeling the invocation of proxies and data exchange through them, we propose to introduce cross-partner data objects and data flow. Translated to corresponding annotations on the level of process models, the process engine is then able to transparently enact the data flow together with the TraDE middleware which serves as data flow coordinator. As a result, process models are enriched instead of changed and the coordination of cross-partner data flow is outsourced to the TraDE middleware instead

of explicitly specified in process models. The former preserves the portability of the models on run time environments without TraDE support and the latter provides more flexibility and optimization possibilities during run time.

An approach similar to some of our discussed ideas, but with focus on the level of process models and BPEL in particular, is provided by Habich et al. [7]. They try to overcome the issue of centralized and only implicitly specified data flow in BPEL through variables and assign activities and the resulting *by value* semantics of data exchange. Therefore, they combine their concept of Data-Grey-Box Web Services with an extension of BPEL through so-called *BPEL data transitions*. The former allow to enhance web service interfaces with an explicit data aspect allowing the separation of parameters passed by value and data passed by reference. The latter support the annotation of BPEL processes with explicit data flows between the composed Data-Grey-Box Web Services. Both concepts together allow to integrate specialized data propagation tools and logic, e. g., specified using Extract Transform Load (ETL) tools, as a means to implement the specified data transitions and act as mediators between Data-Grey-Box Web Services during run time to provide and resolve data by reference. Although the authors propose to introduce explicit data flow on the level of BPEL, we argue that data flow can be specified easier and more intuitive on the level of a choreography showing all participants of a service composition and not only a centralized view as on the level of BPEL or process models in general. Furthermore, while we also aim at supporting the exchange of data by reference, our overall goal is to hide as much as possible of the data flow related logic on the level of the process models by outsourcing the required functionality to the TraDE middleware. Therefore, the TraDE middleware can be used to propagate and resolve data by reference as well as an integration layer for specialized data propagation tools and logic.

The essential flow model [11] tackles the issue that modelers have to decide early which paradigm they follow when modeling collaborations. By following the orchestration paradigm the whole collaboration is specified within one consolidated process model where all tasks are connected through control flow. By following the choreography paradigm the collaboration can be split into multiple interacting process models where tasks from different models are connected through message flow. This imposes not only restrictions during modeling, but also on the IT infrastructure for the execution of the resulting collaboration models. The notion of essential flow models allows to defer the decision how to split and organize a collaboration to a later phase. It enables modelers to specify their collaborations by modeling essential flows between tasks and responsible partners. Based on that, an essential flow model can be mapped and implemented in different ways, e. g., as orchestration or choreography, taking the target IT infrastructure into account. The authors idea of removing the burden from modelers to distinguish between and decide on local or remote control flow, i. e., message flow, is similar to our approach regarding data flow. Although the authors do not take data flow into account, the notion of essential flow models will be useful as an abstraction technique and entry point for specifying data-aware service choreographies.

8 Conclusions and Outlook

The importance of data-awareness and its effect on the BPM domain is increasing. A more prominent role of data in service compositions is therefore a must in order to benefit from these developments. In this work, we motivated and discussed our vision of data-aware service choreographies aiming at improving data modeling capabilities and data-awareness during both modeling and run time. The results of the discussion are used as a basis for our proposal of corresponding modeling extensions, namely cross-partner data objects and data flow. Since most of the existing approaches concentrate on modeling or execution aspects only, our goal is to provide an end-to-end approach together with a supporting modeling and run time environment for data-aware service choreographies. Therefore, we introduced a corresponding system architecture which supports the proposed modeling extensions and new run time capabilities of data-aware service choreographies followed by a brief description of our prototypical implementation of this architecture.

In future work, we are planning to support distributed, multi-node deployments of the TraDE middleware towards the goal of identifying and enabling further data flow optimization possibilities. Since this increases the level of concurrency from a data perspective, we therefore will conduct a thorough analysis of potential issues and cases for which we have to provide corresponding synchronization and scheduling mechanisms. Furthermore, we plan to provide a Web UI for the TraDE middleware in order to enable its use for human users. Regarding the modeling perspective, we will define a formal framework for data-aware service choreographies and corresponding algorithms to enable modelers to seamlessly translate data-aware choreography models into standards-based models and vice versa.

Acknowledgements This research was supported by SimTech (EXC 310/2) and SmartOrchestra (01MD16001F).

References

1. van der Aalst WMP, Weske M, Grünbauer D (2005) Case handling: a new paradigm for business process support. Data & Knowledge Engineering 53(2):129–162
2. Barker A, Walton C, Robertson D (2009) Choreographing Web Services. IEEE Transactions on Services Computing 2(2):152–166
3. Barker A, Weissman JB, Van Hemert J, et al (2012) Reducing Data Transfer in Service-Oriented Architectures: The Circulate Approach. IEEE Transactions on Services Computing 5(3):437–449
4. Binkele P, Schmauder S (2003) An atomistic Monte Carlo simulation of precipitation in a binary system. Zeitschrift für Metallkunde 94(8):858–863
5. Decker G, Kopp O, Barros A (2008) An Introduction to Service Choreographies. Information Technology 50(2):122–127
6. Decker G, Kopp O, Leymann F, Weske M (2009) Interacting services: from specification to execution. Data & Knowledge Engineering 68(10):946–972
7. Habich D, Richly S, Preissler S, Grasselt M, Lehner W, Maier A (2008) BPEL^{DT} - Data-Aware Extension for Data-Intensive Service Applications. In: Emerging Web Services Technology, vol II, Springer
8. Hahn M, Karastoyanova D, Leymann F (2016) A Management Life Cycle for Data-Aware Service Choreographies. In: Proceedings of ICWS'16, IEEE Computer Society
9. Knuplesch D, Pryss R, Reichert M (2012) Data-aware interaction in distributed and collaborative workflows: Modeling, semantics, correctness. In: 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing, IEEE, pp 223–232
10. Knuplesch D, Pryss R, Reichert M (2012) A formal framework for data-aware process interaction models. Open Access Repository of University Ulm
11. Kopp O, Leymann F, Unger T, Wagner S (2011) Towards The Essential Flow Model. In: Proceedings of ZEUS'11, CEUR-WS.org, CEUR Workshop Proceedings, vol 705, pp 26–33
12. Kopp O, Leymann F, Wagner S (2011) Modeling Choreographies: BPMN 2.0 versus BPEL-based Approaches. In: EMISA 2011, GI, LNI
13. Lohmann N, Nyolt M (2011) Artifact-centric Modeling Using BPmn. In: International Conference on Service-Oriented Computing, Springer, pp 54–65
14. Lohmann N, Wolf K (2010) Artifact-centric choreographies. In: International Conference on Service-Oriented Computing, Springer, pp 32–46
15. Meyer A, Smirnov S, Weske M (2011) Data in Business Processes. Technical Report 50, HPI, University of Potsdam
16. Meyer A, Pufahl L, Batoulis K, Fahland D, Weske M (2015) Automating Data Exchange in Process Choreographies. Information Systems
17. Meyer S, Sperner K, Magerkurth C, Pasquier J (2011) Towards Modeling Real-world Aware Business Processes. In: Proceedings of WoT'11, ACM, pp 81–86
18. Nigam A, Caswell NS (2003) Business artifacts: An approach to operational specification. IBM Systems Journal 42(3):428–445
19. OASIS (2007) Web Services Business Process Execution Language Version 2.0. URL `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`
20. OMG (2011) Business Process Model And Notation (BPMN) Version 2.0. URL `http://www.omg.org/spec/BPMN/2.0/`
21. Reimann P, Kopp O, Decker G, Leymann F (2008) Generating WS-BPEL 2.0 Processes from a Grounded BPEL4Chor Choreography. Technischer Bericht 2008/07, Universität Stuttgart
22. Reimann P, Reiter M, Schwarz H, Karastoyanova D, Leymann F (2011) SIMPL-A Framework for Accessing External Data in Simulation Workflows. In: BTW, Citeseer, vol 11, pp 534–553
23. Schmidt R, Möhring M, Maier S, Pietsch J, Härting RC (2014) Big Data as Strategic Enabler - Insights from Central European Enterprises. In: Business Information Systems, LNBIP, vol 176, Springer International Publishing, pp 50–60
24. Sonntag M, Hotta S, Karastoyanova D, Molnar D, Schmauder S (2011) Using Services and Service Compositions to Enable the Distributed Execution of Legacy Simulation Applications. In: ServiceWave'11, Springer, pp 1–12
25. Weiß A, Andrikopoulos V, Gómez Sáez S, Karastoyanova D, Vukojevic-Haupt K (2013) Modeling Choreographies using the BPEL4Chor Designer. Technical Report 2013/03, University of Stuttgart
26. Zimmermann O (2016) Microservices tenets. Computer Science - Research and Development pp 1–10

All links were last followed on July 21, 2017.