**Institute of Architecture of Application Systems**

# TraDE - A Transparent Data Exchange Middleware for Service Choreographies

Michael Hahn, Uwe Breitenbücher, Frank Leymann, Andreas Weiß

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

BibTeX:

```
@InProceedings{Hahn2017,
  author    = {Hahn, Michael and Breitenb{\"u}cher, Uwe and Leymann, Frank and
               Wei{\ss}, Andreas},
  title     = {{TraDE - A Transparent Data Exchange Middleware for Service
               Choreographies}},
  booktitle = {On the Move to Meaningful Internet Systems. OTM 2017
               Conferences: Confederated International Conferences: CoopIS,
               C{\&}TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017,
               Proceedings, Part I},
  editor    = {Panetto, Herv{\'e} and Debruyne, Christophe and Gaaloul, Walid
               and Papazoglou, Mike and Paschke, Adrian and Ardagna, Claudio
               Agostino and Meersman, Robert},
  publisher = {Springer International Publishing AG},
  address   = {Cham},
  series    = {Lecture Notes in Computer Science},
  volume    = {10573},
  pages     = {252--270},
  month     = oct,
  year      = {2017},
  isbn      = {978-3-319-69462-7},
  doi       = {10.1007/978-3-319-69462-7_16}
}
```



University of Stuttgart
Germany

# TraDE - A Transparent Data Exchange Middleware for Service Choreographies

Michael Hahn, Uwe Breitenbücher,
Frank Leymann, and Andreas Weiß

Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Germany
{hahnml,breitenbuecher,leymann,weissas}@iaas.uni-stuttgart.de

**Abstract.** Due to recent advances in data science the importance of data is increasing also in the domain of business process management. To reflect the paradigm shift towards data-awareness in service compositions, in previous work, we introduced the notion of data-aware choreographies through cross-partner data objects and cross-partner data flows as means to increase run time flexibility while reducing the complexity of modeling data flows in service choreographies. In this paper, we focus on the required run time environment to execute such data-aware choreographies through a new Transparent Data Exchange (TraDE) Middleware. The contributions of this paper are a choreography language-independent metamodel and an architecture for such a middleware. Furthermore, we evaluated our concepts and TraDE Middleware prototype by conducting a performance evaluation that compares our approach for cross-partner data flows with the classical exchange of data within service choreographies through messages. The evaluation results already show some valuable performance improvements when applying our TraDE concepts.

**Keywords:** Service Choreographies, Data-awareness, Cross-Partner Data Flow, Transparent Data Exchange, BPM

## 1 Introduction

Service-oriented architectures (SOA) have seen wide spread adoption. The concept of composing self-contained units of functionality as services over the network has found application in many research areas and application domains [18]. For example, in Business Process Management (BPM), Cloud Computing, the Internet of Things, or eScience. The composition of services can be specified through a broad variety of modeling languages which can be grouped into two categories: service orchestrations and service choreographies.

While service orchestrations, also known as *processes*, are specified from the viewpoint of one party that acts as a central coordinator, service choreographies provide a global view on the potentially complex conversations between multiple interacting services without relying on a central coordinator [3]. Therefore, the notion of service choreographies focuses on services taking part in a collaboration, as

so-called *participants*, and their interplay with other services by specifying corresponding conversations through message exchanges between them [3]. Prominent modeling languages for service orchestrations are the Business Process Management Notation (BPMN) [13] and the Business Process Execution Language (BPEL) [12]. Service choreographies can be modeled, for example, with modeling languages such as BPMN or BPEL4Chor [9].

With recent advances in data science the importance of data is increasing also in the domain of business process management [11,15]. For improving the level of data-awareness of service choreographies, we introduced an extended management life cycle [8] for data-aware choreographies and proposed an approach for enabling transparent data exchange (TraDE) in choreographies motivated on shortcomings of current choreography modeling languages [7]. The overall goal is to support data capabilities already on the level of the choreography to reduce modeling complexity while increasing run time flexibility.

In this work, we focus on the required middleware and its integration with process engines to support the run time perspective of data-aware choreographies as briefly outlined in Hahn et al. [7]. More specifically, the contributions of this paper can be summarized as follows: (i) we introduce and discuss an internal metamodel for a TraDE Middleware, (ii) present an architecture of such a middleware together with a prototypical implementation, and (iii) evaluated the prototype and the underlying TraDE concepts. The rest of this paper is structured as follows: Section 2 provides an overview of our previous work on the concepts of transparent data exchange in choreographies and the role of a corresponding middleware. Based on that, we introduce a modeling language-independent metamodel and a generic TraDE Middleware architecture and discuss how the middleware can be integrated with corresponding process engines in Sect. 3. In Sect. 4, we present a prototypical implementation of the architecture and its integration with a process engine solution. Section 5 presents an evaluation of the performance alteration when applying our TraDE concepts to a choreography execution by integrating our TraDE Middleware prototype. Finally, the paper discusses related work in Section 6, and concludes with our findings together with an outlook on future work in Section 7.

## 2 Transparent Data Exchange Approach

As background, in the following, we shortly outline our previously presented concepts for modeling and execution of data-aware choreographies through a transparent data exchange as discussed in detail in Hahn et al. [7]. Therefore, we briefly explain our modeling extensions, namely *cross-partner data objects* and *cross-partner data flows*, and how these extensions can be supported during run time by our new TraDE Middleware which is in the focus of this paper.

### 2.1 Cross-Partner Data Objects and Cross-Partner Data Flows

Figure 1 shows a motivation example of a data-aware choreography model with three interacting participants with our applied modeling extensions. We use the

Business Process Management Notation (BPMN) [13] as a basis to illustrate our modeling extensions. However, the underlying concepts and the middleware presented in this work are not bound to BPMN and can therefore be applied or integrated with any other choreography modeling language.

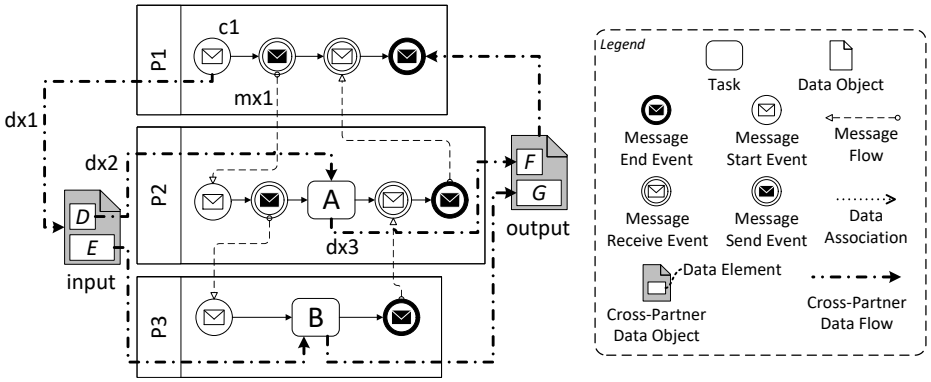



**Fig. 1.** Example choreography illustrated as BPMN collaboration model with applied *cross-partner data objects* and *cross-partner data flows*.

The conversations between the participants shown in Fig. 1 are modeled by BPMN message intermediate events and message flows, e. g., *mx1* in Fig. 1. Each of the participants is instantiated through a corresponding BPMN message start event, e. g., *c1* in P1, which consumes an incoming request message and extracts the contained data for processing it within the choreography. Choreography data is modeled by our cross-partner data objects, e. g., *input* in Fig. 1, and the reading and writing of the cross-partner data objects from tasks and events is specified through cross-partner data flows, e. g., *dx1* or *dx3* in Fig. 1. To avoid confusion between BPMN data objects as language-specific constructs and cross-partner data objects as a general concept, we use the generic term *data container* when we talk about modeling constructs that allow the specification of data on the level of a specific modeling language, e. g., BPMN data objects or BPEL variables. While data between participants is normally transfered through the exchange of messages within conversations, the notion of cross-partner data objects and cross-partner data flows allows us to decouple the exchange of data from the exchange of messages. For example, instead of forwarding the data of the initial request from participant P1 to participant P2 through the message flow mx1, we can directly specify a cross-partner data flow to task *A* of participant P2 where the data are actually processed. The same applies for the result data of task A, instead of forwarding it to other participants through the exchange of messages, it can be directly stored in cross-partner data object *output* by the specified cross-partner data flow *dx3* as shown in Fig. 1.

The notion of cross-partner data objects allows us to specify all data relevant for a choreography by specifying a set of cross-partner data objects. Such a set expresses the commonly agreed data of a choreography shared by and accessible from all participants and can therefore be seen as a *choreography data model*. This enables modelers to specify required data and their structures in a self-contained and consolidated manner within a choreography model. A *cross-partner data object* has a unique identifier and contains one or more *data elements*. For example, the cross-partner data object *input* contains the two data elements $D$ and $E$ as shown in Fig. 1. A data element has a name and contains a reference to a definition of its structure, e. g., using a build-in type system or an XML Schema Definition [16]. The actual data values during run time are held by these data elements. Therefore, they are comparable to the classical data containers of standardized choreography and orchestration modeling languages.

By introducing cross-partner data flows we support modelers so that they are able to intuitively specify data flows within and across participants in a choreography. While in classical choreography modeling languages, such as BPMN, data can only be passed across participants through message flows, cross-partner data flows allow to decouple the exchange of data from the exchange of messages. This means that instead of introducing additional modeling constructs for passing the value of a data container from one participant to another through a message flow (e. g., mx1 in Fig. 1), cross-partner data flows allow to model the exchange of data between participants and globally shared cross-partner data objects, e. g., dx2 or dx3 in Fig. 1. Since a lot of choreography modeling languages do not allow to specify directly executable models, an established approach is to transform the choreography models into a collection of private process models [4]. The resulting private process models can then be manually refined by adding corresponding internal logic for each participant. We extended this transformation step for data-aware choreographies in our previous work [7] by translating all cross-partner data objects into standard data containers on the level of the private process models again, e. g., using data objects in BPMN or variables in BPEL. The reason for this translation is that it allows modelers to refine the private processes using both locally and globally shared data containers in the same manner.

### 2.2 Towards a TraDE Middleware

After we have introduced our modeling extensions for transparent data exchange, the open research question is: "How can we support these modeling extensions during run time in order to actually execute a modeled data-aware choreography?".

Towards this goal in Hahn et al. [7] we outlined an overall system architecture for a modeling and run time environment that enables the execution of data-aware choreographies by supporting cross-partner data objects and cross-partner data flows. Therefore, we discussed our concept of introducing links between the data containers in the private process models of a choreography and the cross-partner data objects and data elements they represent. To provide and manage the cross-partner data objects of a data-aware choreography model we sketched a new middleware layer, the TraDE Middleware. This middleware acts

as a data hub between the choreography participants and therefore supports the process engines that execute the private process models with the realization of the modeled cross-partner data flows. In the following, we want to shortly recap our vision presented in Hahn et al. [7] by describing how such a TraDE Middleware is used to execute the example choreography shown in Fig. 1. This will provide us the basis for the introduction and detailed discussion of an underlying architecture and a prototypical implementation of a corresponding TraDE Middleware solution within the context of this paper.

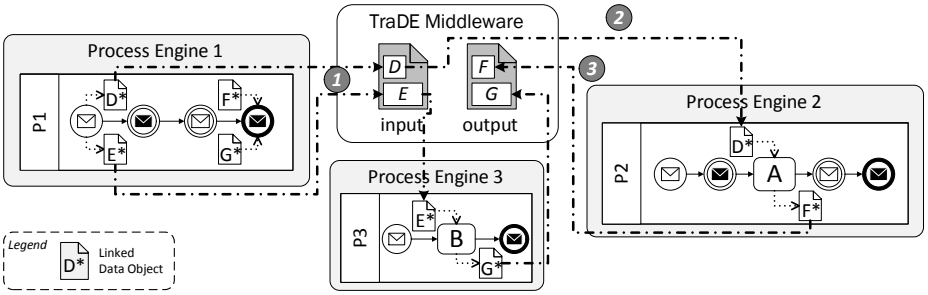



**Fig. 2.** Execution of cross-partner data flows of example choreography shown in Fig. 1.

Figure 2 shows the private process models of all three participants from our motivation scenario depicted in Fig. 1. For the sake of simplicity we omitted the message flows between the participants. Furthermore, we only describe the execution of the first three cross-partner data flows, namely dx1, dx2 and dx3, depicted in Fig. 1 as an example. Each task and event that was connected through a data flow with a cross-partner data object in Fig. 1 has a corresponding BPMN data object associated. As indicated by the *, these BPMN data objects are linked with the respective data elements of cross-partner data objects provided by the TraDE Middleware as depicted in Fig. 2.

In the following, we use the term *choreography instance* to refer to the collection of interconnected instances of the private process models implementing the choreography. Thus, in this example, a new choreography instance is created whenever participant P1 receives a new request message which is modeled by the BPMN message start event. The request message contains values for both data containers D and E of private process model P1. Process Engine 1 extracts these values from the request message to store them in the associated data containers. Since the data containers D and E are linked with the respective data elements *D* and *E* of cross-partner data object *input*, the process engine detects this linking and instead of storing the data internally, it directly uploads the data to the corresponding data elements in the TraDE Middleware (step 1 in Fig. 2). Subsequently, participant P1 invokes participant P2 through a message exchange (cf. *mx1* in Fig. 1). As soon as participant P2 reaches task A, Process Engine 2 reads the value of its data container D. Again the process engine detects that

this data container is linked to a cross-partner data object in the middleware and, therefore, retrieves the value directly from data element E of cross-partner data object input at the middleware (step 2 in Fig. 2). After task A is completed, the process engine stores the tasks' result data in data container F. Based on the linking, the data is directly uploaded to data element *F* of cross-partner data object *output* from where it can be retrieved by participant P1.

## 3 The TraDE Middleware

While in Hahn et al. [7] we outlined the overall vision and concepts of data-aware choreographies focusing on their modeling, in this paper, our main focus is on the TraDE Middleware. In the following we introduce a choreography language-independent metamodel and a detailed architecture for such a middleware as well as describe how it can be integrated with process engines.

### 3.1 Metamodel

The middleware and its underlying concepts should not be bound to any specific choreography and process modeling languages or related run time environments. Therefore, the TraDE Middleware has its own internal metamodel shown in Fig. 3.

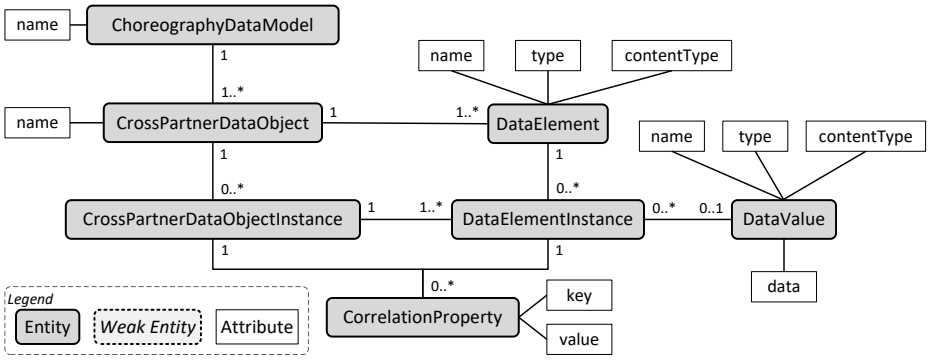



**Fig. 3.** Metamodel of the TraDE Middleware

The collection of cross-partner data objects of a choreography is represented by a ChoreographyDataModel entity within the TraDE Middleware. Therefore, a ChoreographyDataModel has a qualified *name* and contains one or more CrossPartnerDataObject entities which represent the cross-partner data objects of a choreography. A CrossPartnerDataObject has a unique *name*, a reference to its ChoreographyDataModel and contains one or more DataElement entities. A DataElement has a *name*, a *type* and a *contentType* definition. While the type allows to specify a concrete data structure and its syntax, e. g., in XML or JSON,

the contentType enables to specify the kind of data and its semantics. Therefore, the middleware can handle binary data without its interpretation while still being aware of its content and how to represent it. This allows us to support various types of data, e. g., structured and unstructured data, videos or pictures, as well as data formats and representations, e. g., XML, plain text, MPEG, or PNG. Such content type information can be specified, for example, using *Media Types*[1].

Since we have to represent and manage the data of multiple instances of a choreography model referring to the same CrossPartnerDataObject and DataElement entities during run time, we apply the well-known concept of *model instances* from BPM to our metamodel. Therefore, we introduce corresponding CrossPartnerDataObjectInstance and DataElementInstance entities which allow us to represent concrete instances of cross-partner data objects and their data elements for one specific instance of a choreography model. In order to correlate the data managed by the TraDE Middleware, i. e., CrossPartnerDataObjectInstance and DataElementInstance entities, with a choreography model instance, corresponding correlation information have to be supported and provided by the metamodel. Therefore, we associate a set of CorrelationProperty entities to the DataObjectInstance and DataElementInstance entities. These CorrelationProperty entities allow to uniquely identify a choreography instance on the level of a process engine as well as to identify the data that belongs to this instance on the level of the TraDE Middleware. Since the concept of property-based correlation is well known in the domain of BPM, we therefore reuse existing correlation mechanisms as provided, for example, by BPMN or BPEL in order to enable the instance correlation between process engines and the TraDE Middleware. To enable the reuse of data across choreography instances, concrete data should not be bound directly to one DataElementInstance entity. Therefore, the actual data is provided by an independent DataValue entity as shown in Fig. 3. This allows us to reuse and share DataValue entities across multiple DataElementInstance entities by referencing them. Moreover, it enables the manual creation of DataValue entities and therefore the upload of data to the middleware independent of a choreography instance. A DataValue has a *name*, a *type* defining the structure of its data and a *contentType* definition similarly as for DataElement entities. Furthermore, it holds the concrete *data*.

### 3.2 Architecture

Figure 4 presents the TraDE Middleware architecture. The design follows the three layer architecture pattern [5], which we describe in a top-down manner.

The *Presentation* layer provides a *Web User Interface* (UI) and a (set of) *REST API(s)* which enable the interaction with the TraDE Middleware, and its integration with other systems. By following the REST architectural style, each entity of our internal TraDE metamodel shown in Fig. 3 is represented as a resource and can, therefore, be easily accessed, referenced, and shared through

[1] Internet Assigned Numbers Authority (IANA), Media Types: https://www.iana.org/assignments/media-types/media-types.xhtml

a Uniform Resource Locator (URL). For example, process engines can use the REST API to integrate with the middleware in order to upload or retrieve data. Modeling tools can use the REST API to support modelers with deploying their specified collections of cross-partner data objects, i. e., choreography data models, to the middleware, so that they are available to the process engines during choreography run time.

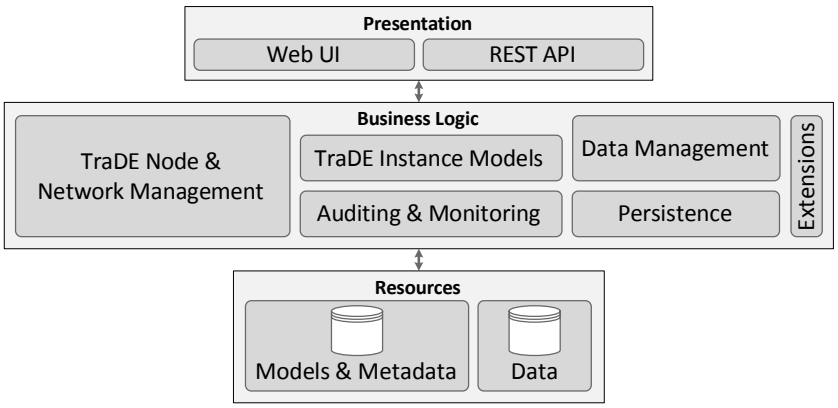



**Fig. 4.** Architecture of the TraDE Middleware

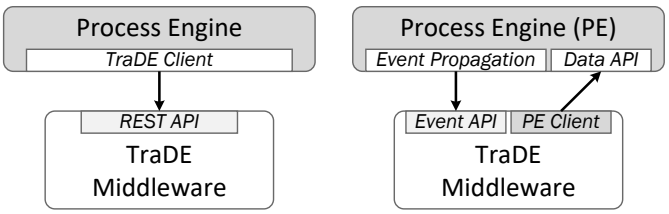
The *Business Logic* layer contains the core functionality of the middleware which is grouped into the following components. The *TraDE Instance Models* component contains instances of the metamodel shown in Fig. 3 to represent concrete cross-partner data objects and their instances within the middleware. All functionality related to data management is provided by the *Data Management* component which is the core component of the middleware. It supports the access and inspection of data associated to corresponding cross-partner data objects through the REST API. Furthermore, it provides the functionality to upload and retrieve data for a corresponding DataValue or DataElementInstance entity. Related to that, it also handles the correlation of the TraDE Instance Models with choreography instances in order to enable the process engines to access and retrieve the correct TraDE Instance Models and their associated data. Moreover, it is responsible for the management of the life cycle of the TraDE Instance Models. Therefore, it provides and implements corresponding life cycle operations such as create, instantiate, archive and delete. The *TraDE Node & Network Management* component is responsible for enabling a decentralized deployment of multiple middleware nodes and their connection into networks to allow more efficient data placement and staging as well as further optimizing the data exchange between the choreography participants. In order to decouple the life time of the data from its choreography instance the *Persistence* component stores both the internal metamodels of the middleware as well as the managed data in an underlying data source in order to guarantee its availability for later (re)use.

The *Auditing & Monitoring* component provides an associated life cycle for each of the entities in the above introduced metamodel. This enables the auditing and monitoring of all entities by emitting corresponding events whenever the life cycle of an entity changes. Furthermore, these internal events can be consumed by any interested component within the middleware, for example, allowing the Data Management component to trigger corresponding actions on state changes in order to realize the life cycle management of the TraDE Instance Models. The whole middleware is designed to be extensible in order to integrate new or adapt existing components. Therefore, the *Extensions* component provides corresponding functionality and mechanisms to plug-in new functionality as well as extensions or variants of existing components. For example, the default persistence component can be replaced by a new implementation that uses a different technology stack by adding it as an extension to the middleware.

The *Resources* layer contains all required resources used within the Business Logic layer. This comprises data sources for the persistence of TraDE Instance Models and related metadata about nodes and networks (*Models & Metadata* data source in Fig. 4) as wells as a data source for the actual data managed by the TraDE Middleware (*Data* data source in Fig. 4).

### 3.3 Integration with Process Engines

In the following we want to briefly discuss two approaches how the TraDE Middleware can be integrated with a process engine as shown in Fig. 5.




(a) TraDE-aware integration. (b) Two-way integration.

**Fig. 5.** Approaches for the integration of the TraDE Middleware with a process engine.

The *TraDE-aware integration* approach depicted in Fig. 5a explicitly introduces the TraDE Middleware at the process engine by extending its implementation with a *TraDE Client*. Therefore, the process engine is aware of the existence and functionality of the TraDE Middleware and actively uploads or retrieves data from the middleware in order to execute the specified cross-partner data flows. The advantage of this approach is that the process engine remains in control of the overall choreography execution or the corresponding private processes it is responsible for, respectively. The main disadvantage is that the process engine implementation has to be extended in order to integrate the client and introduce

required functionality to identify and handle the linking of data containers to cross-partner data objects. Especially, if the collaborating partners use different process engine solutions this integration approach requires too much effort.

In contrary, the *two-way integration* approach depicted in Fig. 5b integrates the process engine and the TraDE Middleware in a loosely coupled manner through corresponding APIs. The basic idea of this integration approach is to extract any data-related knowledge from the data-aware choreography models, e. g., which participant requires or produces which cross-partner data objects, to move the control of executing specified cross-partner data flows to the TraDE Middleware. Therefore, the process engines have to expose the execution state of their private process model instances through a corresponding event propagation mechanism, e. g., using messaging. The emitted state change events can then consumed by an *Event API* at the TraDE Middleware, so that it is always aware of the current execution state of the overall choreography instances for which it executes the cross-partner data flows. Furthermore, the process engine implementations have to expose an *Data API* which allows external parties, such as the TraDE Middleware, to retrieve and write data from and to data containers of the private process model instances executed by a process engine. The advantage of this integration approach is that the process engine implementation is not directly coupled with the TraDE Middleware. Instead it has to be only extended with generic event propagation functionality and expose its data management capabilities through an API. Some process engine implementations potentially already provide such capabilities and if not, the required extensions are not only bound and therefore usable for the integration with the TraDE Middleware. The main disadvantage of this approach is that the TraDE Middleware has to keep track of the execution state of all choreography instances to fully take over control of the execution of the cross-partner data flows which increases the complexity of the TraDE Middleware implementation and requires (to a certain extend) control over and access to the process engines.

## 4 Validation

To validate the practical feasibility of our concepts, we prototypically implemented the TraDE Middleware architecture shown in Fig. 4 and integrated it with a process engine following the TraDE-aware integration approach shown in Fig. 5a. For the implementation of our TraDE concepts, we extended the choreography modeling language BPEL4Chor and the process modeling language BPEL. The extension of BPEL4Chor allows us to specify cross-partner data objects and cross-partner data flows and the extension of BPEL enables us to link BPEL variables with cross-partner data objects in the resulting private process models constituting the overall data-aware choreography. An extended version of the open source BPEL engine Apache *Orchestration Director Engine* (ODE)[2] is used as process engine solution. In order to support the reading and writing

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

of cross-partner data objects, we extended the underlying implementation of Apache ODE and integrated it with our TraDE Middleware following the TraDE-aware integration approach discussed in Sect. 3.3. Therefore, the communication between the process engine and the TraDE Middleware is realized by integrating a REST API client into Apache ODE. On top of this client, we introduced a new *TraDE Manager* component that encapsulates logic for the retrieval and upload of data to the TraDE Middleware. For example, this comprises the creation and resolution of DataElementInstance and DataValue entities for a private process instance in order to upload data to the TraDE Middleware.

The TraDE Middleware itself is realized as a Java-based web server which exposes its functionality through a REST API. As underlying web server we are using Eclipse *Jetty*[3] in embedded mode. The REST API is specified and documented in form of a *Swagger Specification*[4] and implemented based on the *Jersey* RESTful Web Services framework[5]. For the implementation of the REST API, we are following an API-first approach which means that we developed against the API specification. Therefore, we use the related Swagger tooling support to generate client code as well as server code skeletons directly from the API specification. This approach has two major advantages. First, only the relevant business logic of the REST API has to be implemented and provided and second, changes in the API specification are directly reflected on the level of the code keeping the API and its implementation in sync. For the persistence of the TraDE instance models and the associated (business) data, we support MongoDB as a document-oriented database and the local file system as persistence layer for the middleware at the moment. Which persistence layer to use and a lot of other configuration options for the middleware can be specified in configuration files. At the moment, we only support a single-node deployment of the TraDE Middleware, but for future work, we are aiming at supporting also multi-node deployments by leveraging the capabilities of corresponding distributed data grid frameworks as provided, for example, by the *Hazelcast In-Memory Data Grid*[6]. The complete open source code of the middleware is available on GitHub[7].

## 5 Evaluation

In the following, we introduce a performance evaluation comparing cross-partner data flows with the classical exchange of data through messages within service choreographies. Therefore, we first present the underlying evaluation methodology we apply, followed by a description of the experimental setup and finally a discussion of the evaluation results.

[3] The Eclipse Foundation, Eclipse Jetty: https://www.eclipse.org/jetty/

[4] TraDE Swagger Specification: https://github.com/traDE4chor/trade-core/blob/master/server/swagger.json

[5] Oracle Corporation, Jersey: https://jersey.java.net

[6] Hazelcast, Inc., Hazelcast IMDG: https://hazelcast.org/

[7] TraDE Middleware: https://github.com/traDE4chor/trade-core

### 5.1 Evaluation Methodology and Experimental Setup

The focus of the evaluation is at empirically analyzing the performance variation when introducing our TraDE concepts. We therefore use the example choreography depicted in Fig. 1 to measure variations of the response time perceived by a user invoking the choreography. As a baseline, we use the same choreography model without our concepts applied, i. e., the data is passed within messages through the modeled message flows between the participants of the choreography. For example, the data contained in the initial request sent to participant P1 for data elements D and E is not uploaded to the TraDE Middleware, instead the data is stored in corresponding local data containers at the process engine and then passed within a message exchange to participant P2. The same applies for all other data exchanges depicted through cross-partner data flows in Fig. 1. For the sake of completeness, the standards-based choreography model used as baseline is shown in Fig. 6.

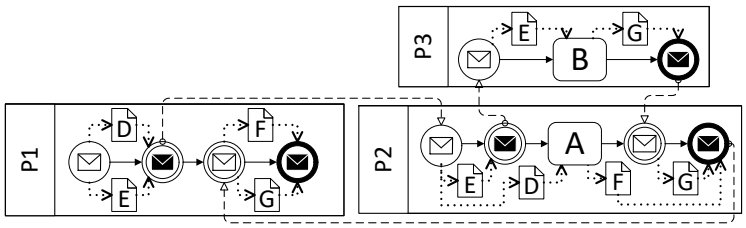



**Fig. 6.** Standards-based version of the example choreography shown in Fig. 1.

The two choreography models are implemented using BPEL4Chor [4] as choreography modeling language and are transformed [14] to three BPEL process models, one for each participant, which are manually refined in order to make them executable. The tasks A and B of participant P2 and P3 are implemented through BPEL assign activities so that they duplicate the random data contained in data containers D and E by its concatenation and store the result in data containers F and G. To guarantee that for both, the baseline scenarios and the TraDE scenarios, data has to be actually transferred through messages or the TraDE Middleware, respectively, we deploy each of the resulting executable private process models to a separate process engine instance using an extended version of Apache ODE with its default configuration. The BPEL process models for the TraDE scenarios are furthermore extended with corresponding TraDE annotations so that the process engine is aware of the linking of the BPEL variables with the cross-partner data objects managed by the TraDE Middleware.

In order to also measure a potential impact of the size of the data being processed, we introduce three scenarios with increasing data size for each of the input data elements (data object *input*, data elements $D$ and $E$): 1KB, 128KB and 256KB. While for the baseline scenarios all data is stored locally in corresponding data containers at the process engines, in the TraDE evaluation scenarios the

data are uploaded once to the corresponding cross-partner data objects in the middleware and retrieved directly from there only when required. For each of the six scenarios summarized in Table 1, a workload consisting of randomly generated request messages with the above mentioned data element sizes is created. The workload is distributed among a *warm-up phase* ($w(t_0)$) with 10 messages followed by an *experimental phase* comprising a set of 310 requests sent in five load bursts according to the following function over time:

$$m(t_i) = w(t_o) + \sum_{i=1}^{5} 2^{i-1} \cdot k \mid k = 10, w(t_0) = 10$$

The experimental environment is set up in an on-premise private cloud infrastructure on two virtual machines (VM). The evaluation VM is configured with 8 virtual CPUs, Intel® Xeon® CPU E5-2690 v2 3.00GHz, 32GB RAM, 120GB disk space, and is running an Ubuntu 14.04.4 64bit server distribution. We use Docker[8] within this VM to deploy the required three separate instances of Apache ODE and in addition one TraDE Middleware instance in the TraDE scenarios. The idea behind this level of nesting and using Docker for the deployment of the evaluation environment is that we want to have a clean and therefore identical setup for each of the conducted experiments towards creating reproducible evaluation results.

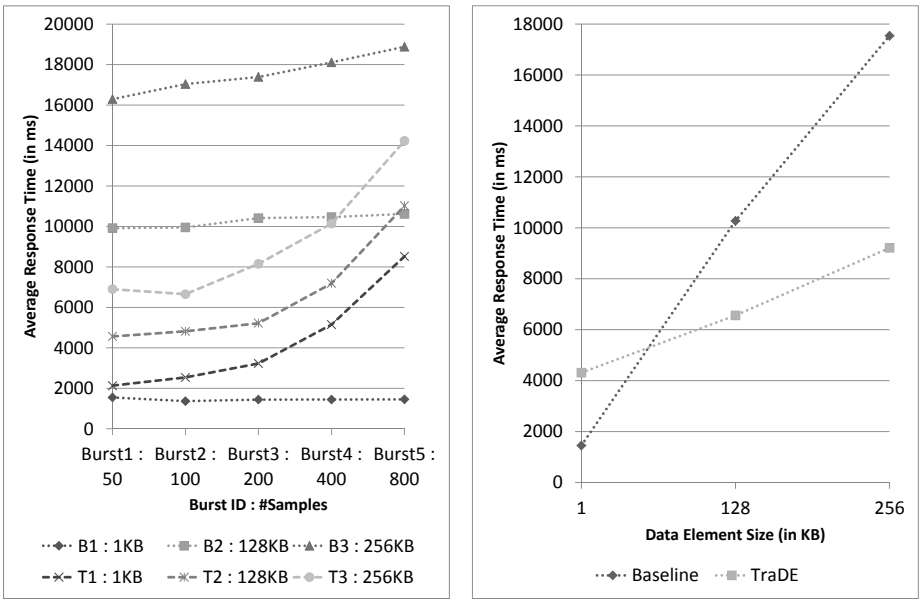
In order to setup the evaluation environment and to conduct the workload for each of the defined evaluation scenarios, we use Apache JMeter 3.2[9] as load driver which is deployed in a separate VM, with the following configuration: 2 virtual CPUs, Intel® Xeon® CPU E5-2690 v2 3.00GHz, 4GB RAM, 40GB disk space, running an Ubuntu 14.04.2 64bit desktop distribution. We created a JMeter test plan for each of the defined six scenarios, i. e., three baseline scenarios and three TraDE scenarios with data sizes of 1KB, 128KB and 256KB each, which concurrently sends the above defined workload for five concurrent users to the endpoint of the BPEL process model implementing participant P1. To alleviate the effect of outliers in the experimental results, we execute ten rounds of each scenario and calculate the average response time for each load burst while excluding the samples which are timed-out at the process engine.

### 5.2 Experimental Results

Figure 7a shows the experimental results comparing the user-perceived performance (average response time) of the load bursts of all scenarios. If we compare the baseline (B1-B3) with the TraDE (T1-T3) scenarios, there exists an overall beneficial impact to the user-perceived performance when introducing cross-partner data flows. However, this impact greatly varies on the size of the data being exchanged as well as on the workload applied throughout the load bursts. Comparing the scenarios with 256KB data size (B3 vs. T3) shown in Fig. 7a the

[8] Docker Community Edition: `https://www.docker.com/community-edition`
[9] Apache JMeter: `http://jmeter.apache.org/`

performance is improved by approximately 90% in total. When we have a look at the different load bursts in detail, this improvement decreases from approximately 136% in burst 1 to approximately 32% in the last load burst. Therefore, we can assume that when increasing the load on the middleware, the improvement will further degrade and actually convert to an overall performance deterioration.




(a) Average response time in milliseconds (ms) for the load bursts of all scenarios.

(b) Average response time (in ms) of scenarios based on data element size.

**Fig. 7.** Evaluation results for the defined six scenarios.

This is also underpinned when comparing the 128KB scenarios (B2 vs. T2) shown in Fig. 7a where the performance is still improved by approximately 56% in total. However, again the performance alters from an improvement of approximately 117% in burst 1 to a small performance degradation of approximately 0.04% in the last load burst. Comparing the scenarios with 1KB data size (B3 vs. T3) shown in Fig. 7a, the performance is degraded by approximately 66% in total when introducing the middleware and cross-partner data flows. There the overhead of introducing additional communication between the process engines and the TraDE Middleware to conduct the cross-partner data flows is higher than the improvements gained by reducing the amount of data to be exchanged.

Furthermore, Fig. 7a shows that for the baseline scenarios with message-based data exchange, the performance maintains quite stable in terms of increasing the workload across the load bursts but decreases significantly when increasing data element sizes. This fact is also underpinned when comparing the overall average

response time among all load bursts of the six scenarios based on the processed data element sizes as shown in Fig. 7b. In contrast, for the TraDE-based scenarios with cross-partner data flows, the performance maintains quite stable in terms of data element sizes as shown in Fig. 7b, but decreases significantly when increasing the workload throughout the five load bursts as shown in Fig. 7a.

Both types of scenarios are not fully able to process the complete workload in all load bursts without having a set of samples that timeout at the process engine (by default after 120s for Apache ODE). For the baseline scenarios this is especially the case in scenario B3 with a data element size of 256KB, where about 30% of the samples in load burst 5 (after approximately 556 successful samples) result in timeouts. A reason for this behavior might be the large amount of data (1550 instances $*$ 3840KB $\approx$ 5.9GB) ODE is not capable of handling in its default configuration at a certain point in time. For the TraDE scenarios such samples causing timeouts are randomly distributed across nearly all scenarios and load bursts, but also with a peak in load burst 5. The reason for such an unpredictable behavior is most probably related to the resolution of required data from the TraDE Middleware through the process engine. To correlate process instances and data object instances and to finally retrieve data element values, the process engines poll the middlewares' REST API by sending repeated requests every second as long as the process instance is not timed-out. These requests are queued up at the TraDE Middleware while throttling its performance for a certain amount of time which again results in timeouts at the process engines. The average amount of timed-out samples as well as a summary of the scenarios and their average response times is shown in Table 1.

**Table 1.** Summary of the experimental evaluation scenarios and their results.

| Scenario | ID | Data Element Size (in KB) | Total Data Size (in KB/instance) | Timed-out Req. (in %) | Avg. Resp. Time (in ms) |
|---|---|---|---|---|---|
| Baseline | B1 | 1 | 15 | 0.04 | 1451.58 |
| | B2 | 128 | 1920 | 0.32 | 10272.86 |
| | B3 | 256 | 3840 | 6.49 | 17540.36 |
| TraDE | T1 | 1 | 6 | 0.20 | 4313.86 |
| | T2 | 128 | 768 | 0.47 | 6560.86 |
| | T3 | 256 | 1536 | 0.61 | 9214.08 |

In summary the evaluation results show that introducing a TraDE Middleware layer and applying our concept for cross-partner data flows in service choreographies provide valuable performance improvements already for relatively small data sizes above 128KB. To alleviate the performance degradation when increasing the load at the middleware, in future work, we will improve our prototypical implementation and its integration with Apache ODE so that its performance maintains stable when increasing the workload. Therefore, future experiments will aim at investigating current capacity limitations of the TraDE Middleware

when increasing the data sizes as well as the number of concurrent users and requests. The complete evaluation result data and any related material, e. g., BPEL process models and JMeter test plans, are available on GitHub[10].

## 6 Related Work

In this section, we will compare our TraDE approach and the introduced middleware with related work on cross-partner data flows in service choreographies. 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 related work following the same paradigm.

Meyer et al. [10] introduce a model-driven approach towards improving the modeling and enactment of data exchange in choreographies through messages. Through an extension of the BPMN modeling language with annotations on BPMN data objects they enable the specification and enactment of message extraction from and message storage to local databases. This allows them to completely automate the exchange of data across participants and also to enrich model transformations with data-related aspects. However, our approach introduces the TraDE Middleware as an abstraction layer and data hub, instead of directly binding data containers to databases on the level of the models. This allows us to decouple the exchange of data from the exchange of messages towards increasing run time flexibility while reducing modeling complexity of choreographies.

Barker et al. [1] introduce MAP as new language for executable service choreographies. By introducing the concept of so-called *peers* they provide a mechanism to apply extra functionality that enables web services to participate in a choreography without requiring to adapt the underlying service implementations. In contrast to their approach, we are building on top of standardized languages and tools in order to support cross-partner data flows in choreographies.

Furthermore, Barker et al. [2] introduce the *Circulate approach* which combines the advantages of the orchestration and choreography paradigm. While the control flow remains orchestration-based the data flow is realized in a choreography-based manner. Similar to the above mentioned peers, *proxies* are introduced to enable the transfer of data across services. Based on that, the process engine communicates with the proxies in order to invoke services and exchange data between them. In general, our approach is similar since we introduce the TraDE Middleware as an intermediary to realize the data exchange. However, instead of explicitly invoking proxies to conduct the data exchange, we propose to introduce cross-partner data flows which are transparently conducted through the TraDE Middleware and its integration with the process engines. With our approach, models are enriched instead of changed to preserve their portability.

Habich et al. [6] provide an approach for cross-partner data flows similar to ours but with focus on the level of process models and BPEL in particular. They try to solve the problem of BPEL's *by value* semantics for data exchange resulting

[10] TraDE Evaluation: `https://github.com/traDE4chor/trade-core-evaluation/tree/master/initial-evaluation`

from the centralized and implicitly specified data flows in BPEL through variables and assign activities. Therefore, they extend BPEL with the notion of *BPEL data transitions* and apply their concept of Data-Grey-Box Web Services. These web services provide enhanced interfaces specifying related data aspects and therefore allow to define which parameters are passed by value or by reference. With the help of the introduced data transitions, explicit data flows between the composed Data-Grey-Box Web Services can be specified in BPEL process models. The combination of both concepts further allows to integrate specialized data propagation tools and logic, e. g., Extract Transform Load (ETL) tools, to implement the modeled data transitions which act as mediators to provide and resolve data by reference between the composed Data-Grey-Box Web Services during run time. In contrast to introducing explicit data flows between interacting services on the level of BPEL, or process models in general, we argue that cross-partner data flows can be specified much easier and more intuitively on the level of choreography models since choreographies provide a global view on the interactions and conversations between the services. Our overall goal is to hide cross-partner data flows on the level of the process models by transparently providing the required logic and functionality through the TraDE Middleware.

## 7 Conclusions and Outlook

To support the notion of data-aware service choreographies, we previously introduced our concepts for transparent data exchange through cross-partner data objects and cross-partner data flows in choreographies. In this work, we focused on the execution of data-aware choreographies with the help of a new middleware layer: the TraDE Middleware. Therefore, we introduced an architecture and an internal metamodel for the TraDE Middleware and discussed its integration with a process engine. To evaluate the feasibility and applicability of our approach and the middleware, we conducted a performance evaluation comparing our approach for cross-partner data flows with the classical exchange of data through messages within service choreographies. The evaluation results already show interesting performance improvements for relatively small data sizes when applying our TraDE concepts and integrating the TraDE Middleware with process engines.

In future work, we will provide a Web UI for our TraDE Middleware prototype in order to enable its use by human users and also ease manual data access and inspection. Furthermore, we plan to tackle the performance weaknesses identified within the evaluation towards improving the overall performance and robustness of the middleware as well as to improve its integration with Apache ODE, e. g., introducing a callback mechanism instead of periodically polling resource changes. Moreover, we are planning to integrate our middleware with the *ChorSystem* middleware introduced in Weiß et al. [17]. The goal is to leverage the capabilities of the ChorSystem middleware to ease and improve the deployment and management of data-aware choreographies in the future.

## Acknowledgment

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

## References

1. Barker, A., Walton, C., Robertson, D.: Choreographing Web Services. IEEE Transactions on Services Computing 2(2), 152–166 (2009)
2. Barker, A., Weissman, J.B., Van Hemert, J., et al.: Reducing Data Transfer in Service-Oriented Architectures: The Circulate Approach. IEEE Transactions on Services Computing 5(3), 437–449 (2012)
3. Decker, G., Kopp, O., Barros, A.: An Introduction to Service Choreographies. Information Technology 50(2), 122–127 (2008)
4. Decker, G., Kopp, O., Leymann, F., Weske, M.: Interacting services: from specification to execution. Data & Knowledge Engineering 68(10), 946–972 (2009)
5. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley Professional (2002)
6. Habich, D., Richly, S., Preissler, S., Grasselt, M., Lehner, W., Maier, A.: BPEL$^{DT}$ - Data-Aware Extension for Data-Intensive Service Applications. In: Emerging Web Services Technology, vol. II. Springer (2008)
7. Hahn, M., Breitenbücher, U., Kopp, O., Leymann, F.: Modeling and Execution of Data-aware Choreographies. An Overview. Springer CSRD (2017)
8. Hahn, M., Karastoyanova, D., Leymann, F.: A Management Life Cycle for Data-Aware Service Choreographies. In: Proc. of ICWS'16. IEEE (2016)
9. Kopp, O., Leymann, F., Wagner, S.: Modeling Choreographies: BPMN 2.0 versus BPEL-based Approaches. In: Proc. of EMISA'11. LNI, GI (2011)
10. Meyer, A., Pufahl, L., Batoulis, K., Fahland, D., Weske, M.: Automating Data Exchange in Process Choreographies. Information Systems (2015)
11. Meyer, S., Sperner, K., Magerkurth, C., Pasquier, J.: Towards Modeling Real-world Aware Business Processes. In: Proc. of WoT'11. pp. 81–86. ACM (2011)
12. OASIS: Web Services Business Process Execution Language Version 2.0 (2007), http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
13. OMG: Business Process Model And Notation (BPMN) Version 2.0 (2011), http://www.omg.org/spec/BPMN/2.0/
14. Reimann, P., Kopp, O., Decker, G., Leymann, F.: Generating WS-BPEL 2.0 Processes from a Grounded BPEL4Chor Choreography. Technical Report 2008/07, University of Stuttgart (2008)
15. Schmidt, R., Möhring, M., Maier, S., Pietsch, J., Härting, R.C.: Big Data as Strategic Enabler - Insights from Central European Enterprises. In: Business Information Systems, LNBIP, vol. 176, pp. 50–60. Springer International Publishing (2014)
16. W3C: XML Schema Definition Language (XSD) 1.1 Part 1: Structures. Online (2012), http://www.w3.org/TR/xmlschema11-1/
17. Weiß, A., Andrikopoulos, V., Sáez, S.G., Hahn, M., Karastoyanova, D.: ChorSystem: A Message-Based System for the Life Cycle Management of Choreographies. In: Proc. of OTM'16 Conferences. LNCS, vol. 10033, pp. 503–521 (2016)
18. Zimmermann, O.: Microservices tenets. Springer CSRD pp. 1–10 (2016)

All links were last followed on Thursday 31st August, 2017.