**Institute of Architecture of Application Systems**

# Modeling Data Transformations in Data-Aware Service Choreographies

Michael Hahn, Uwe Breitenbücher, Frank Leymann,
Michael Wurster, Vladimir Yussupov

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

BibTeX:

```
@InProceedings{Hahn2018_TraDEDataTransformationModeling,
  author    = {Hahn, Michael and Breitenb{\"u}cher, Uwe and Leymann, Frank and
               Wurster, Michael and Yussupov, Vladimir},
  title     = {{Modeling Data Transformations in Data-Aware Service
                Choreographies}},
  booktitle = {Proceedings of the IEEE 22nd International Enterprise Distributed
               Object Computing Conference (EDOC)},
  publisher = {IEEE Computer Society},
  pages     = {28--34},
  year      = {2018},
  doi       = {10.1109/EDOC.2018.00014}
}
```



University of Stuttgart
Germany

# Modeling Data Transformations in Data-Aware Service Choreographies

Michael Hahn, Uwe Breitenbücher, Frank Leymann, Michael Wurster, and Vladimir Yussupov
Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Germany
{michael.hahn, uwe.breitenbuecher, frank.leymann, michael.wurster, vladimir.yussupov}@iaas.uni-stuttgart.de

***Abstract*—The importance of data is steadily increasing in the domain of business process management due to recent advances in data science, IoT, and Big Data. To reflect this paradigm shift towards data-awareness in service choreographies, we introduced the notion of data-aware choreographies based on concepts for Transparent Data Exchange (TraDE) in our previous works. The goal is to simplify the modeling of business-relevant data and its exchange in choreography models while increasing their run time flexibility. To further improve and simplify the modeling of data-related aspects in service choreographies, in this paper, we focus on the extension of our TraDE concepts to support the modeling of data transformations in service choreographies. Such data transformation capabilities are of dire need to mediate between different data formats, structures and representations of the collaborating participants within service choreographies. Therefore, the paper presents a modeling extension as means for specifying and executing heterogeneous data transformations in service choreographies based on our TraDE concepts.**

## I. Introduction

With recent advances in data science the importance of data is increasing also in the domain of Business Process Management (BPM) [1, 2]. The concept of Service-oriented Architectures (SOA), i. e., composing units of functionality as services over the network, has found application in many research areas and application domains besides BPM [3, 4]. For example, in Cloud Computing, the Internet of Things, or eScience. To specify such compositions of services, a broad variety of modeling languages exist 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 [5, 6]. Therefore, the notion of service choreographies focuses on services taking part in a collaboration as *participants* and their interplay with other services by specifying corresponding conversations through message exchanges between them [6]. However, current state of the art in service choreographies, despite some promising works [7, 8], fails to provide an overall solution that allows data to assume its deserved primary role. We tackled this issue by introducing the notion of *data-aware choreographies* through concepts for *Transparent Data Exchange (TraDE)* [9]. Our main goal is to simplify the specification of business-relevant data and its exchange across participants in choreography models.

However, participants in service choreographies rely on their own internal data models on which their business logic is based. Therefore, data transformation capabilities are of dire need to mediate between different data formats, structures and representations of the collaborating participants within service choreographies. For example, if one participant needs to aggregate the data provided by other participants, data transformations are required. Such data transformations have to be explicitly specified in choroeography models by introducing additional activities that conduct the required transformations. This pollutes the choreography models with data transformation functionality that is not relevant from a business perspective but technically required, leading to more complex models.

In this work, the focus is on how data transformations can be specified on the level of data-aware service choreographies to provide aforementioned data transformation capabilities. The contributions of this paper can be summarized as follows: (i) we present concepts for modeling and transparent execution of data transformations in service choreographies based on our previous work on TraDE, and (ii) present a prototypical implementation of an integrated ecosystem for data-aware service choreographies with data transformation support.

The rest of this paper is structured as follows. Section II motivates this work and provides further details on background and problem statements to be tackled. In Section III, we introduce and provide a modeling extension for data transformations and describe its execution semantics. The prototypical implementation of the resulting ecosystem is outlined in Section IV. Finally, the paper discusses related work (Section V), and concludes with our findings together with an outlook on future work in Section VI.

## II. Motivation, Background, and Problem Statements

Before going into the discussion on how to support data transformations in service choreographies, we first want to provide some background on our previous works on TraDE as well as further motivate this work. We therefore use the example choreography models depicted in Fig. 1 to describe the standard way of exchanging data across participants (*Message-based Data Exchange*), how our concepts and modeling extensions for transparent data exchange (TraDE) are applied (*TraDE Extensions*) and discuss open problems regarding the support of data transformations within service choreographies.

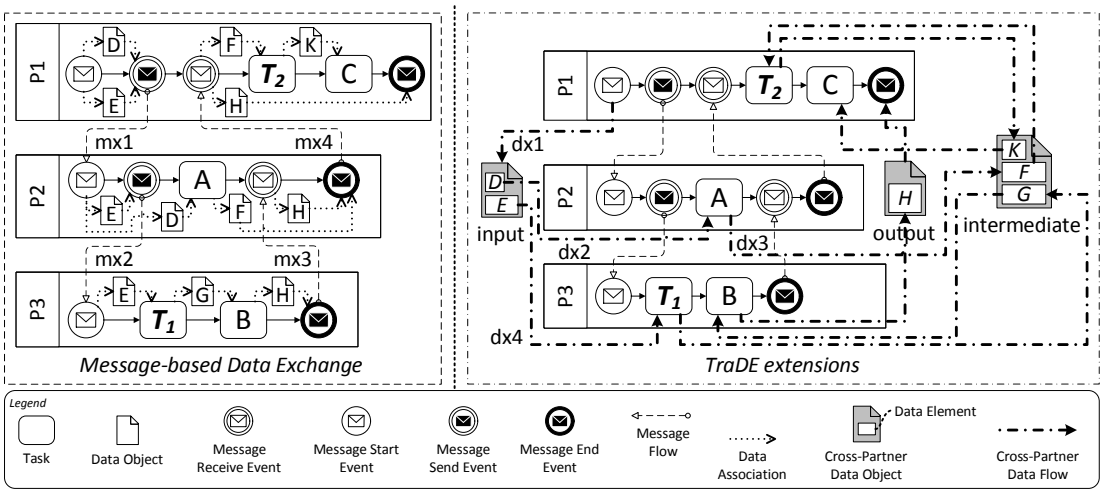


Figure 1. An example choreography model based on standard message-based data exchange (left) and in comparison using our TraDE concepts (right).

### A. Data in Service Choreographies

Both choreography models depicted in Fig. 1 have three interacting participants and are illustrated as Business Process Management Notation (BPMN) [10] collaboration models. The conversations between the participants are modeled by BPMN message intermediate events and message flows, e. g., *mx1* in Fig. 1. The participants are instantiated through corresponding BPMN message start events which consume incoming request messages to extract and process the contained data within the choreography participants. In the following, a short description of the behavior of the example choreography models shown in Fig. 1 is provided using the left one as a basis and then describe how our TraDE concepts are applied.

Following the standard way of modeling and exchanging data in choreographies, we call this *Message-based data exchange*, choreography data is modeled in form of BPMN data objects and exchanged as content of messages through specified message flows as shown in the left choreography model depicted in Fig. 1. The BPMN data objects have therefore to be modeled within the context of each participant processing the related data. Whenever participant ***P1*** receives an incoming request, it extracts the contained data from the request message and stores it in data objects ***D*** and ***E***. The data is then wrapped within a message and sent to participant ***P2*** through message flow ***mx1***. There, data object ***E*** is directly forwarded to participant ***P3*** by sending a message depicted through message flow ***mx2***. Before participant ***P3*** can process the data of data object ***E*** using task ***B***, the data has to be transformed into a processable representation and stored in data object ***G*** which can then be used as input by task ***B***. Therefore, the participant specifies a corresponding transformation task $T_1$ that executes required data transformation logic. The result of task ***B*** is then stored in data object ***H*** and sent back to participant ***P2*** through message flow ***mx3***. In the meantime, participant ***P2*** runs task ***A*** using data object ***D*** as input and stores its result in data object ***F***. As soon as the results of both participants, i. e., data object ***F*** and data object ***H***, are available, participant ***P2*** is sending the data wrapped in a message back to participant ***P1*** (message flow ***mx4***). Finally, participant ***P1*** also has to transform data object ***F*** received from participant ***P2*** to enable its further processing by task ***C***. Therefore, an additional transformation task $T_2$ provides related data transformation logic which transforms data object ***F*** into the representation required by task ***C***. The result of the transformation is stored in a new data object ***K*** that is then used as input of task ***C***. Finally, the choreography returns a message to the initial requester containing data object ***H***.

As discussed in our previous work [9], following the *Message-based data exchange* approach has some drawbacks. First, the same data objects have to be specified in a repetitive manner within the context of each participant that is reading or writing data to them, e. g., data objects ***E*** and ***H*** have to be specified in all three participants. Furthermore, data flow within participants and across participants has to be modeled differently. While the data flow within a participant can be modeled through corresponding BPMN data associations, data exchange across participants has to be modeled through message flows and related control flow modeling elements, e. g., BPMN *send* and *receive* tasks, and *message throw* or *message catch* events. This results in the fact, that data cannot be exchanged across participants without introducing additional control flow constructs at two participants (sender and receiver) within a choreography model. Another related issue is that data are potentially unnecessary routed through a set of participants, e. g., data object ***D*** is routed from participants ***P1*** to ***P3*** over participant ***P2*** through message exchanges ***mx1*** and ***mx2***.

To tackle these issues, we introduced concepts for data-aware choreographies through Transparent Data Exchange (TraDE) modeling extensions and the TraDE Middleware supporting their execution in our previous works [9, 11]. The choreography model depicted on the right of Fig. 1 applies our TraDE modeling extensions, namely *cross-partner data objects* and *cross-partner data flows*, to replace the message-based data exchange used in the left choreography model. The choreography data is now modeled in a participant-independent

manner using 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***. This allows us to exchange data across participants independent of the modeled conversations (message flows) and therefore decouples the exchange of data from the exchange of messages. For example, instead of forwarding and routing the data of the initial request from participant ***P1*** to participants ***P2*** and ***P3*** through message flows ***mx1*** and ***mx2***, we can directly specify cross-partner data flows to task ***A*** of participant ***P2*** (***dx2***) and task $T_1$ of participant ***P3*** (***dx4***) where the data are actually processed. Furthermore, the use of cross-partner data objects makes it explicit which data the choreography and its participants require and produce. For the example shown in Fig. 1, a modeler can directly identify the three cross-partner data objects ***input***, ***intermediate***, and ***output***. According to their names, they store the input and output data of the overall choreography as well as some intermediary data that is exchanged between the participants. Each *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 [12]. The actual data values during run time are represented by these data elements. The idea of this level of nesting is that data elements can be grouped based on their relations to create more understandable and better readable choreography models. Based on the fact that a lot of choreography modeling languages do not produce directly executable models, an established approach is to transform the choreography models into a collection of private process models [13]. The resulting private process models can then be manually refined by adding corresponding internal logic for each participant. To execute the resulting data-aware service choreographies (i. e., by executing the refined private process models) the TraDE Middleware and its integration into process engines is introduced and described in detail in our previous work [11]. The TraDE Middleware acts as a data hub between the choreography participants and therefore supports the process engines that execute the private process models with the modeled cross-partner data flows.

However, the choreography model with our TraDE concepts applied still forces modelers to specify data transformations by adding corresponding tasks and related data flows that conduct the required data transformation logic within the participants of a choreography model. For example, participant ***P3*** requires transformation task $T_1$ to transform the data produced by participant ***P1*** (data element ***E***) and participant ***P1*** requires another transformation task $T_2$ to be able to process the data produced by participant ***P2*** (data element ***F***). To further improve the modeling of data-related aspects in service choreographies, concepts for a seamless specification and execution of data transformations in service choreographies are required.

### B. Problem Statements

The main problem to solve is how to enable modelers to specify required transformations of choreography data in a simple and understandable manner and outside of the scope of choreography participants. By forcing modelers to explicitly define transformation tasks on the level of participants, as depicted on the left of Fig. 1, the participant models become polluted with functionality that solely operates on data having no direct impact on the control flow of a participant and therefore can be executed in a decoupled fashion. The transformed data is therefore only available at a single participant and has to be exchanged to make it explicitly available and enable its (re)use at other participants. Furthermore, the underlying transformation logic is scattered across the participants and not easily identifiable on the level of the choreography model since it is integrated to the choreography through a task definition which provides the respective transformation logic as its implementation. Therefore, modelers should be enabled to define data transformations directly on the level of choreography data instead of forcing them to introduce transformation logic on the level of participants.

By supporting the definition of data transformations independent of participants' control flow, another challenge is on how to provide and invoke the related data transformation logic during choreography execution. While explicitly modeled transformation tasks are executed by the process engine responsible for the execution of the private process model of a choreography participant, now the required transformation logic has to be otherwise integrated and triggered within the context of a choreography based on the choreography data. This requires concepts on how to provide and trigger modeled data transformations in a data-driven manner as a substitution for explicit transformation tasks in choreography models. In the next section, related concepts and a modeling extension are introduced as a solution to these problems.

## III. Data Transformations in Service Choreographies

As outlined in Section II-A and depicted by the left model shown in Fig. 2, applying our TraDE concepts still forces modelers to manually specify data transformations by adding corresponding transformation tasks to a choreography model. Furthermore, it even requires modelers to introduce additional cross-partner data flows connecting the transformation tasks and their inputs and outputs represented through cross-partner data objects leading to more complex models. While the TraDE concepts allow to decouple data from participants by specifying cross-partner data objects, something similar for data transformations is missing, i. e., decoupling data transformations from concrete participants. Therefore, our goal is to provide an end-to-end support for the modeling and execution of data transformations in service choreographies independent of participants directly between cross-partner data objects. The choreography model on the right of Fig. 2 presents our vision on modeling data transformations in service choreographies in a seamless and straightforward manner.

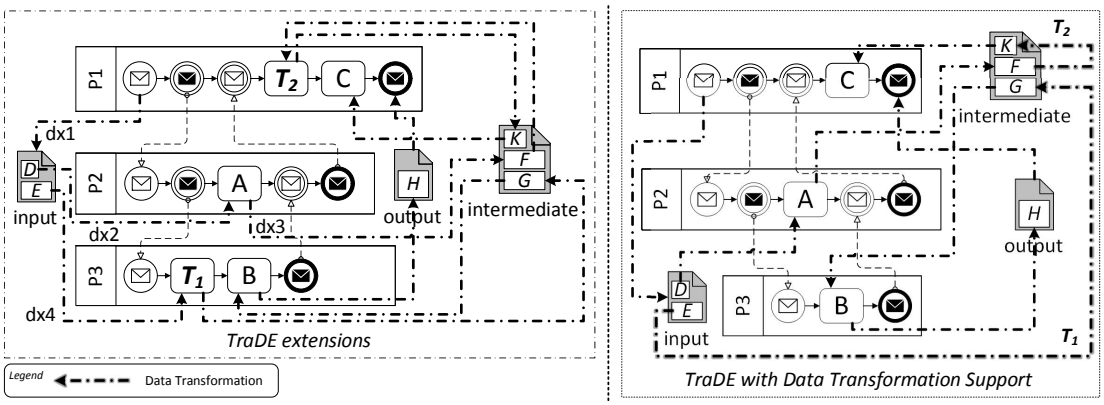


Figure 2. Comparison of the example choreography depicted in Fig. 1 using TraDE concepts (left) and with targeted support for data transformations (right).

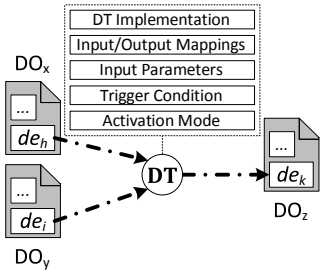


Figure 3. Modeling data transformations through a new *Data Transformation* (DT) element and cross-partner data flows.

There, the transformation tasks $T_1$ and $T_2$ are replaced by corresponding cross-partner data flows with associated data transformation logic between the data objects *E* and *G* (for task $T_1$) as well as *F* and *K* (for task $T_2$). The underlying software, e. g., services, scripts or executables, that is used in the transformation tasks and provides the data transformation logic is referred to as *DT Implementation* in the following. In the following, a modeling extension for data transformations and its execution semantics are presented.

### A. Modeling Data Transformations in Service Choreographies

Figure 3 depicts our data transformation (DT) modeling extension for the specification of data transformations in data-aware service choreography models. Modelers can use the new DT element to specify a required data transformation directly between a set of cross-partner data objects independent of participants. All sources and targets of such a DT element have to be cross-partner data objects. The rationale behind that restriction is twofold. On one hand, this guarantees that there always exists an independent container (i. e., cross-partner data object) for all input and output data of a data transformation which will be materialized during run time. This is required since the specified cross-partner data objects define the choreography data in terms of data formats, structures and related properties. Furthermore, cross-partner data objects represent the data that will be produced and consumed by the choreography's tasks and therefore reflect the potential inputs for data transformations on the level of a choreography. Therefore, a cross-partner data flow connecting a cross-partner data object and a DT element is well-defined, while for a cross-partner data flow between a task and a DT element it is unclear what data can be expected as the tasks' output. On the other hand, this allows modelers to graphically specify the inputs and outputs of a data transformation by simply connecting cross-partner data objects via cross-partner data flows with them. This restriction will be also valuable for providing more advanced functionalities in future, e. g., data provenance, sharing of transformation results, or monitoring of data exchange and transformations.

Figure 3 shows an example for a data transformation defined through a DT modeling element between the cross-partner data objects $\mathrm{DO}_x$, $\mathrm{DO}_y$ and $\mathrm{DO}_z$ connected through corresponding cross-partner data flows. The DT element contains a reference to the software that provides the related data transformation logic, e. g., a web service. Concepts on how the integration, invocation and execution of heterogeneous data transformation software can be realized are not in the scope of this work and will be presented in future work. If a data transformation requires or produces more than one input or output, modelers are able to map the connected cross-partner data objects to the respective inputs and outputs of the underlying DT Implementation through specifying a set of *Input/Output Mappings*. The definition of such mappings might differ based on the type of the underlying DT Implementation. Therefore, modelers might be also graphically supported through the choreography modeling tool by utilizing available knowledge about required inputs and outputs from a DT Implementation, e. g., extract related information from the interface definition of a web service. By default, required inputs of a DT Implementation will be provided through mapping the cross-partner data objects of incoming cross-partner data flows to them and the resulting outputs are mapped to respective cross-partner data objects of outgoing cross-partner data flows to store the transformation results. Furthermore, the DT element allows to specify a set of *Input Parameters* which enables modelers to specify inputs for a DT Implementation that are not provided through corresponding cross-partner data

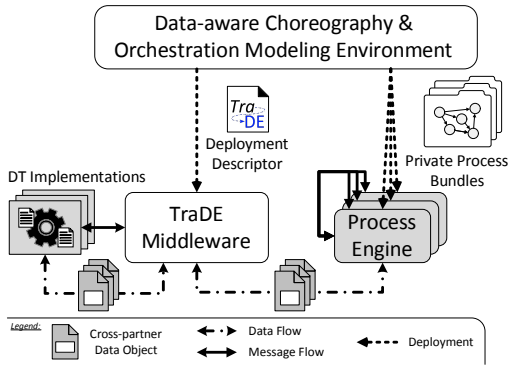


Figure 4. Integrated system architecture and deployment artifacts of the TraDE ecosystem.

objects. For example, input parameters can be used to provide constant values, e. g., for the configuration or initialization of the underlying DT Implementation. Furthermore, an optional *Trigger Condition* and *Activation Mode* can be specified for each data transformation. A trigger condition allows to specify a certain logical condition which has to be evaluated to *true* before the referenced DT Implementation is executed. The activation mode enables modelers to specify when the data transformation should be conducted: *on-read* or *on-write*. This will be discussed in more detail in the context of the execution semantics of the DT element in the following section.

### B. Transparent Execution of Data Transformations

Before describing how modeled data transformations will be executed based on TraDE, we first have a short look on the deployment artifacts of a data-aware choreography and how they are distributed and utilized by the different components of the overall TraDE ecosystem shown in Fig. 4.

The *Data-aware Choreography & Orchestration Modeling Environment* enables modelers to specify data-aware service choreographies by modeling *cross-partner data objects*, cross-partner data flows and *Data Transformation* (DT) elements. As outlined in Section II-A, the resulting choreography models will then be transformed into a collection of private process models to enable their execution. The refined private process models are finally packaged with related files, e. g., process engine deployment descriptors, as *Private Process Bundles* for the deployment on *Process Engines* as shown in Fig. 4. Furthermore, the specification of all modeled cross-partner data objects and their dependencies are exported to a *TraDE Deployment Descriptor* file. This deployment descriptor file is uploaded to the TraDE Middleware where it is compiled into the middleware's internal metamodel so that all specified cross-partner data objects and data elements are provided and exposed as resources through the middleware's REST API [11]. Moreover, the TraDE Deployment Descriptor contains a representation of all specified data transformations, i. e., DT elements, of a modeled choreography. This comprises all the information outlined in Section III-A, i. e., the reference to a DT Implementation and mappings of cross-partner data objects to inputs and outputs of such a DT Implementation. The TraDE Middleware extracts all this information from the deployment descriptor and translates it into an internal representation. Since the integration and execution of heterogeneous DT Implementations through the TraDE Middleware is an additional challenge, required concepts and tools will be presented in future work. In the following, we therefore focus on the execution semantics of the defined DT elements and how the TraDE Middleware conducts them by triggering respective transformations on data provided through cross-partner data objects.

Since the realization of data exchange across participants through integrating the TraDE Middleware with the underlying process engines is described in detail in our previous work [11], in the following, our focus is on the execution semantics of DT elements and how the TraDE Middleware conducts them. Therefore, we first want to have a closer look on the choreography language-independent metamodel underlying the TraDE Middleware as presented in our previous work [11]. For the sake of conciseness, we only provide a brief recap of it. As outlined in Section II-A, all cross-partner data objects and their data elements are represented through *CrossPartnerDataObject* and *DataElement* entities at the TraDE Middleware. In addition to the model perspective, the metamodel of the TraDE Middleware provides further entities to represent instances of cross-partner data objects and data elements to reflect the run time perspective of data-aware choreographies, i. e., manage the data of choreography instances. For each choreography instance *CrossPartnerDataObjectInstance* and *DataElementInstance* entities will be created within the TraDE Middleware with associated *CorrelationProperty* entities that enable to uniquely identify to which choreography instance the data object and data element instances belong. The actual data is provided through *DataValue* entities which are referenced by one or more *DataElementInstance* entities. All these entity types have an event model associated, i. e., a life cycle with states and transitions. We distinguish two types of event models, one for model entities (e. g., CrossPartnerDataObject, DataElement) and one for instance entities (e. g., DataElementInstance, DataValue). Whenever such an entity changes its state, a corresponding event is fired within the TraDE Middleware.

The middleware uses this event propagation mechanism to trigger specified data transformations in an event-driven manner based on corresponding data-related events, e. g., a DataValue is *initialized* or *modified*. To discuss the different possibilities of triggering data transformations within the middleware, we describe the overall process of invoking a DT Implementation and how to resolve its input data and persist its output data as cross-partner data objects.

First, all required information to execute a specified DT Implementation is collected to prepare its invocation. This comprises the resolution of corresponding DataValue entities for a specific choreography instance that holds the input data of the transformation as modeled in the choreography through a corresponding DT element. For the example depicted in Fig. 3, this means that the middleware has to first identify the DataObjectInstance and DataElementInstance entities of data

elements $\mathrm{de}_h$ of cross-partner data object $\mathrm{DO}_x$ and $\mathrm{de}_i$ of cross-partner data object $\mathrm{DO}_y$ based on the correlation properties of a choreography instance. Based on that, the DataValue entities associated to the resulting data element instances can be resolved within the middleware. Input parameters defined with DT elements can be passed as defined to the DT Implementation. After the data of all required inputs are resolved the specified DT Implementation can be invoked. As soon as the transformation of the data is completed, the DT Implementation replies the results back to the TraDE Middleware which stores the data in respective DataValue entities. These resulting DataValue entities are then associated to the corresponding instance entities (DataObjectInstance, DataElementInstance) of the cross-partner data objects specified as output of the DT element. For the example depicted in Fig. 3, this means that the middleware resolves the DataObjectInstance and DataElementInstance entities of data element $\mathrm{de}_k$ of cross-partner data object $\mathrm{DO}_z$ based on the correlation properties of a choreography instance and associates the DataValue entity that holds the transformation result data to them. Therefore, the transformation result data is available at the TraDE Middleware through the cross-partner data object specified within the choreography model for further use or processing.

As introduced in Section III-A modelers can specify an *Activation Mode* and a *Trigger Condition* for a DT element which influences the behavior of the TraDE Middleware regarding the execution of modeled data transformations. Based on the specified Activation Mode the triggering will take place in one of the following two ways.

If the Activation Mode is set to *on-write*, the TraDE Middleware initially waits until all inputs of a DT element, i. e., DataValue entities associated to cross-partner data objects being the source of incoming cross-partner data flows of the DT element, are successfully initialized. As soon as all related events are emitted, i. e., the transformation inputs are available, the TraDE Middleware triggers the specified DT Implementation and passes the data as described above. Furthermore, whenever one or more of the inputs of a DT element are modified and therefore a corresponding event is emitted, the TraDE Middleware again triggers the underlying data transformation using the updated input data. This guarantees that the specified outputs of a DT element provide always up-to-date data. While this is beneficial in cases where the transformation output data objects are read frequently, it might introduce a lot of unnecessary work in cases where the input data objects are often modified but the output data objects are only rarely read. Based on the example shown in Fig. 3, this means that the DT Implementation is triggered as soon as both cross-partner data objects $\mathrm{DO}_x$ and $\mathrm{DO}_y$ are initialized or whenever one of them is modified no matter if $\mathrm{DO}_z$ is read or not. By default, the Activation Mode on-write is applied within the middleware.

Setting the Activation Mode to *on-read* instead changes the behavior of the TraDE Middleware so that it triggers a data transformation on-demand on a read request to one of the output data objects of a DT element, i. e., cross-partner data objects being the target of outgoing cross-partner data flows of the DT element. Based on the example shown in Fig. 3, this means that the DT Implementation is triggered whenever $\mathrm{DO}_z$ is read using the current values of the associated transformation input data objects $\mathrm{DO}_x$ and $\mathrm{DO}_y$. In case all required input data is available, i. e., the DataValue entities associated to cross-partner data objects specified as transformation inputs are initialized, the TraDE Middleware triggers the specified DT Implementation and passes the data as described above. If one of the specified source data objects is not initialized yet, the TraDE Middleware will block the invocation of the DT Implementation until all the required data is available. This is beneficial in cases where transformation output data objects are read only rarely and inputs often change, while it introduces the overhead of running the DT Implementation to each read access on the output cross-partner data object.

Since sometimes it does not make sense to trigger a data transformation just because all required input data exists, the specification of a *Trigger Condition* allows to provide a more fine-grained specification on when to actually trigger a data transformation or not. For example, this is required in cases where an input data value should be within certain margins, e. g., above a threshold, or if a collection data element (i. e., a data element that can hold a collection of values, like an array structure) needs to have a certain number of DataValue entities associated before the specified transformation can be triggered. Therefore, the TraDE Middleware evaluates the logical expressions provided as Trigger Conditions at DT elements. The definition of a related expression language on top of the metamodel of the TraDE Middleware to specify and evaluate such trigger conditions for DT elements during modeling and run time is planned for future work.

## IV. Prototype

For the modeling of data-aware choreographies and to support the introduced data transformation concepts, we extended the choreography modeling language BPEL4Chor [13] and use BPEL [14] for the modeling of the private process models. The Data-aware Choreography & Orchestration Modeling Environment is built on existing tools, i. e., *Chor Designer* [15] and an extended version of the Eclipse *BPEL Designer*. The resulting environment also provides required model transformation capabilities [16] to transform a BPEL4Chor choreography model into a collection of BPEL process models.

An extended version of the open source BPEL engine Apache *Orchestration Director Engine* (ODE) is used as Process Engine. The execution of cross-partner data flows is realized by integrating Apache ODE with our TraDE Middleware to enable the reading and writing of cross-partner data objects [11].

The TraDE Middleware itself is realized as a Java-based web server which exposes its functionality through a REST API. We therefore use Eclipse *Jetty* in embedded mode. The REST API is specified and documented using Swagger and implemented based on the *Jersey* RESTful Web Services framework. For the persistence of the TraDE internal representations and the actual data processed within the choreographies, we support MongoDB as a document-oriented database and the local file

system. For the implementation of the event-driven triggering of data transformations within the middleware Apache Camel is used. The support of trigger conditions and activation modes for DT elements within the TraDE Middleware is planned for future work. The complete open source code of the TraDE Middleware is available on GitHub[1].

## V. Related Work

The model-driven approach by Meyer et al. [7] supports the modeling and enactment of data exchange and data transformations in choreographies on the level of 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. To model and enact data transformations between messages and local data they refer to standard data query languages, e. g., XQuery [17].

Habich et al. [18] provide related concepts on the level of process models and BPEL in particular to resolve the issue of centrally controlled and implicitly modeled data flow in BPEL. 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. The concepts together allow to integrate specialized data propagation and transformation logic, e. g., specified using Extract Transform Load (ETL) tools, as 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 and enable its transformation.

Both of these works do not provide an integrated, generic solution for the modeling and execution of data transformations within service choreographies or orchestrations, respectively. Our focus is on defining data transformations on the level of choreography models and enable their execution in a transparent manner, decoupled from the control flow of choreography participants. To the best of our knowledge, any other related work with focus on data-related aspects of service choreographies or orchestrations, e. g., Barker et al. [8], rely on the explicit modeling of data transformation logic in form of adding corresponding tasks to the underlying choreography models.

## VI. Conclusion and Outlook

To further strengthen our notion of data-aware choreographies, we extended our TraDE concepts to support the modeling of data transformations in service choreographies. Therefore, we introduced a modeling extension for data-aware choreographies which allows to define data transformations on the level of choreography data decoupled from choreography control flow. Furthermore, the execution semantics of the introduced data transformation modeling extension and how choreography data is actually transformed during choreography execution in a data-driven manner is described as a blueprint for a supporting run time environment. Finally, our prototypical implementation of an underlying TraDE ecosystem is outlined.

In future work, we want to provide the required support and concepts for the integration and invocation of heterogeneous data transformation logic as means to provide end-to-end support for data transformations in data-aware service choreographies. One of the core challenges there is to provide concepts for the technology independent specification, packaging and invocation of heterogeneous data transformation logic in an easy and reusable manner through the TraDE Middleware. Moreover, we want to improve and extend the introduced features and capabilities for data transformations for both choreography modeling and execution. For example, introducing an expression language to support the specification of transformation trigger conditions for more fine-grained control when a data transformation should take place.

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


## Acknowledgment

This research was supported by the projects SmartOrchestra (01MD16001F) and SePiA.Pro (01MD16013F).


## References


[1] R. Schmidt *et al.*, "Big Data as Strategic Enabler - Insights from Central European Enterprises," in *Business Information Systems*, 2014.

[2] S. Meyer *et al.*, "Towards Modeling Real-world Aware Business Processes," in *WoT*, 2011.

[3] O. Zimmermann, "Microservices tenets," *Computer Science - Research and Development*, 2016.

[4] A. Bouguettaya *et al.*, "A Service Computing Manifesto: The Next 10 Years," *Communications of the ACM*, 2017.

[5] F. Leymann and D. Roller, *Production Workflow - Concepts and Techniques*. PTR Prentice Hall, 2000.

[6] G. Decker *et al.*, "An Introduction to Service Choreographies," *Information Technology*, 2008.

[7] A. Meyer *et al.*, "Automating Data Exchange in Process Choreographies," *Information Systems*, 2015.

[8] A. Barker *et al.*, "Choreographing Web Services," *IEEE Transactions on Services Computing*, 2009.

[9] M. Hahn *et al.*, "Modeling and Execution of Data-Aware Choreographies: An Overview," *Computer Science - Research and Development*, 2017.

[10] OMG, "Business Process Model And Notation (BPMN) Version 2.0," Jan. 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0/

[11] M. Hahn *et al.*, "TraDE - A Transparent Data Exchange Middleware for Service Choreographies," in *OTM Conferences*, 2017.

[12] W3C, "XML Schema Definition Language (XSD) 1.1 Part 1: Structures," 2012. [Online]. Available: http://www.w3.org/TR/xmlschema11-1/

[13] G. Decker *et al.*, "Interacting services: from specification to execution," *Data & Knowledge Engineering*, 2009.

[14] OASIS, "Web Services Business Process Execution Language Version 2.0," 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[15] A. Weiß *et al.*, "Modeling Choreographies using the BPEL4Chor Designer," University of Stuttgart, Technical Report 2013/03, 2013.

[16] P. Reimann *et al.*, "Generating WS-BPEL 2.0 Processes from a Grounded BPEL4Chor Choreography," Universität Stuttgart, Technischer Bericht 2008/07, 2008.

[17] W3C, "XQuery 3.1: An XML Query Language," 2017. [Online]. Available: https://www.w3.org/TR/xquery-31/

[18] D. Habich *et al.*, "BPEL$^{DT}$ - Data-Aware Extension for Data-Intensive Service Applications," in *Emerging Web Services Technology*, 2008.


All links were followed on 2018-07-19.