Approach and Refinement Strategies for Flexible Choreography Enactment

Andreas Weiß, Santiago Gómez Sáez, Michael Hahn, Dimka Karastoyanova

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{andreas.weiss, santiago.gomez-saez, michael.hahn, karastoyanova}@iaas.uni-stuttgart.de

```
@inproceedings{INPROC-2014-60,
  author    = {Andreas Wei{\ss} and Santiago G{\'o}mez S{\'a}ez and Michael
               Hahn and Dimka Karastoyanova},
  title     = {{Approach and Refinement Strategies for Flexible Choreography
               Enactment}},
  booktitle = {Proceedings of 22nd International Conference on
               Cooperative Information Systems (CoopIS 2014),
               27-31 October 2014, Amantea, Italy},
  year      = {2014},
  pages     = {1--18},
  series    = {Lecture Notes in Computer Science (LNCS)},
  publisher = {Springer-Verlag}
}
```

Approach and Refinement Strategies for Flexible Choreography Enactment

Andreas Weiß, Santiago Gómez Sáez, Michael Hahn, and Dimka Karastoyanova

Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Stuttgart, Germany
`{andreas.weiss,santiago.gomez-saez,michael.hahn,
dimka.karastoyanova}@iaas.uni-stuttgart.de`

Abstract. Collaborative, Dynamic & Complex (CDC) systems such as adaptive pervasive systems, eScience applications, and complex business systems inherently require modeling and run time flexibility. Since domain problems in CDC systems are expressed as service choreographies and enacted by service orchestrations, we propose an approach introducing placeholder modeling constructs usable both on the level of choreographies and orchestrations, and a classification of strategies for their refinement to executable workflows. These abstract modeling constructs allow deferring the modeling decisions to later points in the life cycle of choreographies. This supports run time scenarios such as incorporating new participants into a choreography after its enactment has started or enhancing the process logic of some of the participants. We provide a prototypical implementation of the approach and evaluate it by means of a case study.

Keywords: Process Flexibility, Choreography Flexibility, Refinement Strategies, Late Modeling, Late Selection, Process Fragments

1 Introduction

In our research in Collaborative, Dynamic & Complex systems (CDC) [3] we aim at supporting several application domains exhibiting overlapping requirements, but so far diverging solutions, with unified concepts, techniques and tools. The domains we consider are scientific experiments and workflows, pervasive adaptive systems, service networks, configurable multi-tenant applications (see also Sec. 2.1). We have introduced a three-phase life cycle of CDC systems comprising Modeling, Provision, and Execution phases (cf. Sec. 3). To capture the overall communication behavior of applications in such systems, we use choreographies. Choreographies are a concept known from the business domain that enables independent organizations to collaborate and reach a common business goal. They provide a global view on the interconnection of independent, but collaborating organizations, which are denoted by choreography participants, communicating without a central coordinator [8]. The publicly visible communication interfaces of each choreography participant are enacted, i.e., implemented by services or orchestrations of services (workflows). In previous work, we have enabled the enactment of choreographies modeling CDC systems through a mapping/transformation to abstract workflows (containing only the communication activities), and their manual refinement to executable service orchestrations for the above mentioned domains [3]. This implies that the CDC Modeling phase is divided into a choreography modeling, a transformation, and a workflow modeling phase. A central concept of CDC systems is flexibility. In our work and in numerous related works, concepts, methods and techniques for flexible workflows have been studied in detail [24], [25]. Generally, the flexibility aspects can be divided into means to arbitrarily change workflows during modeling or run time [19], [24] or means for changes in predefined regions (cf. Sec. 2.3). Flexibility in choreographies, however, while instrumental for CDC systems, has not yet been studied as much as workflow flexibility. The few existing works on arbitrary choreography adaptation, changes and change propagation such as [10], [21], [27] are only one aspect of choreography flexibility. Towards reaching the goal of more choreography related flexibility, we propose an approach in Sec. 3 that introduces abstract placeholders/constructs, namely abstract activities and abstract connectors, on the level of choreographies in order to defer choreography modeling decisions to a later point in time. That means, our approach enables changes in predefined regions considering also the choreography level and refinement of the regions in all CDC life cycle phases. We call this end-to-end flexibility. We identify a set of strategies to refine the abstract constructs and position them with respect to the life cycle of CDC systems. The need for placeholders and their refinement not only on the orchestration level but also on the level of choreographies is explicitly motivated using three application domains (Sec. 2.1) and an motivating example (Sec. 2.2) to derive a set of requirements. These requirements are compared to related work to show the research gap we aim to fill with our approach (Sec. 2.3). Furthermore, we present a prototypical implementation of our approach (Sec. 4), which includes the extension of the choreography language BPEL4Chor [9] and the workflow language BPEL [18], and evaluate the approach in a case study. The paper is concluded with a summary and outlook to future work (Sec. 5).

2 Motivation and Related Work

2.1 Motivating Scenarios

In our research we are working towards enabling IT support for scenarios from three different fields: scientific experiments, collective adaptive systems, and configurable multi-tenant applications. We have already shown that these areas impose overlapping requirements on the supporting applications systems [3] and we introduced the notion of CDC systems as a type of applications fulfilling these requirements. Since in this work we focus on the *dynamic* aspect of CDCs, here we will identify the requirements on CDC systems that, when fulfilled, will significantly improve their flexibility.

Scientific (multi-scale and multi-field) experiments: In our work in the scope of the Cluster of Excellence Simulation Technology (SimTech[1]) we want to enable

[1] SimTech: `http://www.iaas.uni-stuttgart.de/forschung/projects/simtech/`

scientists to model and execute multi-scale and multi-field experiments in a user friendly manner. Previously [24], concepts and a workflow management system for supporting the trial-and-error nature of scientific simulations have been developed. Workflows that are not completely specified can be started and altered during run time. These simulation workflows orchestrate scientific services dealing with one scientific field which operates on one scale. For scientists, however, it is extremely important to be able to couple scientific models representing distinct scientific fields, such as biology, chemistry, and physics, and operating on different length scales, such as micro, macro or continuum scale, just by combining the already available simulation applications in a more complex application that represents the multi-scale and multi-field experiment. For this, we model the overall experiment as a choreography which is enacted by simulation workflows. The support for the trial-and-error manner of interleaved modeling and execution of experiments and the cooperative nature of the interactions among scientists is also particularly important. When collaboratively modeling the multi-scale and multi-field experiment as a choreography, scientists may want to define with an abstract placeholder that certain simulation steps have to be conducted such as building a Finite Element Method (FEM) but the concrete specification is left open for the particular scientist with the expertise to refine the enacting simulation workflow. The abstract placeholder defined on the choreography level would provide some guidance for refinement on the workflow level. The refinement of the abstract placeholders may even be conducted after the experiment has been started. This would allow to take intermediate results into account when deciding on the next concrete steps in a choreography of scientific workflows. Furthermore, in order to hide technical details of the communication between different scientific fields and scales, facilities to define communication relationships in an abstract way and refine them later would be helpful for scientists not familiar with technical details of the execution environment.

Collective Adaptive Systems (CAS) comprise heterogeneous entities that collaborate towards achieving their own objectives, and the overall objective of the collective [2]. These entities can be either virtual or physical, and organizationally and geographically distributed. The interaction of such entities with the collective highly influences the behavior of the system, as individual entity objectives may conflict with the behavior or decisions of other entities. Furthermore, unexpected changes in the entities' environment may trigger an individual or collective behavioral change. For purposes of providing a system capable of supporting this behavioral definition, monitoring, and adaptation in the EU ALLOW Ensembles project[2], we define the underpinning concepts for modeling, execution, and adaptation of CAS entities and their interactions. In particular, we propose to model and manage entities as collections of cells encapsulating their functionalities; cells are realized by service orchestrations. Entities collaborate with each other to achieve their objectives in the context of ensembles, enacted as choreographies, describing the interactions among them [2]. CAS are based on fundamental concepts from the so-called Adaptable Pervasive Flows (APF) [7], [13]. Adaptability

[2] ALLOW Ensembles: `http://www.allow-ensembles.eu`

of CAS is required on the level of both cells and ensembles and hence has to be supported by means of flexible orchestrations and the interactions among them in choreographies. The cells' behavior can be partially or completely specified during the modeling phase. The partial specification of the cell behavior covers the partial definition of one or more of their tasks as abstract tasks, which must be refined by concrete tasks during run time.

Configurable Workflows is a research area that we investigate towards enabling multi-tenant applications, i.e., applications that are designed to maximize the resource sharing between multiple consumers. Configurable applications are capable to adapt their behavior or appearance without the necessity to change the underlying implementation [11], [14] to the needs of entities (e.g., tenants, tenant users). The *multi-tenant aware Service Composition Engine* (SCE^{MT}) introduced in [12] is one example of a SCE that can be shared by different entities. Furthermore, it enables users to customize predefined workflow models based on their needs by providing a set of configuration data without the necessity to create new or adapt existing workflow models. These configuration data are used in all instances of a workflow model which belong to a corresponding user. Currently, SCE^{MT} supports the registration of runtime data (e.g., variable or service endpoint values). On the one hand, this enables users to customize an existing model by overriding predefined values (e.g., constants specified in a workflow model). On the other hand, workflow modelers are able to define more generic and reusable models (process templates) which can be adapted by the users themselves through configuration. For example, any country specific values of a payment process such as tax rates can be dynamically replaced during run time by using registered model configurations related to the user who instantiates the workflow model. In order to provide a user the ability to customize parts of the control flow logic of a workflow model, a modeler should be able to insert placeholders into a workflow model. The placeholders can be refined with the registered logic of a user during a later point in the life cycle of the workflow. This provides a much more general, flexible and powerful solution for the configuration of workflow models because a user would be able to customize the control flow and therefore the behavior of a workflow model. Configurability is also required on the level of choreographies which would enable the configuration of the entire set of interconnected workflow models by a user.

2.2 Example and Requirement Identification

Fig. 1 shows a trip booking example from the CAS domain to illustrate the need for placeholder flexibility constructs beginning from the choreography level to the execution of the enacting workflows. The example forms a choreography consisting of three participants showing only their public interfaces: the passenger, the passenger management system, and a payment manager. The passenger participant initiates a conversation with the passenger management system exchanging trip and payment details. After specifying the payment details, the payment manager participant comes into play. Let's assume that the modeler of the choreography only wants to model a communication relationship between the passenger management system and the payment manager and does not care about the technical details of the communication. That means for example that it is left unspecified by the modeler if the communication pattern is synchronous or asynchronous. Furthermore, the choreography modeler only knows that the payment manager participant will somehow conduct a payment depending on the payment preferences of the passenger. However, it is left unspecified how the payment will be processed and whether other participants will join the choreography to execute the payment. To realize this kind of flexibility that allows the specification of an abstract placeholder for the logic to invoke services or other participants during the Modeling phase of a CDC system but defers the decision about the concrete workflow logic to a later point in time (at latest to run time), two concepts are needed: (i) a placeholder for workflow logic and (ii) a logical connector between choreography participants that can be refined later.

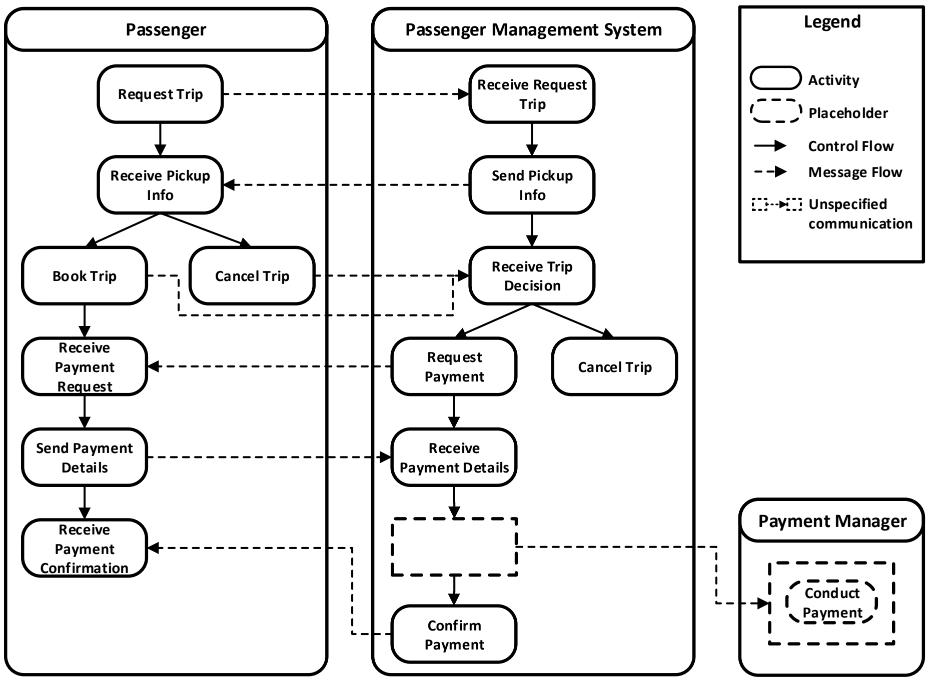


Fig. 1: Trip booking example from the Collective Adaptive System domain

Based on the three application domains in general and the motivating example in particular, we identify the following requirements for an approach to enable the discussed end-to-end flexibility features on both the level of choreographies and the enacting workflows.

R1. **Choreography placeholders**: Choreography modelers need capabilities for specifying placeholders in their models to defer decisions about concrete orchestration steps. The refinement of the placeholders must be feasible in any CDC life cycle phase.

R2. **Workflow placeholders:** Workflow modelers refining the generated non-executable workflows which enact a choreography must also have access to placeholder constructs to defer the decision about orchestration steps to the Provision or Execution phase.

R3. **Same semantics:** The placeholder constructs should have the same semantics on both the choreography and workflow level and modelers should be able to use them consistently on both levels.

R4. **Manual and automatic refinement:** The refinement of placeholders with process fragments must be possible both manually by human modelers and automatically through a model (choreography or workflow) fragment discovery operation.

R5. **Execution of unrefined workflows:** Workflows having unrefined placeholders must still be executable, i.e., the workflows must be instantiable and executable until a placeholder is reached. Then either a manual or automatic refinement must take place.

R6. **Phase annotation:** The placeholder must be annotated with information in which life cycle phase the refinement has to take place. The CDC system has to enforce the annotated information.

R7. **Abstract definition of collaborations:** It must be possible for choreography modelers to specify the collaboration between participants in an abstract manner to defer the decision about the technical realization in terms of transport protocol and Message Exchange Pattern (MEP). This helps non-technical users of the system to avoid dealing with technical details.

R8. **Discovery of participants:** The definition of abstract collaborations of choreography participants which are unknown at design time but dynamically discovered during run time must be possible. That means an placeholder indicates the communication with an not yet known participant.

R9. **Generality:** The concept of the placeholders should be generic and not tailored to a specific application domain, but also support the idiosyncrasies of different domains.

2.3 Review of Existing Concepts

The central concept needed by all three presented domains is an abstract placeholder (on the level of choreographies and workflows) that can be refined with concrete workflow logic even after the Modeling life cycle phase. The workflow logic for refinement contains either single tasks, also called activities, or process fragments, which are a set of activities connected by control and data flow. Existing work uses the concept of predefined regions representing placeholders in workflow models that can be refined after the instantiation of the model. The refinement is denoted by *Late modeling of process fragments* in case the activities or process fragments are newly specified inside a placeholder during run time and *Late selection of process fragments* in case a set of activities or process fragments has been pre-modeled but the actual selection happens during run time based on predefined rules or user decisions [25]. There are several approaches realizing these generic patterns. In [1], the notion of worklets, which are completely specified YAWL processes that refine a worklet enabled parent YAWL task at run time, is used for workflows in order to provide workflow run time flexibility. The selection of the most appropriate worklet for a worklet enabled task is based on rules considering context information. In [22], the concept of Pockets of Flexibility (PoF) is introduced. The PoF which contain a set of activities and the control flow between the activities are defined during run time, either (semi-)automatically or manually depending on previously executed activities. Ardissono et al. use a planning based approach for retrieving sub-processes implementing a so-called abstract activity [4], while in [6] from the field of pervasive flows planning based techniques are used to generate a sub-process. The domain of Adaptable Pervasive Flows (APF) [7], [13] also uses the concept of abstract activities which are refined depending on the context of the entities represented by a workflow.

All these approaches have in common that they concentrate on providing flexibility on the level of individual workflows. They fulfill the requirement R2 for having a workflow placeholder, R5 for being executable even with unrefined regions, R4 demanding manual or automatic refinement, and are mostly generic and not limited to a particular domain as stated in R9. However, our scenarios explicitly demand the concept of predefined regions on the level of choreographies in order to give guidance for workflow refinement, to incorporate further participants during later life cycle phases or defer the decision about communication patterns to later phases (R1, R3, R6, R7, R8). To the best our our knowledge, these requirements are not fulfilled by the existing concepts.

Regarding choreography flexibility in general, i.e., without the use of placeholders, several approaches exist. For example, in [21] the propagation of changes appearing in the private process of one choreography participant to the affected business partners is enabled without considering already running choreographed workflow instances. Formal methods are used to calculate if and what changes are necessary and if the new message interchange is deadlock free. In [27], an improved formal approach is introduced for change propagation from choreographies to orchestrations. In [10], a generic approach for propagation of local changes to directly as well as transitively dependent business partners is shown. A framework for collaborative choreography customization is presented in [16]. The so-called global (choreographies) and local views of each participant are expressed in the agent-oriented software development methodology Tropos. Participants agree on customization alternatives that best fulfill their business needs, i.e., their local view. Basically, such approaches enable the change of already existing choreography models and propagate changes to the enacting workflows and services. They are orthogonal to our requirement for abstract modeling constructs in order to enable end-to-end flexibility through late modeling and selection for choreographies and their enacting workflows. To the best of our knowledge, there are no available approaches satisfying this requirement. In Sec. 3 we propose a new approach to close this research gap.

3 End-to-End Flexibility and Refinement Strategies

Based on the requirements identified in Sec. 2.2, we present our approach for providing end-to-end flexibility throughout the Modeling, Provision, and Execution life cycle phases of CDC systems which are realized through choreographies and orchestrations. We propose the use of *abstract constructs* during the different life cycle phases of a CDC system. These abstract constructs are denoted in the scope of our work as abstract activities and abstract connectors.

3.1 Abstract Constructs

We define an *abstract activity* as a placeholder in a choreography or workflow model, which should be refined to concrete choreography or orchestration logic immediately before its execution at the latest. An abstract activity is connected to the preceding and succeeding activities by control and data flow. Definition 1 expresses formally the properties of an abstract activity.

Definition 1 (Abstract Activity). *An abstract activity* $\mathfrak{a}$ *is represented by the tuple* $\mathfrak{a} = (n, \alpha, \omega, \tau, \epsilon, T, \sigma)$ *where* n *is the name of* $\mathfrak{a}$*,* α *specifies the life cycle phase in which the refinement of* $\mathfrak{a}$ *may first be conducted,* ω *specifies the life cycle phase in which the refinement of* $\mathfrak{a}$ *has to be conducted at latest,* τ *is the plugin type of* $\mathfrak{a}$*,* ϵ *is the failure handling strategy for* $\mathfrak{a}$*,* T *denotes the type-specific content of* $\mathfrak{a}$*, and* σ *is the plugin selection function* $\sigma : \tau \mapsto T$*, which maps the plugin type* τ *to the type-specific content* T*. For* α *and* ω *the following applies:* $\alpha, \omega \in \{\text{"choreography\_modeling"}, \text{"workflow\_modeling"}, \text{"provision"}, \text{"execution"}, \emptyset\}$*. For the failure handling strategy* ϵ *applies:* $\epsilon \in \{\text{"none"}, \text{"retry"}, \text{"manual"}\}$*.*

An abstract activity $\mathfrak{a}$ is a generic placeholder that can be typed through the plugin type attribute τ. This attribute allows the assignment of different implementations realizing different refinement behavior or simply indicates that the refinement has to be conducted manually. The plugin selection is formally defined by the plugin selection function σ mapping the plugin type of τ to the type-specific content T of the abstract activity (i.e., the plugin). The plugin selection function σ is important for providing the correct execution plugin in a workflow engine and the correct user interface plugin for modeling in a modeling tool. For example to take into account the context and objectives of the entities represented by the workflows for the refinement of an abstract activity, the value $APF\_AdaptationManager$ could be assigned for τ. This implies that the type-specific content T is populated with concepts from the Adaptable Pervasive Flows domain (APF) [7], [13]. For example, *preconditions* and *effects* attributes are introduced that specify the desired behavior of the logic the abstract activity represents without limiting it to predefined execution patterns. The preconditions specify in which state the context of the particular workflow has to be prior to the refinement of an abstract activity. The effects specify the desired state after the execution of the refined logic. A workflow engine implementation supporting the $APF\_AdaptationManager$ plugin expects the specified type-specific attributes and contacts an external adaptation manager to retrieve a process fragment based on the context values. For other domains τ and T may have completely different values. The failure handling strategy ϵ of an abstract activity specifies how a supporting workflow engine should behave if $\sigma(\tau) = \emptyset$ in case an appropriate plugin is not installed in the engine. Possible strategies could be retrying the selection or asking a user for resolution.

The discovery of an appropriate fragment is the responsibility of the selected plugin, i.e., the type-specific content T of an abstract activity. The discovery is formally defined in Definition 2.

Definition 2 (Fragment Discovery). *The fragment discovery function* $\delta : T \mapsto f$ *is a function that maps the type-specific content* T *to a model fragment* $f \in F$ *where* F *is a set of process or choreography fragments. We denote the application of* δ *as the refinement of the abstract activity* $\mathfrak{a}$.

Each plugin implements the function δ offering different discovery methods/ algorithms for example for manual or automatic selection. A further advantage of using a plugin concept is the possibility to specify different refinement/discovery methods inside an individual choreography or workflow model.

An *abstract connector* is a composite modeling element for choreographies that consists of two so-called *abstract containers* and a communication link. The abstract containers represent the source and the target of the communication link connecting the involved choreography participants. If one participant has to send messages to more than one participant, this is not modeled by an abstract connector but rather the choreography language itself has to provide means for expressing a set of participants whose number is not known at design time [9]. An abstract connector can be modeled between sending and receiving participants (or sets of them) in order to postpone the decision about the MEP to be used. Definition 3 describes the properties of an abstract connector formally.

Definition 3 (Abstract Connector). *An abstract connector* $\mathfrak{c}$ *is represented by a tuple* $\mathfrak{c} = (n, C_{source}, C_{target}, \lambda, \alpha, \omega, \tau, \epsilon, T)$ *where* n *is the name of* $\mathfrak{c}$, C_{source} *is the abstract container that represents the source of* $\mathfrak{c}$, C_{target} *is the abstract container that represents the target of* $\mathfrak{c}$, *and* $\lambda : C_{source} \mapsto C_{target}$ *is the connector link of* $\mathfrak{c}$. *An abstract container* C *is itself a tuple* $C = (n_C, \mathfrak{M})$ *where* n_C *is the name and* $\mathfrak{M}$ *is the content of the abstract container which may be empty or contain all elements for specifying control and data flow in the used choreography or orchestration language, as well as abstract activities and nested abstract containers. The remaining elements* α, ω, τ, ϵ, *and* T *have the same semantics as defined in Definition 1.*

The abstract containers C_{source} and C_{target} define the region in which the message exchange between two choreography participants has to take place. The content $\mathfrak{M}$ of an abstract container C may be empty and thus only expressing the communication relationship between two participants or contain communication constructs, abstract activities, or nested abstract containers. The optional plugin type τ and the corresponding type-specific content T of an abstract connector

$\mathfrak{c}$ define how the source and the target abstract containers C_{source} and C_{target} have to be refined, i.e., the MEPs, the derived roles for the participants, and the implementation in terms of communication protocols and middleware. During refinement of an abstract connector, communication activities are placed in the abstract containers . This is also realized by a corresponding implementation of the function δ.

3.2 Refinement Strategies

The main focus of our approach is to provide modeling constructs that span from choreography modeling to execution of the enacting workflows and to identify refinement strategies, which will be positioned within the CDC life cycle. While identifying refinement strategies and suggesting the use of some refinement mechanisms such as the refinement with process fragments, it is not our goal here to provide decisions support for finding the most suitable process fragments for refining an abstract activity or an abstract connector. That means we do not discuss and provide implementations for the function δ in this paper but leave it open for future work.

Fig. 2 shows the phases of our approach and the models and artifacts needed as input to and produced as output by each phase. Additionally, we also show the use of abstract constructs and their potential refinements throughout all phases. For every step in our approach the following is true: Abstract constructs are annotated by its modeler with information in which steps of our approach the refinement is allowed to be carried out (thus addressing requirement R6). The information defines a range of approach steps, i.e., in which step the refinement can be conducted first and in which step the refinement has to be conducted at the latest. This relates to the attributes α and ω in Definition 1. With the concept of the plugin type τ and the type-specific content T, i.e., the plugin concept, of an abstract construct, we fulfill requirement R9 since the abstract constructs are generic but provide possibilities to consider the idiosyncrasies of different domains.

The first three phases of Fig. 2 belong to the CDC Modeling phase. A domain problem (Fig. 2, (1)) is transformed into a choreography (Fig. 2, (2)) through collaborative *manual modeling* using a choreography editor. We identify the following cases where abstract constructs can be used: (i) Choreography modelers add abstract activities to define placeholders for orchestration logic that has to be concretized later during workflow refinement, deployment or execution (R1). The modelers must specify the plugin type τ and, thus, type-specific content T, e.g., preconditions and effects relating to a choreography context model (R9). (ii) Choreography modelers specify abstract activities to defer the decision if and which participant should be part of the overall choreography to a following life cycle phase (R8). (iii) A choreography modeler specifies two or more communicating participants without defining the Message Exchange Pattern [17]. That means, it is only specified that the choreography participants will exchange messages during runtime but all details are left open (R7) (e.g., if an immediate response has to follow after a request or if fault messages could be thrown). To realize this

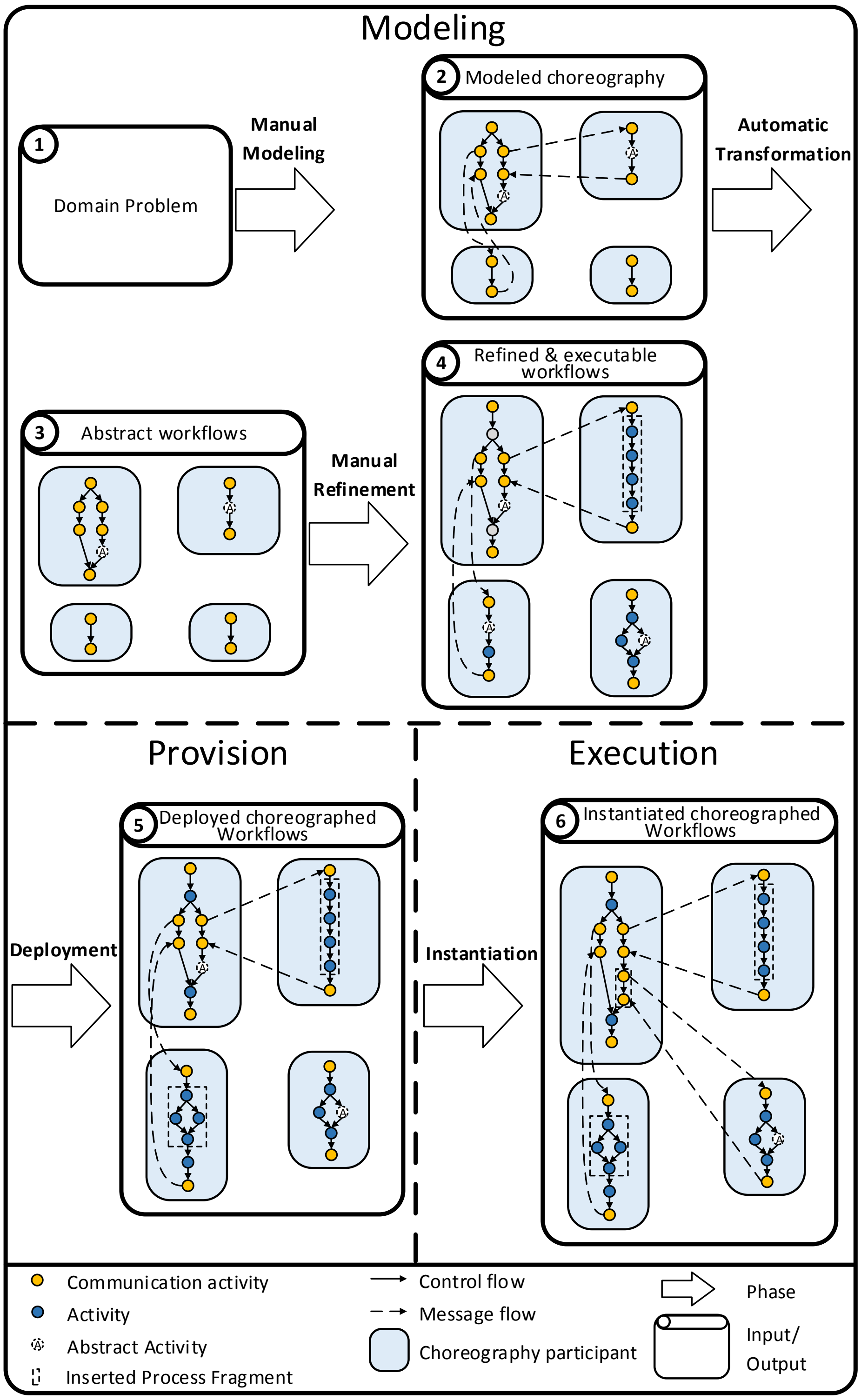


Fig. 2: Phases of the approach for enactment of flexible choreographies

we use the *abstract connectors*. The plugin type of the abstract connector can be used to specify a plugin realizing a specific MEP. If the plugin type is left empty by the choreography modeler, we envision an wizard that let the user answer predefined questions to determine an appropriate refinement. Since choreography modeling languages are often not executable [8], the modeled choreography is *automatically transformed* into an abstract representation (Fig. 2, (3)) of an executable workflow language. This representation is not yet executable. For each participant a separate workflow model is generated capturing the individual communication behavior through communication constructs such as sending and receiving activities. The abstract constructs modeled in the previous phase are embedded into the abstract workflow models. The communication links represented by the abstract connectors are expressed in the existing language capabilities of the target workflow language, e.g., *Partner Links* in the case of BPEL.

In the next phase the abstract workflow models are *refined manually* by domain experts, who add orchestration logic such as backend service invocations and variable assignments thus making the models executable (Fig. 2,(4)). We identify the following refinement cases that can take place during the workflow refinement phase: (i) The abstract activities inserted in the choreography modeling phase can be refined using workflow fragments from a repository and by placing the fragments on the abstract activity in a workflow modeling editor (R2). In this case the process modeler may optionally consider the type-specific content T for the fragment selection. Alternatively, an appropriate process fragment may be selected automatically by involving a recommendation service that contacts a external context-aware decision component for this purpose. Note that the inserted process fragments may also contain abstract activities which have to be refined immediately or during workflow run time. (ii) Abstract activities of a workflow model with communication related process fragments may be inserted, i.e., fragments containing sending and receiving activities. The actual collaboration partner has to be discovered during deployment or run time. (iii) Existing abstract activities may remain unchanged. (iv) Abstract connectors can be refined by inserting the necessary communication related process fragments in the corresponding abstract containers of the collaborating workflows.

The *deployment phase* of the enacting workflows is related to the CDC Provision phase. The executable workflow models are deployed (Fig. 2, (5)) on one or more workflow engines. The refinement possibilities during deployment are: (i) Modeled abstract activities can be refined automatically through the assistance of the external context-aware decision component that is contacted by a deployment manager component in order to retrieve an appropriate process fragment. (ii) Modeled abstract activities are refined manually by asking a user to provide appropriate process fragments. (iii) Abstract connectors can be refined by automatically inserting appropriate process fragments in the involved workflows. (iv) Abstract connectors can be refined by asking a user to insert appropriate process fragments in the involved workflows.

The last phase of our approach is the *Execution* phase of the CDC system. Possible refinements in this phase are: (i) Substitution of abstract activities with fragments or executable activities automatically. Hence, partially refined workflows can be executed, too (R5). Note that the process fragment may also contain further abstract activities. (ii) Manually specification of process fragments replacing an abstract activity using a workflow editor or retrieving the fragments from a repository. This implies pausing the execution and explicitly asking a user to provide the necessary refinement by considering the current context. (iii) Refinement of abstract connectors by inserting process fragments and/or executable activities in the affected workflow instances on both sides of the connector, i.e., refining the corresponding abstract containers. Note that abstract activities can also be placed in fault handling or compensation constructs on choreography and workflow level. The fault handling or compensation activities are refined using the same strategies as described above. To ensure that a choreography is still enactable and deadlock free the resulting choreography has to be verified [15] and the correctness of the enacting workflows guaranteed. Verification mechanisms, however, are not in the scope of this paper.

4 Realization

4.1 Required Language and Tool Extensions

In order to enable the modeling of different domain problems with our approach, a choreography language and a choreography editor are needed. As a basis for a CDC system modeling tool we use our ChorDesigner [26] as choreography editor and we have chosen BPEL4Chor [9] as the choreography language to serialize the choreographies in. BPEL4Chor forms a layer on top of BPEL and specifies the participants of a choreography, their communication behavior and the message links between them. The technical details of the communication are separated from the choreography. Because BPEL4Chor itself is not executable, we transform the choreography models to abstract BPEL processes [20] and refine them with orchestration logic. For the purpose of enabling our approach, extensions of both the choreography language and the modeling tool were required. BPEL4Chor and BPEL have been extended with the concept of abstract activities. Additionally, BPEL4Chor has been extended with the concept of abstract connectors.

Extending BPEL4Chor for flexibility: The communication behavior of a choreography participant is specified in BPEL4Chor by the so-called Participant Behavior Descriptions (PBD). PBDs are based on abstract BPEL processes with a more restricting profile excluding BPEL Partner Links. To enable more flexibility when modeling choreographies, we extended the BPEL4Chor language with abstract activities using the extension mechanism of BPEL to define an extension activity denoted as *abstractActivity* in the PBD. Listing 1 shows an example of an abstract activity already taking into account the APF domain (pluginType = APF_AdaptationManager) embedded into a BPEL4Chor PBD. The abstract activity represents the behavior of a payment participant, which is a part of a trip booking choreography from our motivating example in Sec. 2.2. The participant has to receive a payment request, process it and reply accordingly. Instead of specifying the payment orchestration logic, the PBD contains the extension activity `<abstractActivity>`. The *refinementStartPhase* and *refinementEndPhase* attributes indicate in this example that it can be done in the workflow modeling phase, or during deployment or run time. The refinement manner, i.e., manually or automatically, depends entirely on the plugin.

Listing 1: Extended BPEL4Chor Participant Behavior Description

```xml
<process name="payment" targetNamespace="urn:allowEnsembles:payment"
  abstractProcessProfile="urn:HPI_IAAS:choreography:profile:2006/12"
  xmlns:aa="urn:abstractActivity" xmlns:apf="urn:flows">
  <sequence>
    <receive wsu:id="receivePaymentRequest"/>
    <extensionActivity>
      <aa:abstractActivity name="ConductPayment"
          refinementStartPhase="workflow_modeling"
          refinementEndPhase="execution" pluginType="
          APF_AdaptationManager" failureStrategy="none">
        <aa:typeSpecificContent>
          <apf:entities/>
          <apf:preCondition/>
          <apf:effect/>
          <apf:goal/>
          <apf:compansationGoal/>
        </aa:typeSpecificContent>
      </aa:abstractActivity>
    </extensionActivity>
    <reply wsu:id="replyPaymentRequest"/>
  </sequence>
</process>
```

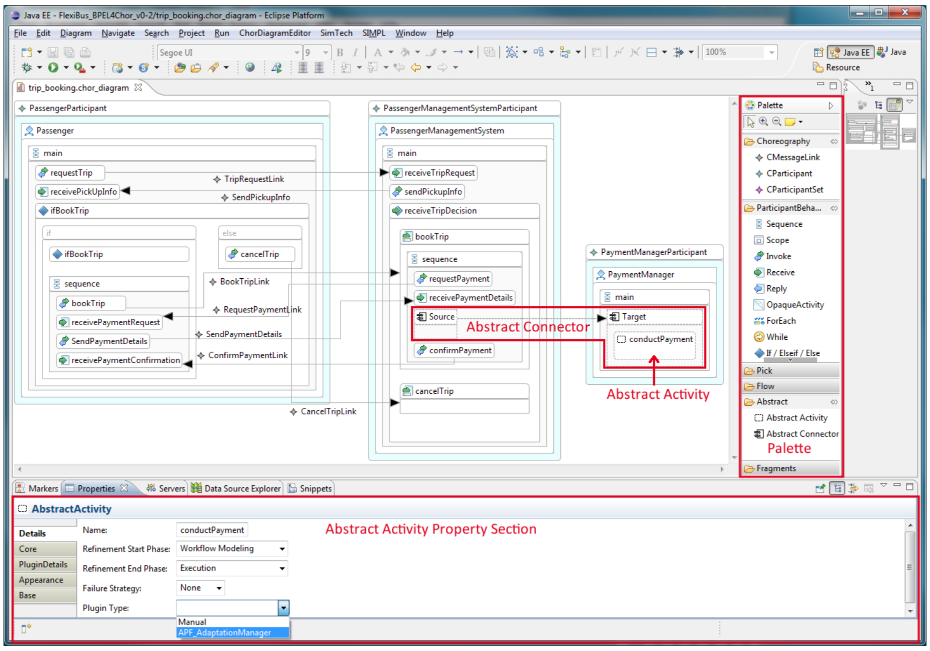
In the example of Listing 1, the plugin is of the type *APF_AdaptationManager* integrating entity, precondition, effects, goal, and compensations goal elements without specifying their details for the sake of brevity. *Abstract connectors* are realized by an extension activity in the PBDs representing source and target abstract containers of an abstract connector as well as by an extension of the BPEL4Chor Message Link concept with the attributes defined in Definition 3.

Extending BPEL for flexibility: We use the standard extensibility mechanism of BPEL and define an extension activity representing an abstract activity as a part of an executable BPEL process. By using the standard extensibility mechanism we remain compliant to the BPEL standard. The extension activity in a BPEL process has an identical structure as the one depicted in Listing 1 for the BPEL4Chor Participant Behavior Description. Furthermore, an additional extension activity represents the source and target abstract containers of an abstract connector in the BPEL workflows.

Extensions in the transformation from BPEL4Chor to BPEL: We have extended the transformation from BPEL4Chor to BPEL with the necessary mapping of the BPEL4Chor abstract activity to the BPEL abstract activity. BPEL4Chor Message Links that represent abstract connectors are transformed into BPEL Partner Links. The corresponding source and target abstract containers are transformed to their BPEL extension activity equivalent.

Plugin Concept: The realized abstract constructs specify the attributes defined formally in Sec. 3.1 and exhibit a plugin-point for type-specific content. In our Eclipse-based[3] implementation of the choreography [26] and workflow editor [23], the plugins consists of an EMF data model to describe the type-specific content and a user interface extension for each plugin. The user interface extension provides input fields for populating the type-specific content of the plugin. Depending on the type-specific content, the refinement of the abstract constructs, i.e., the application of the function δ, is either done manually or automatically. In our workflow engine SCE^{MT} [12], plugins can be registered implementing automatic refinement functions for different plugin types.

4.2 Case Study



AbstractActivity

Details | Core | PluginDetails | Appearance | Base

Name: conductPayment
Refinement Start Phase: Workflow Modeling
Refinement End Phase: Execution
Failure Strategy: None
Plugin Type:
Manual
APF_AdaptationManager

Fig. 3: Trip booking scenario modeled as choreography with our ChorDesigner

For the evaluation of our approach we use the motivating example from Sec. 2.2 to demonstrate how the modeling and manual refinement of the abstract activities and connectors have been enabled. Fig. 3 shows the trip booking scenario modeled as a choreography with the ChorDesigner. There are three participants connected with message links. Note that an abstract activity has been modeled in

[3] http://www.eclipse.org

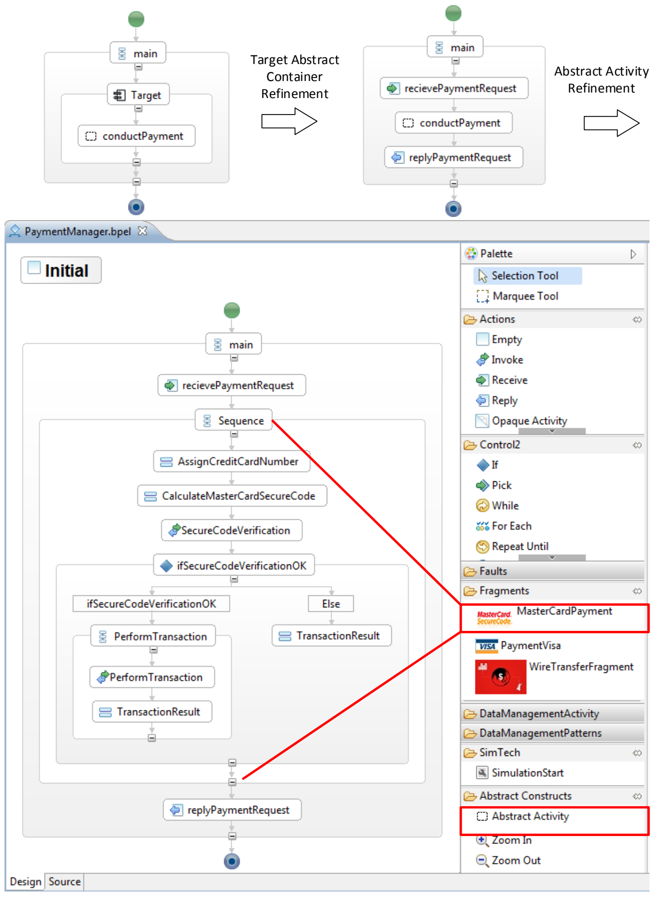


Fig. 4: Exemplary refinement of abstract constructs using the Mayflower BPEL Designer

the *PaymentManager* participant to provide a placeholder for the actual payment logic that can be refined during later phases. Furthermore, the interaction between the *Passenger Management System* and the *Payment Manager* is not explicitly specified via message links indicating an MEP, but rather with an abstract connector allowing to defer this decision to a later point in time. The tool set presented here allows for transforming the BPEL4Chor choreography into abstract BPEL processes [26]. All newly introduced choreography level abstract constructs are transformed into the corresponding constructs on the orchestration level; our system also generates valid WSDL interfaces. The refinement of abstract BPEL processes is conducted with the Mayflower BPEL Designer [23] and Fig. 4 shows some possible refinement steps of the generated BPEL process of the PaymentManager participant. First, the target abstract container is refined with communication activities, then the *conductPayment* abstract activity is replaced by a credit card process fragment. Additionally, our workflow management system is capable of refining abstract activities during run time on the workflow level [5] as well as arbitrary manual insertion of process fragments during run time [24]. Not yet realized but planned for the near future is the run time refinement of abstract connectors and abstract activities that implies that new participants are joining the choreography.

5 Conclusions and Future Work

In this work we have introduced an approach that uses abstract constructs (abstract activities and abstract connectors) and refinement strategies as means to provide end-to-end flexibility to Collaborative, Dynamic & Complex systems and their realization in terms of choreographies and enacting workflows. The approach permits the use of abstract constructs and their refinement to concrete logic – manually or automatically – throughout the whole life cycle of CDC systems. With this approach modeling decisions can be deferred to later points in the CDC life cycle – a requirement from different domains where CDC systems are applied, which has not been addressed by literature yet. With this approach we also support the use of many existing approaches for modeling and/or generation of concrete logic on the level of the enacting orchestrations. We plan to investigate the run time refinement of abstract activities and connectors on the level of choreographies and address the case of introducing new choreography participants during execution. Run time changes of choreographies and their propagation to the enacting workflows are also part of our future research.

Acknowledgment

This work is funded by the projects FP7 EU-FET 600792 ALLOW Ensembles and the German DFG within the Cluster of Excellence (EXC310) Simulation Technology.

References

1. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In: OTM Conferences (1). Springer
2. Andrikopoulos, V., Bucchiarone, A., Sáez, S.G., Karastoyanova, D., Mezzina, C.A.: Towards Modeling and Execution of Collective Adaptive Systems. In: Proceedings of WESOA'13. pp. 1–12. Springer (2013)
3. Andrikopoulos, V., Gómez Sáez, S., Karastoyanova, D., Weiß, A.: Collaborative, Dynamic & Complex Systems: Modeling, Provision & Execution. In: CLOSER'14. pp. 0–10. SciTePress (2014)
4. Ardissono, L., Furnari, R., Goy, A., Petrone, G., Segnan, M.: A framework for the management of context-aware workflow systems. In: WEBIST (1). pp. 80–87 (2007)
5. Bialy, L.: Dynamic Process Fragment Injection in a Service Orchestration Engine. Diploma Thesis No. 3564, University of Stuttgart, Germany (2014)

6. Bucchiarone, A., Marconi, A., Pistore, M., Raik, H.: Dynamic Adaptation of Fragment-Based and Context-Aware Business Processes. In: ICWS'12. pp. 33–41. IEEE (2012)
7. Bucchiarone, A., Lafuente, A.L., Marconi, A., Pistore, M.: A formalisation of adaptable pervasive flows. In: WS-FM'09. pp. 61–75. Springer (2009)
8. Decker, G., Kopp, O., Barros, A.: An Introduction to Service Choreographies. Information Technology 50(2), 122–127 (2008)
9. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for Modeling Choreographies. In: ICWS'07. IEEE (2007)
10. Fdhila, W., Rinderle-Ma, S., Reichert, M.: Change Propagation in Collaborative Processes Scenarios. In: CollaborateCom'12. IEEE (2012)
11. Guo, C.J., Sun, W., Huang, Y., Wang, Z.H., Gao, B.: A Framework for Native Multi-Tenancy Application Development and Management. In: CEC/EEE'07. pp. 551–558. IEEE (2007)
12. Hahn, M.: Approach and Realization of a Multi-tenant Service Composition Engine. Diploma Thesis No. 3546, University of Stuttgart, Germany (2013)
13. Herrmann, K., Rothermel, K., Kortuem, G., Dulay, N.: Adaptable Pervasive Flows - An Emerging Technology for Pervasive Adaptation. In: SASOW'08. pp. 108–113. IEEE (2008)
14. Krebs, R., Momm, C., Kounev, S.: Architectural Concerns in Multi-tenant SaaS Applications. In: CLOSER'12. pp. 426–431. SciTePress (2012)
15. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing bpel4chor: Verification and participant synthesis. In: Web Services and Formal Methods, Fourth Int. Workshop. pp. 77–91. Springer (2007)
16. Mahfouz, A., Barroca, L., Laney, R., Nuseibeh, B.: Requirements-Driven Collaborative Choreography Customization. In: Service-Oriented Computing, pp. 144–158. Springer (2009)
17. Nitzsche, J., van Lessen, T., Leymann, F.: Extending BPEL light for Expressing Multi-Partner Message Exchange Patterns. In: EDOC'08. pp. 245–254. IEEE (2008)
18. OASIS: Web services business process execution language version 2.0 (April 2007), `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`
19. Reichert, M., Rinderle-Ma, S., Dadam, P.: Flexibility in process-aware information systems. LNCS ToPNoC, Spec. Iss. on Concurr. in PAIS. 2, 115–135 (2009)
20. Reimann, P.: Generating BPEL Processes from a BPEL4Chor Description. Student Thesis No. 2100 (2007)
21. Rinderle, S., Wombacher, A., Reichert, M.: Evolution of Process Choreographies in DYCHOR. In: CooplS'06. pp. 273–290. Springer (2006)
22. Sadiq, S., Sadiq, W., Orlowska, M.: Pockets of Flexibility in Workflow Specification. In: Conceptual Modeling ER'01, pp. 513–526. Springer (2001)
23. Sonntag, M., Hahn, M., Karastoyanova, D.: Mayflower - Explorative Modeling of Scientific Workflows with BPEL. In: CEUR Workshop'12. pp. 1–5. Springer (2012)
24. Sonntag, M., Karastoyanova, D.: Model-as-you-go: An Approach for an Advanced Infrastructure for Scientific Workflows. Grid Computing 11(3), 553–583 (2013)
25. Weber, B., Reichert, M., Rinderle-Ma, S.: Change Patterns and Change Support Features - Enhancing Flexibility in Process-aware Information Systems. Data Knowl. Eng. 66(3), 438–466 (2008)
26. Weiß, A., Andrikopoulos, V., Gómez Sáez, S., Karastoyanova, D., Vukojevic-Haupt, K.: Modeling Choreographies using the BPEL4Chor Designer: an Evaluation Based on Case Studies. Technical Report 2013/03, University of Stuttgart (2013)
27. Wombacher, A.: Alignment of Choreography Changes in BPEL Processes. In: SCC'09. pp. 1–8. IEEE (2009)