Enabling Reusable and Adaptive Modeling, Provisioning & Execution of BPEL Processes

Santiago Gómez Sáez, Vasilios Andrikopoulos, Michael Hahn, Dimka Karastoyanova, and Andreas Weiß
Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Stuttgart, Germany
Email: {gomez-saez, andrikopoulos, hahn, karastoyanova, andreas.weiss}@iaas.uni-stuttgart.de

***Abstract*—The Business Process Execution Language (BPEL) is a well established language for the definition of process models as service orchestrations. Service orchestrations are used in conjunction with service choreographies in order to create distributed, complex service-based applications. An important requirement for such applications is the need for flexibility during both their modeling and their execution. This work builds on this need by proposing an extension of BPEL in order to allow the definition of abstract constructs on the level of executable process models. Such constructs can be refined to concrete activities at any time, enabling the reuse of existing models and the dynamic adaptation to changing requirements. The design and implementation of the language extension, as well as that of the supporting environment required for the modeling, provisioning, and execution of such process models is further discussed. A case study on a city-wide public transportation system offers the means for an evaluation of the proposed approach.**

I. Introduction

The service oriented computing paradigm provides an ideal platform for the development and operation of complex distributed systems spanning multiple organizations. Different application areas for such systems usually focus on their domain-specific requirements, creating a fragmented landscape of service based systems. However, a closer examination of the requirements that each application area imposes shows many commonalities and opportunities for cross-domain solutions [1], [2]. Based on our experience with three of these application areas, in [1] we propose the concept of *Collaborative, Dynamic, and Complex (CDC)* systems. Participants of CDC systems are services, representing software systems of different granularity, virtual and physical devices, and individuals. Such participants join and leave the system at will in order to fulfill their individual goals. CDC systems are capable of adapting with respect to different triggers in the system and/or in their environment. Such systems have three fundamental aspects: *Modeling*, *Provisioning* and *Execution*. Service choreographies and service orchestrations (as implementations of the roles prescribed by each choreography) are used for modeling purposes. This creates the need for adaptive and context-aware provisioning mechanisms that enable the distributed execution of choreographies across potentially multiple organizations.

A central concept of CDC systems is flexibility, expressed as the ability to modify on demand predefined regions in the choreography and orchestration models by means of *abstract placeholders* in them [3]. Beyond the ability to dynamically adapt the execution of a model based on e.g. the context of its execution, such placeholders also enable the reusability of models by promoting the creation of generic and agnostic models to be refined and made concrete during provisioning or execution. In [3] we focused the discussion on the conceptual mechanisms that are required for enabling such abstract placeholders on the level of service choreographies and as an extension of the BPEL4Chor [4] choreography definition language. In this work we focus on orchestration models expressed as BPEL processes[1]. BPEL provides an inherent extensibility mechanism at both process and activity level, on which we build in our approach in order to allow the modeling, provisioning, and execution of flexible and configurable BPEL process models coordinated by flexible choreographies.

The contributions of this work can be summarized as:

1) the extension of the BPEL language with abstract placeholder constructs in executable process models that can be refined into concrete activities at different phases of the life cycle,
2) the design of an architecture for an execution environment that supports this extension and allows for the dynamic injection of process fragments for refinement purposes,
3) the implementation and proof-of-concept evaluation of the proposed architecture in the context of the ALLOW Ensembles EU project.

The rest of this paper is structured as follows: Section II provides the motivation behind this work. Requirements are subsequently derived in Section III, which are then used as the basis to create the conceptual foundations for the extension of BPEL in Section IV. The architecture and implementation of the execution environment is discussed in Section V. Section VI presents the case study evaluation. Finally, Section VII summarizes related works and Section VIII concludes with some future work.

II. Motivation

For purposes of further motivating our work we focus on *Collective Adaptive Systems* (CAS) as a type of CDC systems that we are particularly interested in. CAS comprise heterogeneous entities that collaborate towards achieving their own objectives, and the overall objective of the collective [5]. These entities can be either virtual or physical, and organizationally as well as geographically distributed. The interaction of such entities with the collective highly influences the behavior of the system. The objectives of individual entities may be in agreement or conflict with the behavior or decisions

[1]WS-BPEL 2.0 Specification: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

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 ALLOW Ensembles EU 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 [5].

The case of an Urban Mobility System as a combination of non-fixed route buses, taxis, and car-sharing options acts as a use case for the ALLOW Ensembles project. In this scenario, physical entities like passengers and drivers (of buses, taxis, or cars) have to collaborate with software entities like bus route planners and managers to allow the former to reach their destinations. Joining a bus route, for example, is reflected in the system as the respective passenger entity becoming part of the ensemble consisting of the bus driver, the passengers already on the bus, as well as the route manager software agent which keeps track of the progress of the route. The flexibility of the system stems from the fact that no separate processes have to be modeled for the passenger taking the bus or using a taxi. Instead, following the work on Adaptive Pervasive Flows (APF) [6], [7], each passenger acquires the necessary process logic for e.g. joining the bus route ensemble from the route manager and inserts it into his model. As a result, adaptability in this context is required on the level of both cells and ensembles and has to be supported by means of flexible orchestrations and the interactions among them in choreographies. The cells' behavior can be partially or fully 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.

III. Requirements Identification

Developing a system such as the one described in the previous section entails satisfying the following requirements:

R1. **Life Cycle**: reusability and adaptiveness of BPEL processes must be supported throughout the modeling, provisioning, and execution phases of the process life cycle.

R2. **Modeling Support & Graphical Notation**: towards easing the modeling of reusable and adaptive BPEL processes, the modeling environment must incorporate appropriate graphical artifacts, as well as the serialization mechanisms for generating executable BPEL processes.

R3. **Process Adaptation**: dynamic and customizable process adaptation mechanisms must be offered by the system. Moreover, the specification of concrete process adaptation and refinement points in the process modeling language must also be supported.

R4. **Manual and Dynamic refinement:** The refinement of placeholders with concrete activities must be possible both manually by modelers, and automatically by the underlying system, as required.

R5. **Dynamic Fault Handling & Compensation**: the definition of custom adaptation mechanisms must incorporate the dynamic handling of faults and the specification of their corresponding compensation actions due to failures occurred during the execution of adapted processes.

R6. **Partial specification of interactions**: as incoming and outgoing interactions may be partially specified in the business process model, the corresponding communication-related process constructs must allow the partial definition of communication details, to be finalized during the runtime phase.

R7. **Refinement phase definition:** The process adaptation point must be able to be annotated with information in which life cycle phase the refinement has to take place. The modeling and execution environment must interpret the annotated information accordingly.

R8. **Generality:** The concept of the adaptation points (placeholders) should be generic and not tailored to a specific application domain, but rather support its usage for different domain-specific requirements. It also should allow for different adaptation triggers.

IV. BPEL$^{\text{ReAd}}$

Towards fulfilling these requirements in the following we present our proposal for BPEL$^{\text{ReAd}}$, an extension of BPEL that supports the modeling, provisioning, and execution of reusable and adaptive processes.

A. Life-Cycle

Figure 1 depicts the life cycle of CDC systems as discussed in [2], focusing on the reusability and adaptive aspects for modeling, provisioning, and executing BPEL processes. The three major phases in the proposed life cycle are: *Modeling*, *Provisioning*, and *Execution* phases. The modeling, provision and execution can be performed in top-down and bottom-up way. The top-down approach starts with creating, provisioning, and enacting the complete or partial choreography model from which the orchestrations are derived. The bottom-up approach derives the complete or partial choreography model from orchestrations which have to be performed in a coordinated manner. The adaptation and in particular the refinement of choreographies and orchestrations can be performed in any of the life cycle phases.

With respect to the adaptation of orchestrations or processes, during the modeling phase, the control flow and message flow (of BPEL processes) are specified. Process model refinement tasks, i.e. adaptation during the modeling phase, can be performed by means of selecting reusable process fragments, which can replace the refinement points with concrete fragment logic. The deployment and instantiation operations in the provisioning phase use the BPEL process specification, their service interface, and deployment information for enacting and exposing the interfaces to access the deployed process in the process engine. The adaptation of partially specified deployable processes is supported in this phase by means of resolving the process refinement points with concrete process logic w.r.t. non-functional aspects, e.g. based on the service QoS characteristics. During the execution phase of the life cycle, the process engine is responsible for the creation of the process instances and the correlation of incoming and

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

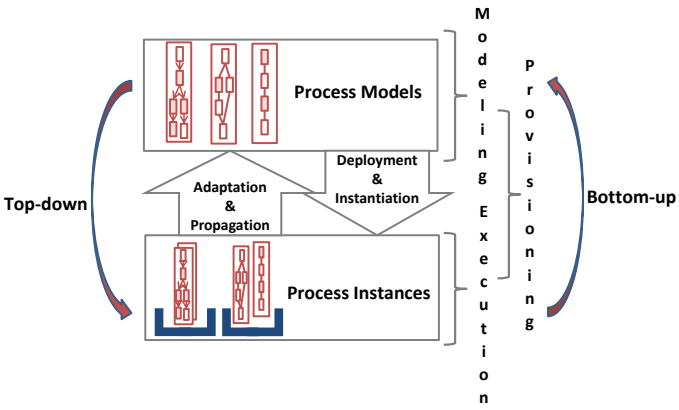

Figure 1. CDC System Life Cycle & Modeling Approaches Support [2].

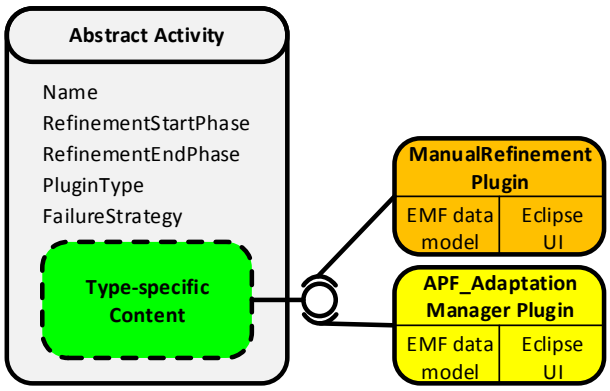

Figure 2. Abstract Activity Construct in BPEL — Overview

outgoing messages with the corresponding process instances. Process adaptation operations at this phase allow the manual or dynamic refinement of incomplete processes. A manual refinement requires pausing the instance and waiting for a human intervention. On the other hand, a dynamic refinement relies on the automatic retrieval of process fragments from an appropriate repository.

B. Abstract Constructs & Operations

As previously discussed, our approach builds upon the definition of refinement points which are subsequently adapted with concrete process logic within one of the phases of the life cycle. For this purpose, in the remainder of this section we present the artifacts that build the set of *Abstract Constructs* and the corresponding operations which build the basis of this work. The definition of abstract constructs for service choreographies was realized as part of previous research in the scope of building the necessary infrastructure capable of enabling the execution of complex interactions in a CDC system [2], [3]. As the existing artifacts for building such interaction models in such a system are coupled with the BPEL4Chor choreography language, the majority of the abstract construct artifacts defined in [3] are also applicable in the scope of this work.

More specifically, an *Abstract Activity* represents a generic placeholder comprising the following ingredients: (i) a name used as identifier, (ii) the earliest and latest life cycle phase in which its refinement must take place, (iii) the type of the mechanism to be used for the discovery of an appropriate fragment and subsequent refinement of the activity using this fragment, and (iv) a failure handling strategy for the refinement process. For example, a context-aware adaptation abstract activity requires the specification of context-aware *pre-conditions* and *effects* as attributes that specify the desired behavior of the logic the abstract activity represents without limiting it to predefined execution patterns. Pre-conditions specify in which state the context of the particular process has to be prior to the refinement of the abstract activity. The effects specify the desired state after the execution of the refined logic.

A *Fragment Discovery* operation realizes the mapping to the custom type-specific behavior content by means of a plugin mechanism, which is responsible for retrieving an appropriate fragment model. For instance, the fragment discovery function for the context-aware adaptation plugin type is responsible for discovering and selecting the most adequate fragment to satisfy the current execution context, and to infer the transition to a new context state. Failure strategies implemented as part of such behavior can be related to compensating and continuing the execution with a new discovered fragment, or by pausing the execution and waiting for manual human intervention.

C. Extensions to BPEL

We based the realization of the BPEL^{ReAd} language on the usage of the BPEL extensibility mechanism towards enabling the representation of the abstract activity as a part of an executable BPEL process. In this manner, we remain compliant to the BPEL standard and the existing technological frameworks already supporting the execution of BPEL processes. The syntax of the extension activity complies to the model presented in Fig. 2. The *Abstract Activity* syntax schema breaks down into two main definition sections: the *Generic Information* and the *Type-specific Content*. The generic information definition adheres to the main activity's ingredients: *Name*, *Refinement Start Phase*, *Refinement End Phase*, *Plugin Type*, and *Failure Strategy*. The plugin type definition refers to the plugin realizing the custom behavior realization. The *Type-specific Content* enables the insertion of the custom specific behavioral syntax for defining the necessary information for the fragment discovery operations. For example, in Fig. 2 two custom type specific contents are depicted: (i) manual refinement based on the retrieval of a fragment specified during design time, and (ii) dynamic refinement based on the discovery of context-aware process fragments during execution.

V. Architecture & Implementation

The architecture of the CDC execution environment supporting the BPEL^{ReAd} language, and targeting the reusable and adaptive modeling, provisioning, and execution of processes, is depicted in Fig. 3. Two interconnected environments comprise the architecture: a *Modeling Environment* and *Runtime Environment*.

A. Modeling Environment

The *Modeling Environment* consists of the *Process Modeler* and the *Fragment Repository* components (Fig. 3) and provides the necessary artifacts to create process models, persist and retrieve reusable fragments, manually refine abstract activities

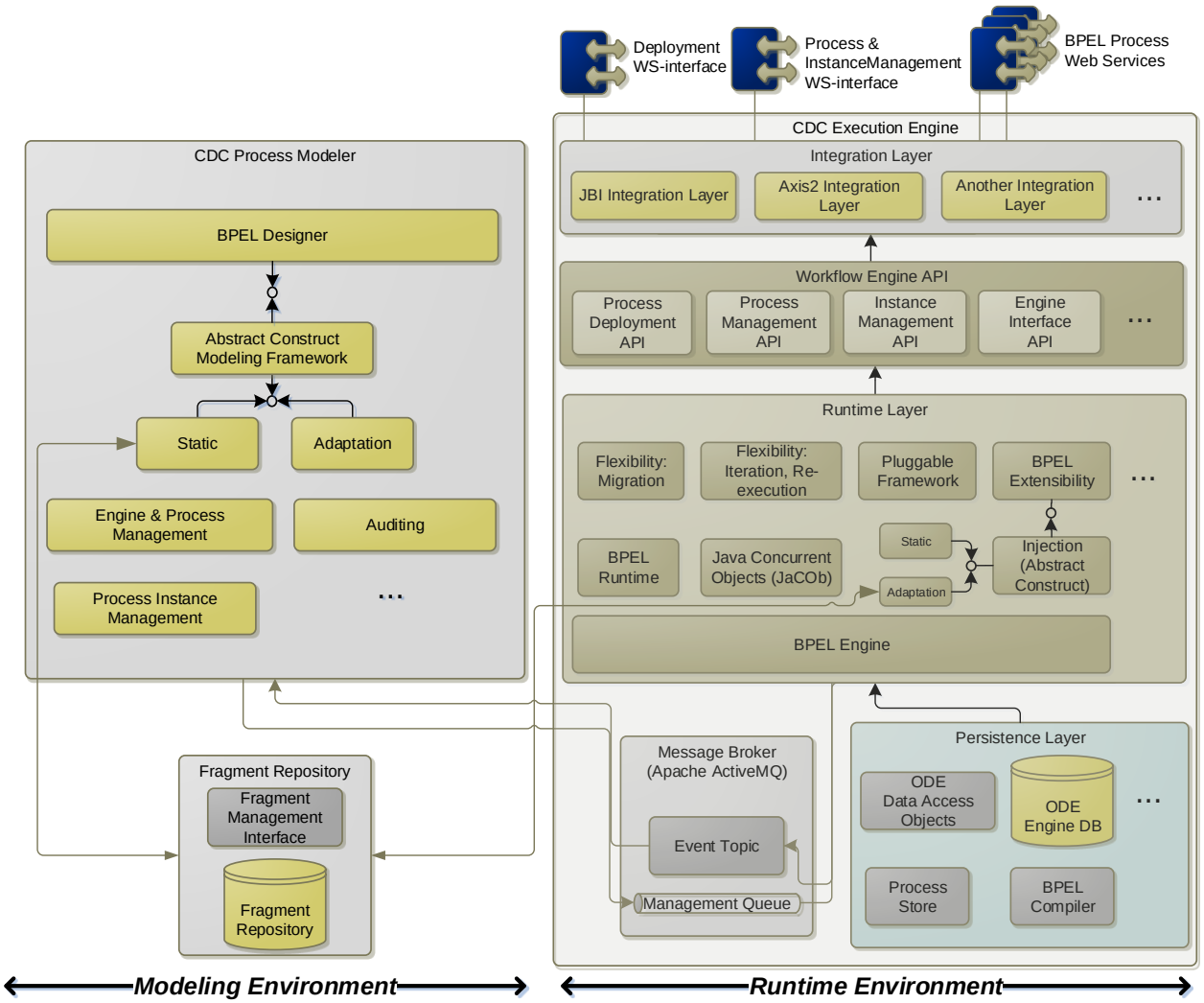

Figure 3. CDC Modeling & Runtime Environment

with retrieved fragments, and to interact with the runtime environment for administering and managing process models and their corresponding instances.

The *CDC Process Modeler* (as shown in Fig. 4) is based on the MayFlower designer [8], an extended version of the Eclipse BPEL designer[3]. Since the MayFlower editor is based on the BPEL designer, the extensibility points defined by BPEL are fully supported (see Fig. 3). The *Engine & Process Management* component enables the execution of process deployment and management operations on the *CDC Execution Engine*. Operations on the running process instances, e.g. pausing, resuming, or stopping a process instance, are supported through the *Process Instance Management* component. Real-time process information retrieved from the CDC Execution Engine is aggregated and visualized within the modeling environment through the *Auditing* component. For instance, when a runtime adaptation occurs, this component retrieves the changes and interacts with the BPEL Designer towards updating the visualization of the process model. Process and process instance management operations, as well as process instance control, are performed through a message-based interaction with the CDC Execution Engine. The *Fragment Repository* supports the storage and retrieval of process fragments which are uniquely identifiable and used for subsequent refinement operations.

Towards enabling the specification of process placeholders through the definition of abstract constructs, a pair of Eclipse Modeling (EMF) and User Interface (UI) plugins were developed. The EMF plugin contains the meta-model and operations allowing serialization to XML, while the UI plugin represents the layout and fields displayed to the user. There are two plugin types currently supported in the modeling framework: (i) the *Static Plugin* supporting the dynamic retrieval and injection of fragments manually defined during the modeling phase of an orchestration, and (ii) the *Adaptation Plugin* enabling the dynamic retrieval and injection of fragments discovered based on context-aware information, e.g. retrieved from an external *AI Planning Mechanism* (see Fig. 3). In order to support the reusability of process models among different scenarios, the CDC Process Modeler also allows the injection of fragments from a repository during the modeling phase of processes. For this purpose, it provides a process fragment repository

[3]Eclipse BPEL Designer: https://eclipse.org/bpel/

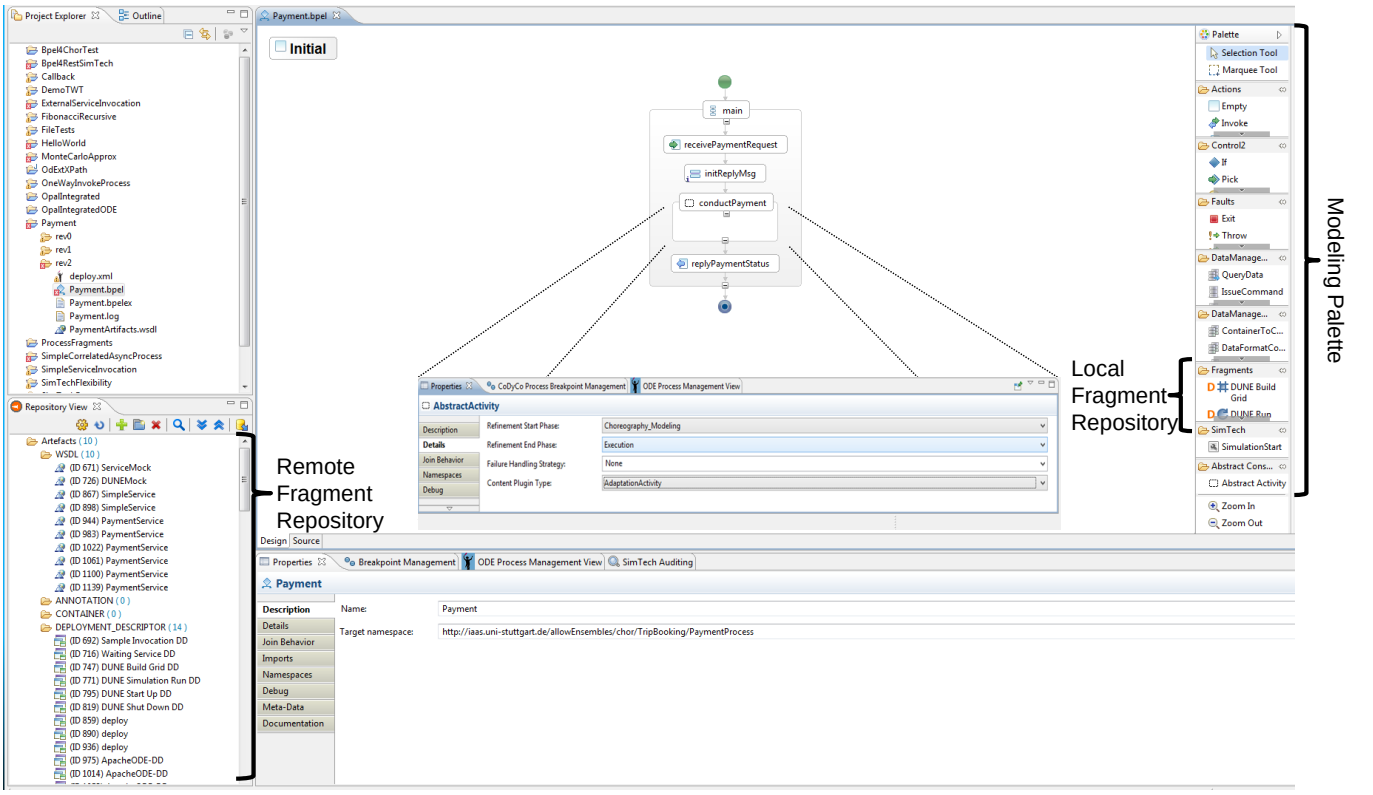

Figure 4. CDC Process Modeler — Overview

view which allows modelers to visualize existing reusable fragments which can be potentially refined during design time. The definition of an Abstract Activity (as summarized by Fig. 2) requires the specification of its properties and content plugin type in the properties tab of the IDE, as depicted in Fig. 4. When the content plugin type is selected, the view automatically changes depending on the behavior realized by the plugin. For the *Adaptation* plugin content type, the view changes according to the plugin content-specific properties.

B. Runtime Environment

The *Runtime Environment* comprises the necessary components to enable the execution of partially (by virtue of inclusion of abstract activities) and completely specified processes. The processes are deployed on one or more *CDC Execution Engines* and can be instantiated at any time. Since the deployed processes may be only partially defined, i.e. they may contain abstract activities which need to be refined during execution, the *CDC Execution Engine* must be able to start the execution of incomplete processes, allowing the dynamic injection of additional activities that are discovered from the fragment repository or triggered through the process modeler. The open source BPEL Engine Apache ODE[4] as extended in [8] is used as the basis for the implementation of the execution engine. In the remaining of this section we provide an overview of the different layers and their components, including the extension mechanisms which are developed as part of our work.

Figure 3 depicts the architecture of the CDC Execution Engine, consisting of three main layers: *Integration Layer*, *Runtime Layer*, and the *Persistence Layer*. The *Integration Layer* exposes the engine-internal functionality to the outside, and provides the communication features required by deployed BPEL processes. Administration and management functionalities of BPEL process models, e.g. deploy and undeploy operations, are provided through the *Deployment* interface. BPEL processes and their running instances can be managed through the *Process & Instance Management* interface, e.g. pausing an instance's execution, re-executing a fragment, etc. The *Axis2* and *JBI Integration Layer* implementations are provided by default with the engine. However, the extensibility features of the engine enable the realization of further integration layer implementations that allow to integrate the engine with other systems. The *Workflow Engine API* comprises a set of APIs for engine-internal operations. These APIs are used as an intermediate layer between the *Integration Layer* and the engine-internal functionalities grouped in the *Runtime Layer*.

The *Runtime Layer* provides the core functionality of the CDC execution engine. The *BPEL Engine* component comprises all the engine-related classes, such as the ones for representing a BPEL process model or the runtime data of a process instance (e.g. variable values). The *BPEL Runtime Component* contains the implementations of all BPEL constructs which are internally interpreted by the engine to instantiate and execute a process model. Towards ensuring reliability during the process execution, the BPEL runtime relies on the capabilities of the *Java Concurrent Objects (JaCOb)* framework and the persistence functionalities of the engine's database in the *Persistence Layer*. JaCOb provides an application-level concurrency mechanism, and a mechanism for interrupting the execution and persisting the state of

[4]Apache ODE: http://ode.apache.org/

running process instances in the engine's database. In case of a failure in the engine, once it resumes its running state again, the runtime data of all interrupted process instances are restored and their execution is resumed. The *Pluggable Framework* realizes a generic BPEL 2.0 Event Model towards propagating engine-internal events for process debugging purposes [9]. Such propagation is achieved through the usage of *Message Topics* (depicted as *Event Topic* in Fig. 3). Subscribers to the topic can influence or debug the execution of process instances by sending management messages to the *Management Queue*, e.g. to set process execution breakpoints or for setting variables values [8]. The *Flexibility* component realizes the support for iterating or re-executing parts of a running process instance, i.e. re-starting the execution of a set of activities or compensating before re-starting the execution of a set of activities.

The *BPEL Extensibility* component realizes the extensibility points defined by the BPEL specification. For purposes of enabling the execution of cells which are partially defined during the modeling phase and resolved during execution, a pluggable *Injection* component was developed as the means to realize the *Injection Framework* capable of injecting retrieved fragments in the *Runtime Environment*. Since the discovery of the fragments requires the specification of different properties and interaction with different external sources, domain specific plugins must be realized and dynamically loaded in the *Injection* framework. We already realized a *Static* plugin that retrieves a fragment that is specified during the modeling phase from a repository for injection during runtime. The *Adaptation* plugin incorporates context and preferences analysis during the refinement step, which then dynamically influences the selection of the discovered fragments. The data sharing and exchange between the running process instance and the injected process fragment is supported in our approach by the definition and sharing of global variables. In future work, we plan to introduce a more complex definition and resolution mechanism between the process instance and the injected fragment variables.

The *Persistence Layer* provides the execution engine with the database to interpret and persist all execution related information, e.g. deployed process models, running instances information, history of the execution, etc. The persistence is performed through the *ODE Data Access Objects* and the *ODE Engine DB* components. The *ODE Data Access Objects* are intermediate mediation components between the *Runtime Layer* and the engine's *Database*. The *BPEL Compiler* interprets the BPEL, WSDL, etc. files comprised in a deployment bundle and converts them into an engine-internal representation suitable for execution purposes. The *Process Store* handles the deployment of new process models and triggers their compilation.

The CDC Modeling & Runtime Environment is developed as an open source toolset and is meant to be publicly available in the short term future.

VI. Evaluation

The evaluation of the presented approach is performed by means of driving a case study scoped in the domain of CAS systems discussed in Section II, and more specifically, in the scope of the *Urban Mobility System* scenario in the ALLOW Ensembles project. Requesting a trip in the system requires the interaction of multiple entities, e.g. the *Passenger*, *Trip Booking System*, *Transportation Mean Systems*, *Payment Gateway*, etc. Since in this work we focus entirely on the adaptation and reusability of each entities' behavior expressed as processes, we focus on one concrete entity which requires high adaptation and reusability functionalities: the *Payment Gateway*. The Payment Gateway deals with the various possible payment methods, and the different preferences and constraints of different users for using them. For this purpose, it is necessary to abstract the definition of the payment process by partially specifying it during the design phase. For instance, one concrete passenger may require to pay using a credit card method, while another one may require, due to security concerns, to pay using a direct wire transfer service. The protocol, i.e. the orchestration logic, required for interacting with different back-end services is different for each service, and requires a high degree of freedom during the process design time and adaptability during the process runtime.

For such a case study scenario we partially specified the Payment Process depicted in Fig. 4. The abstract activity conductPayment represents the refinement point to be refined during runtime depending on the users' payment preferences. Figure 5 depicts the execution and adaptation of the payment process. In this scenario, the user's preferred payment method is the MasterCard credit card. Thus, when the execution engine schedules the execution of the conductPayment activity, the preferences are first analyzed, and subsequently the MasterCard payment fragment is retrieved from the repository. The engine then schedules the execution of the fetched fragment and executes it within the running process instance. Modifications in the process model are propagated to the modeling tool, which also indicates in real time the execution state of each process activity (see Fig. 5). A video demonstrating the dynamic selection and injection of the fragment during the execution of the process is also available[5].

VII. Related Work

Existing work on enabling adaptability of large scale collaborations is targeted in the CHOReOS EU Project[6], by focusing on the substitution of services towards satisfying evolving functional or non-functional requirements. [10] targets the adaptability during runtime based on a policy-aware *finding and binding* of services and service types through modeling constructs.

Other current investigations rely on 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 cases where the activities or process fragments are newly specified inside a placeholder during run time, and *late selection of process fragments* in cases where 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 [11]. There are several approaches realizing these generic patterns. In [12], the notion of worklets, which

[5]Process Runtime Refinement: http://www.iaas.uni-stuttgart.de/BPELReAd/PaymentFragmentInjection.html

[6]CHOREoS EU Project: http://www.choreos.eu/

Figure 5. Payment Process Dynamic Adaptation — Execution

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 [13], the concept of Pockets of Flexibility (PoF) is introduced. PoF contain a set of activities, with the control flow between activities to be 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 [14], while in [15] in the field of pervasive flows planning based techniques are used to generate a sub-process. The concept of Adaptable Pervasive Flows (APF) [6], [7] also uses the concept of abstract activities which are refined depending on the context of the entities represented by a workflow.

While these approaches are similar to our work in terms of the underlying concepts to provide flexibility during modeling and execution of workflows, they are not based on a well-known and industry-supported standard such as BPEL, but rely on proprietary workflow technologies. Furthermore, they do not offer an open source based end-to-end tool-suite realizing the complete life process life cycle, i.e. including a process modeling tool allowing the specification of abstract activities which can then be deployed and provisioned in an open source BPEL engine that has been extended with process fragment injection capabilities. Murguzur et al. [16] propose to insert process fragments into a base model during run time using an staged approach for fragment resolution depending on context information. This approach also builds on a standard-based, open-source tool set consisting of BPMN 2.0 and the Activiti engine. However, the presented extension to the BPMN engine remains on a high level in terms of its architecture and implementation.

Adaptation of workflows is not limited to changes in predefined regions. Song et al. [17] propose an approach that allows the migration of instance state from a source process schema to a changed target process schema while ensuring the correctness of the adaptation. In [18], service-based process instances are adapted as a reaction to context-changes by transforming the adaptation to a planning problem and building state transition systems. The state transition system fulfilling a set of goals is then translated back into an executable process and the execution is resumed. Aspect-Oriented Programming (AOP) is used in [19] to enable the run time adaptation of processes. Changes are described as aspects and dynamically woven into BPEL process instances by a corresponding execution engine. A similar, but more generic AOP-like approach is described in [20]. WS-Policy is used to specify aspects representing additional functionality to be added into a process, while the communication between engine and the aspects is managed by a publish/subscribe system. A major difference to our approach is that in these approaches the logic to be included is identified and specified (manually) by the user during either the process modeling or execution phases, and do not rely on explicitly defined abstract activity constructs. Additionally, these works are missing an integrated, end-to-end tool chain spanning from a modeling environment to the execution engine where instance level changes are directly visualized in the modeling tool for the respective user.

VIII. Conclusion and Future Work

Our previous work on Collaborative, Dynamic, and Complex (CDC) systems provides the means for a unified view on complex service based applications spanning multiple organizations [1], [2], expressed as service choreographies and orchestrations. The introduction of abstract placeholders on the level of choreographies allows for flexibility in system modeling level, offering also the possibility for reusability across

choreography models. In this work we focus on applying this concept to the level of service orchestrations, using BPEL, a well established and supported process definition language for this purpose. The presented approach provides a modeling and execution environment capable of supporting the deployment and execution of reusable and adaptive processes expressed as service orchestrations. For this purpose, we first derive a set of requirements for such an approach. We then propose the usage of abstract constructs in executable process models in order to specify process fragment placeholders, which can then be refined during the modeling, provisioning, or execution phases of the process life cycle. Subsequently, we discuss our proposal for an appropriate architecture and its implementation based on extending existing technologies used widely in both research and industry domains. The evaluation of the approach is performed by means of a case study in a European Union-funded project.

Future investigations are focused on providing full support of the CDC systems life cycle by means of a set of custom abstract construct realizations that can then be used by CDC modelers, potentially supported as a plugin repository. For example, such constructs allow focusing on QoS aspects for the provisioning and allocation of processes. Moreover, we plan to extend the case study evaluation by approaching further domains, e.g. simulation workflow in eScience.

ACKNOWLEDGMENTS

This research is partially founded by the EU FP7 600792 project ALLOW Ensembles and the German Research Foundation (DFG) within the Cluster of Excellence in Simulation Technology (EXC310).

REFERENCES

[1] V. Andrikopoulos, S. Gómez Sáez, D. Karastoyanova, and A. Weiß, "Towards Collaborative, Dynamic & Complex Systems," in *Proceedings SOCA'13*. IEEE Computer Society, 2013, pp. 241–245.

[2] ——, "Collaborative, Dynamic & Complex Systems: Modeling, Provision & Execution," in *Proceedings of CLOSER'14*. SciTePress, 2014, pp. 276–286.

[3] A. Weiß, S. Gómez Sáez, M. Hahn, and D. Karastoyanova, "Approach and Refinement Strategies for Flexible Choreography Enactment," in *Proceedings of CoopIS'14*, H. P. et al. R. Meersman, Ed. Springer Berlin Heidelberg, October 2014, Conference Paper, pp. 93–111.

[4] G. Decker, O. Kopp, F. Leymann, and M. Weske, "BPEL4Chor: Extending BPEL for modeling choreographies," in *Proceedings of ICWS'07*. IEEE, 2007, pp. 296–303.

[5] V. Andrikopoulos, A. Bucchiarone, S. Gómez Sáez, D. Karastoyanova, and C. A. Mezzina, "Towards Modeling and Execution of Collective Adaptive Systems," in *Proceedings of WESOA'13*. Springer, 2013, Workshop-Beitrag, pp. 1–12.

[6] A. Bucchiarone, A. L. Lafuente, A. Marconi, and M. Pistore, "A formalisation of adaptable pervasive flows," in *WS-FM'09*. Springer, 2009, pp. 61–75.

[7] K. Herrmann, K. Rothermel, G. Kortuem, and N. Dulay, "Adaptable Pervasive Flows - An Emerging Technology for Pervasive Adaptation," in *Proceedings of SASOW'08*. IEEE, 2008, pp. 108–113.

[8] M. Sonntag and D. Karastoyanova, "Model-as-you-go: an approach for an advanced infrastructure for scientific workflows," *Journal of Grid Computing*, vol. 11, no. 3, pp. 553–583, 2013.

[9] O. Kopp, S. Henke, D. Karastoyanova, R. Khalaf, F. Leymann, M. Sonntag, T. Steinmetz, T. Unger, and B. Wetzstein, "An event model for ws-bpel 2.0," 2011.

[10] D. Karastoyanova, F. Leymann, J. Nitzsche, B. Wetzstein, and D. Wutke, "Parameterized bpel processes: concepts and implementation," in *Proceedings of the 4th international conference on Business Process Management*. Springer-Verlag, 2006, pp. 471–476.

[11] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change Patterns and Change Support Features - Enhancing Flexibility in Process-aware Information Systems," *Data Knowl. Eng.*, vol. 66, no. 3, pp. 438–466, 2008.

[12] M. Adams, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst, "Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows," in *OTM Conferences (1)*. Springer.

[13] S. Sadiq, W. Sadiq, and M. Orlowska, "Pockets of Flexibility in Workflow Specification," in *Conceptual Modeling — ER'01*. Springer, 2001, pp. 513–526.

[14] L. Ardissono, R. Furnari, A. Goy, G. Petrone, and M. Segnan, "A framework for the management of context-aware workflow systems," in *WEBIST (1)*, 2007, pp. 80–87.

[15] A. Bucchiarone, A. Marconi, M. Pistore, and H. Raik, "Dynamic Adaptation of Fragment-Based and Context-Aware Business Processes," in *Proceedings of ICWS'12*. IEEE, 2012.

[16] A. Murguzur, X. De Carlos, S. Trujillo, and G. Sagardui, "Context-Aware Staged Configuration of Process Variants@Runtime," in *Advanced Information Systems Engineering*. Springer International Publishing, 2014, vol. 8484, pp. 241–255.

[17] W. Song, X. Ma, W. Dou, and J. Lu, "Toward a model-based approach to dynamic adaptation of composite services," in *Web Services, 2008. ICWS '08. IEEE International Conference on*, Sept 2008, pp. 561–568.

[18] A. Bucchiarone, M. Pistore, H. Raik, and R. Kazhamiakin, "Adaptation of service-based business processes by context-aware replanning," in *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on*, Dec 2011, pp. 1–8.

[19] C. Courbis and A. Finkelstein, "Weaving aspects into Web service orchestrations," in *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, July 2005, pp. 219–226.

[20] D. Karastoyanova and F. Leymann, "BPEL'n'Aspects: Adapting Service Orchestration Logic," in *Proceedings of 7th International Conference on Web Services (ICWS 2009)*. IEEE Computer Society, 2009, pp. 222 – 229.

All links were last followed on July 12, 2015.