Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Diploma Thesis No. 3564

# Dynamic Process Fragment Injection in a Service Orchestration Engine

Lukasz Bialy

**Course of Study:** Software Engineering

**Examiner:** Jun.-Prof. Dr.-Ing. Dimka Karastoyanova
**Supervisor:** Dipl.-Inf. Santiago Gómez Sáez
M.Sc. Wirt.-Inf. Andreas Weiß
**Commenced:** September 17, 2013
**Completed:** March 19, 2014

**CR-Classification:** C.2.4, D.2.1, D.2.2, D.2.6

## Abstract

The EU Project Allow Ensembles aims to develop a new design principle for large-scale collective systems (CAS) based on the concepts of cells and ensembles, where cells represent a concrete functionality in a system, and the ensembles are collections of cells which collaborate in order to fulfill a certain goal in a given context. Adaptive Pervasive Flows (APF) are based on workflow technology and utilized in pervasive environments to model the cell's behavior. During runtime, APF instances must be able to adapt their behavior, e.g. due to failures or changes in their environment, in order to be able to fulfill a certain goal. For this purpose, APFs may contain a particular type of activities, known as abstract activities [BLMP10]. Abstract activities partially specify the flow behavior, which must then be resolved during runtime into one or multiple concrete activities, which in turn must be injected into the running flow. In the scope of this thesis, APFs are specified using an extension of the WS-BPEL language [OAS], [vLLN11]. The WS-BPEL language, provide the necessary mechanisms for extending the language for modeling custom process activities and specifying their behavior.

In this thesis we focus on the WS-BPEL language and on an extended version of the Apache ODE orchestration engine [Apaa], for business process modeling and execution respectively. With respect to the injection of pervasive process fragments, which contain one or multiple activities and properties, the language and execution engine requirements and constraints are investigated. For this purpose, a State-of-the-Art analysis on generic process fragment injection approaches is first driven. Once the constraints are detected, we present the formalization of the required language extension based on WS-BPEL language, and conduct a specification of requirements and architectural design of the final prototype based on the Apache Orchestration Director Engine (Apache ODE). For integration purposes in the overall required execution environment, the Enterprise Service Bus Apache ServiceMix 4.3 is used.

# Contents

# List of Figures

# List of Tables

# List of Listings

# 1. Introduction

In this chapter the motivation behind this work is presented and the scope of the thesis is specified. Further the structure of the diploma thesis is outlined and the definitions used are explained.

## 1.1. Motivation Scenario

The employment of workflow models has proven itself to be very beneficially, so its natural that the usage of software based business process support is very popular in the corporate world. Companies, to provide high service standards and to work in a effective and efficient manner, have to react on changes in how to perform business and adapt their applications soon as possible. Pervasive computing[Sat01, Wei01] is a emerging paradigm that gains more importance in the modern computer world. In this paradigm numerous smaller computing devises interact in a almost invisible manner in a digital environment to meet human needs. These devises are frequently mobile or are embedded in the environment and are connected to an ubiquitous network structure. In the scope of the EU Project Allow Ensembles ways are researched to exploit the advantages of both: business process models and pervasive computing.

The goal of Allow Ensembles is to develop a new design principle for collective adaptive systems (CAS) based on the concept of cell ensembles and to develop a new foundational framework for it. Cells represent a concrete functionality and are unique building blocks in a multi-cellular system. To fulfil a task or goal entities (physical or virtual organizational units) have to interact with each other. To be able to do so, they expose a set of cells. Such a collection of collaborating entities to is called a ensemble. The realization of a functionality is implemented by multiple cells interacting with each other through pre-defined protocols. Here the process modelling comes into play, more precisely the Adaptive Pervasive Flows(APF's) which are based on workflow technology and are used to specify the behaviour of cells. Also cells expose their functionality for other cells in form of APF-fragments.

During runtime, APF instances must be able to react almost instantly to dynamic changes occurring in their environment and adapt their behaviour accordingly in order to fulfil its goal. Among the activities an adaptive pervasive flow can contain there is one particular, that has an special meaning and basically represents the goal of this thesis. It's the abstract activity, which marks where a other cell or composition of cells is required to implement a functionality. It only partially specifies its behaviour in terms of a goal to be reached. A cell, in order to be able to continue its process and complete its task, have to replace the abstract activity with the process fragment provided by other cells. On other words the process

fragment must be injected in the adaptive pervasive flow to create a complete fully specified process to execute. The selection of which cell, more precisely which process fragment to use, can be done ether during design time or at run time. In [BMPR12] are two advantages mentioned of the later option. The first is the fact that often not all available fragments are known at design time. The second advantage is that the composition strongly depends on the current execution situation. In dynamic pervasive environments its very likely, that changes occur between design and execution time.

The goal of this thesis is to enable the injection functionality described above. Therefore we develop concepts and implementation artifacts. In the scope of this thesis, APFs are specified using an extension of the WS-BPEL language [OAS], [vLLN11]. We use the extension mechanisms that the WS-BPEL language provide to add a custom process activity (the abstract activity) so it can be used for modelling our pervasive adaptive flows. To enable injection of pervasive process fragments(composed of one or multiple activities and properties) in business processes instances requires extending the language and execution engine. For the prototype implementation we are using an extended version of the Apache ODE orchestration engine[Apaa]. It will be further extended to be able to run our APFs and provide the injection feature.

### 1.1.1. Smart City Scenario

To make the explanations in this work less abstract and improve it understandability we show you a real life example where the concepts described in this thesis are used. In this section a motivation scenario is presented based on the one from [ABS+13].

The FlexiBus company provides transportation services in a very flexible manner based on the current customer needs. The public transportation system (e.g., buses, metro, trains) typically have fixed schedules, drive only the regular routes and service only specified stations. This is not always sufficient in an urban environment with a large total population, as usually on the outskirts of the city the public transportation runs not as frequent as in the centre. FlexiBus meets the citizens mobility needs in a different way. The company provides multiple special buses which service a flexible route depending on the passengers needs. The customers can pre-book a pick-up point along the route. The bus drives to a pre-defined destination.

The goal of this scenario it to develop a FlexiBus Management System (FBMS) to support the management and operation of FlexiBuses. IT is essential that the actors (i.e., passengers, buses, routes, FlexiBus operations manager, bus assistance manager, traffic manager, roads manager etc.) in this system cooperate in a synergistic manner to fulfil their own and collective procedures and goals.

As shown in the figure above, such a system must be capable to manage simultaneously multiple routes (e.g. blue and red routes in the figure) set by passengers by pre-booking of pick up points. The functionalities such a system provides, can be summarizes as follows: Each *Passenger* can request a trip to one of the predefined destinations in the system, asking to start at a certain time and from a preferred pick-up point. Special requests form passages also are supported, for example disability related requirements or the travelling with extra sized


**Figure 1.1.:** The FlexiBus Scenario[ABS+13]

luggage. The system provides multiple ways to pay for the trip. It supports payment with cash, a credit card or a monthly pass. The customers can also pay remotely via the FlexiBus company web site. Also a useful feature of the system is the passenger notification about problem (e.g. accidents, bus delays) on the selected route.

A *Bus Driver* has a precise route with a list of passengers assigned to it and a unique final destination. If a bus has enough available seats, passengers without booking can be accepted under the premise, that there are along the calculated road. Each bus driver has a *Route Manager* assigned which provides information about next pick-up points and passengers check-ins. A *Route Planner* creates the different routes in such a way, that all passenger requirements are meet( i.e. pick-up points, destination, arrival time) and bus costs are optimized (i.e. shortest distance, avoiding wasteful gasoline usage, avoiding of traffic jams, etc.). The *Flexibus Manager* provides all necessary information about traffic, closed roads, available buses etc. required by the route manager to calculate the possible routes. There is also a *Bus Assistance Service* for which allows bus drivers to report problems that occur on the way along the road and request for advice other steps to help (e.g. call ambulance and notify police for an accident, pick-up a bus for repair). The last is the *Payment Service* entity that handles the ticket purchases and provides interfaces for various payment systems, to support different payment methods.

The system needs to deal with the dynamic nature of the scenario. It must be able to handle the variability of the actors involved and of their goals, it also has to react to the exogenous context changes, e.g. bus damages, passenger requests cancellations, traffic jams, etc. that can effect its operation. Some of the tasks executed by the actors might require customization, to adopt to the new environmental situations(e.g. payment with credit card instead of cash). Such a system, to perform in a efficient manner, must be capable to make changes in the procedures of some actors participating in the system (i.e. route changes). Such adaptable processes are described in section 2.2.

## 1.2. Scope of Work

In the scope of this thesis the dynamic process fragment injection in the service orchestration engine Apache ODE is enabled. The focus lays on an extended version of the business process engine. To provide this capability the WS-BPEL 2.0 specification will be extended with an additional activity type. A formalization of the language extension will be provided. An overview of the current state-of-the-art on generic process fragment injection approaches will be presented. The language and execution engine requirements and constraints are investigated, with respect to the injection of pervasive process fragments. A specification of requirements and a architectural design of the final prototype based on the Apache Orchestration Director Engine (Apache ODE) is created. For process fragment resolution and retrieval an external service will be used. Further the prototypical implementation of the concept is realized and integrated in the Apache ServiceMix 4.3 Enterprise Service Bus. Finally the implemented approach will be validated and evaluated. In this section we provide an overview of the scope of this work.

As mentioned in the previous section the goal of this work is to enable dynamic process fragment injection in a service orchestration engine. In this thesis we focus on the WS-BPEL language for business process modeling and on an extended version of the Apache ODE orchestration engine [Apaa] for execution respectively. We mention that WS-BPEL 2.0 is used for adaptable pervasive flows. Usually these flows are used together with a context model describing the current state of the executional environment like in [HRKD08, BLMP10, MPS+09]. The realization of such a context model and the interaction between process and the model goes beyond the scope of this work, as we focus on the injection of process fragment in a running process instance. Also implementation of adaptation approaches based on a context model are out of scope. The results of this thesis can be used in a future project to implement all functionalities for APF's.

For both, decision making regarding fragment selection and the retrieval of those process fragments, external services will be used. For the purpose of this work, interfaces for these services will be provided. The interfaces will be created based on the communication standards used in Apache ServiceMix 4.3 so that the interaction with the services will be realized through the enterprise service bus implementation contained in ServiceMix. The implementation of those services is created in to different projects outside the scope of this thesis. The service providing the process fragments is already realized in the Fragmento[DS] project. The service responsible for the decisions regarding composition of process fragments has not been implemented yet. For the purpose of this work we will use a dummy service returning random choices for the process fragments. process fragmen injection in a op In the scope of this work we With respect to the injection of pervasive process fragments, which contain one or multiple activities and properties, the language and execution engine requirements and constraints are investigated. For this purpose, a State-of-the-Art analysis on generic process fragment injection approaches is first driven. Once the constraints are detected, we present the formalization of the required language extension based on WS-BPEL language, and conduct a specification of requirements and architectural design of the final prototype based on the Apache Orchestration Director Engine (Apache ODE). For integration purposes

in the overall required execution environment, the Enterprise Service Bus Apache ServiceMix 4.3 is used.

Its important to mention, that composition of multiple process fragments for the injection is also beyond the scope of this work. If multiple process fragments should be injected it the process instance, they have to be provided already composed as one process fragment. This fits better in the concept of adaptation strategies described in 2.2.

The simultaneous execution of processes is handled in Apache ODE with the JaCOb (Java Concurrent Objects) framework[Foub]. The design and implementation of different optimization strategies for the simultaneous execution of refined activities in the flow activity are not considered in the scope of this thesis. But it will be ensured, that the process fragment will not interfere with the execution of other processes (isolation).

## 1.3. Outline

This diploma thesis contains the following chapters:

**Chapter 1 - Introduction** describes the background and motivation for the topic of this diploma thesis and provides a description of the scope of this thesis.

**Chapter 2 - Fundamentals** provides descriptions of the necessary concepts and technologies used in this thesis.

**Chapter 3 - Related Works** provides an overview on existing approaches for enabling process fragment injection (in general).

**Chapter 4 - Requirements and Concepts** provides the requirements analysis and identified possible constraints in the language and execution environment. Furthermore an overview of the framework enabling process fragment injection and the formalisation of the BPEL language extension is provided. Also the data model of the introduced abstract activity is described based on an example.

**Chapter 5 - Implementation** describes the architectural solution and the realisation of the Modelling Environment and Execution Engine enabling process fragment injection.

**Chapter 6 - Outcome and Future Work** provides a summary of the outcomes of this diploma thesis and describes possible future use and extensions for the resulting system.

## 1.4. Definitions and Conventions

In this section the definition and convention used in this thesis are explained.

### Definitions

In this document the general term BPEL refers to the Web Services Business Process Execution Language (WS-BPEL) 2.0 specification[OAS].

The term operational environment refers to the real world environment where the process is executed. This environment hast entities, which can interact with each other and can have different states.

The term refinement refers to the dynamic selection of an appropriate process fragment and replacement of the refined abstract activity with the business logic from this fragment.

Dynamic selection and execution of business logic during runtime is referred to as process fragment injection.

The term parent process refers to the process or process fragment in relation to a injected process fragment. The parent process is the one which previously contained the abstract activity which was replaced by the mentioned process fragment.

The term main process refers to the initial process which was started by the engine and not as result of an injection.

### Acronyms

| Acronym | Explanation |
|---|---|
| APF | Adaptable Pervasive Flow |
| BPEL | Business Process Execution Language |
| DOM | Document Object Model |
| EMF | Eclipse Modelling Framework |
| EPR | Endpoint Reference |
| ESB | Enterprise Service Bus |
| GEF | Graphical Editing Framework |
| HTTP | Hypertext Transfer Protocol |
| IP | Internet Protocol |
| JaCOb | Java Concurrent Objects |
| JBI | Java Business Integration |
| JMS | Java Message Service |
| (Apache) ODE | (Apache) Orchestration Director Engine |
| OSGi | Open Services Gateway initiative |
| SBSs | Service-Based Systems |
| SOA | Service-Oriented Architecture |
| SOAP | Simple Object Access Protocol |
| TCP | Transmission Control Protocol |
| UDDI | Universal Description, Discovery and Integration |
| XML | Extensible Markup Language |

# 2. Fundamentals

In this chapter an overview and an introduction is provided for the concepts and technologies used in this thesis. To provide the context for the work of the diploma thesis, the fundamental concepts are described. Further the technologies and prototypes are introduced, that the thesis builds on.

## 2.1. Service Oriented Architecture

As interconnected IT-infrastructures became an essential part of today's enterprise reality, the importance of seamless cooperation through digital communication networks increased significantly. To satisfy the high requirements on distributed computer systems used for business purposes and to realize business processes, different approaches for system architectures were created. A widely used architectural paradigm used for modern enterprises IT-infrastructures is the *Service-oriented Architecture* (SOA) paradigm [WCL+05].

The Open Group defines *Service-Oriented Architecture (SOA)* as "an architectural style that supports *service-orientation*" [Gro]. Further Service-orientation is described as "a way of thinking in terms of services and service-based development and the outcomes of services" [Gro]. In this context a service is a self-contained logical representation of a business operation with a specified outcome. Implementation details of a service are hidden, but its possible that a service may be composed of other services[Gro]. SOA relies on loose coupling between services, standardization and interoperability to provide the necessary flexibility to build large distributed systems[WCL+05].

The core concept of Service-Oriented Architecture is the pattern of service publication, finding, binding and invocation. This patter describes the interaction between the three main roles defined in SOA: service provider, service requester and service broker. Following the pattern, first the provider has to register(publish) a description of his service to the service broker. The service requester can find a suitable service by sending a search request to the broker, that returns a concrete endpoint of a service matching the search criteria. The service requester can then bind to the concrete service and invoke the business operation[WCL+05].

### 2.1.1. Web Services

Enterprise business processes use services as their basic building blocks. For effective orchestration of services it is required to provide sufficient business descriptions of the services and enable access through standardized open protocols [Gro]. For this purpose a set of layered

and interrelated technologies was developed, which allow to realize SOA in an platform/vendor neutral manner[Gro04]. Figure 2.1 presents an overview of these technologies and show how there are build on each other.

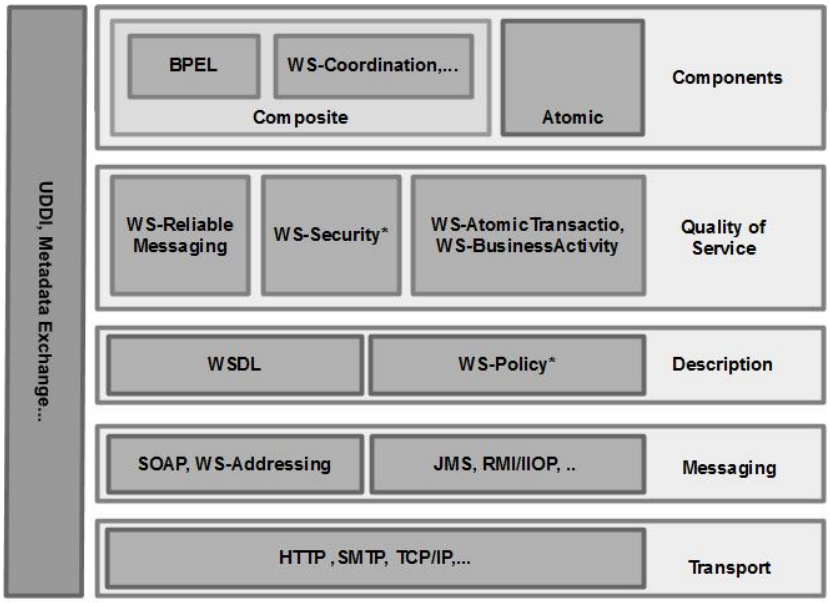


**Figure 2.1.:** Web services architecture stack (based on the diagram presented in [WCL+05]

The base of the web service architecture are the standard transport protocols, most commonly used are HTTP[1] together with TCP/IP[2]. With the protocols form the transport layer messages are exchanged, that are used for the interaction between services. The messaging protocols can be divided in those based on XML[3] and other messaging protocols (e.g. JMS[4]). SOAP[W3C07] is the standard XML-based messaging protocol used for web services. The next layer is the description layer. To describe web services usually a WSDL document is used. WSDL stands for Web Services Description Language[5] and is a XML-based standard for description. Further the web service technology stack contains multiple extensions to provide the means to support quality of service. These are part of quality of service layer. One example is the WS-ReliableMessaging[6] specification allowing to transfer messages between nodes implementing this protocol independently of system or network failures. The upper layer covers technologies used for service aggregation, coordination and orchestration. The best example for such an technology is BPEL[OAS], which is described in the next section. For a SOA implementation, service discovery and composition is needed. The Universal Description, Discovery and Integration (UDDI)[7] specification covers this functionality[WCL+05, Gro].

[1] http://www.w3.org/Protocols/
[2] http://www.w3schools.com/website/web_tcpip.asp
[3] http://www.w3.org/XML/
[4] http://www.oracle.com/technetwork/java/jms/index.html
[5] http://www.w3.org/TR/2003/WD-wsdl20-20031110/
[6] http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-os.html
[7] http://uddi.org/pubs/uddi_v3.htm

### 2.1.2. Web Services Business Process Execution Language

As mentioned before the web service technology stack provides the means to realise distributed systems with respect to loose coupling and platform neutrality. An important part of this technology stack is the *Business Process Execution Language* in short BPEL[OAS]. This language allows the specification of business processes. It enables coordination and orchestration of web services in order to realize business goals. BPEL provides an interoperable interaction model and supports the realization of automated processes inside of corporation and in interactions on the business-to-business level. The actual version is WS-BPEL 2.0[OAS]. Only this version is considered in this thesis.

#### Basic elements

The core of BPEL are processes which can be described as an executable business process model or abstract processes. Executable process specify the exact behavior that will be executed by the processing engine. Abstract processes are only partially specified and are not intended for execution. They rather provide a possibility to describe behavior (e.g of an transaction participant) without revealing concrete operational details. Abstract process models can also be used as templates. The process and all its attracts are defined inside the <process> element[OAS, WCL+05].

As mentioned a process requires multiple logical artefacts. One type of those artefacts are *Partner Links* that describe relationships between the process and the business partners that the process interacts with. A partner link is characterized by one *Partner Link Type*. A <partnerLink> specify two roles, the role of the process itself and the role of the partner, both pointing to the respective definition inside the <partnerLinkType>. The <partnerLinkType> is specified inside the WSDL[8]. For each role inside the <partnerLinkType> a portType[9] is specified, describing the interface for this role. This construct defines an "agreement", regarding the portTypes to use for both sides in the interaction. During process execution, the binding and communication data(e.g. endpoint references (EPR)) must be available for the <partnerLink>. They can be provided upon deployment or dynamically at runtime[OAS].

BPEL processes can define *variables* to allow data handling for a process instance. The variables can be defined ether inside the <process> element or inside <scope> activities. For modification of the value of a variable the <assign> activity is used. Variables in BPEL can be used to persist data for the process(e.g temporal results) or for holding messages or data for the interaction with business partners[OAS].

*Correlation Sets* provide the means to assign messages to the process instance to which the message is addressed to. As for one process model multiple instances can be run, data has to be specified to identify the right instance. For the message routing in BPEL, business data is used to distinguish between process instances. The correlation set specify a list of properties for this purpose. Properties indirectly describe which data should be used for the correlation,

[8] http://www.w3.org/TR/2003/WD-wsdl20-20031110/
[9] http://www.w3.org/TR/wsdl#_porttypes

by providing the names of property aliases. A *property alias* is used to map the property name to the part of the message, that provides the data for correlation. The specified correlation sets can be used in activities modelling interaction through web services, for example in the <invoke> and the <receive> activity[OAS].

#### Activities

The WS-BPEL specification consists a set activities to describe the execution of process logic. These activities can be divided in two types: basic and structured. Basic activities describe single steps or tasks inside the process. Basic activities are atomic and cannot contain other activities. Structured activities are used to specify the control-flow. They can contain other activities, and dependent of the type of the structured activity the contained activities are executed in an particular order, for example in a sequence. Below an overview of the most important BPEL activities is presented[OAS].

Basic activities:

- <receive>: This activity receives a message from a service partner(web service). With createInstance="yes" it can be specified, to create a new instance for the invocation.
- <reply>: Sends a reply message to the service partner.
- <invoke>: Invokes the specified web service by sending an invocation message. This activity is used for a synchronous web service invocation.
- <assign>: This activity is used to assign values to variables. Inside of this activity statements are used for the value modification. This can be also used to perform simple operations on values like addition.
- <throw>: With the <throw> activity the process designer can model an occurrence of an fault in the process. The fault is than forwarded to a superior element containing the source of the fault. Faults can be handled by a Fault-Handler, attached to a superior element.
- <exit>: Terminates the running process instance without termination handling.
- <wait>: Suspends the process execution for the specified time.
- <empty>: This activity specifies no executional logic. When this activity is executed, nothing happens. The <empty> activity is used for description purposes and to synchronize parallel executed processes. It can also be used to suppress faults.

BPEL's structured activities include the following:

- <sequence>: The activities inside the sequence activity are executed one after an other, in the same order as they are listed in the model.
- <if>: This activity allows to specify alternative control-flows. Depending on the result of the evaluation of the specified condition, one of the contained child activities is executed.

- <while>: This activity also has an condition associated to it. On execution the activity inside of an <while> activity is repeated as long as the condition remains valid.
- <repeatUntil>: This activity executes its child activity until the specified condition becomes true.
- <forEach>: The <forEach> activity executes its <scope> activity a particular number of times, specified with the <startCounterValue> and the <finalCounterValue>.
- <pick>: The <pick> activity enables selective event processing. The activity has a set of event-activity pairs logically attached to it. When one of the specified events occur, the associated activity is executed. The <pick> activity react only once to one arbitrary event.
- <flow>: The <flow> activity enables concurrent execution of activities. The activity contains a set of child activities, that are executed at the same time. The child activities can be connected together, thus providing execution dependencies and order. Each activity inside a flow can specificity multiple activities that has to be executed prior to itself(<source>), or after itself ( <target>). Further a <transitionCondition> can be associated to each connection, that provides more control over the execution.
- The <Scope> activity is an essential element for the process specification. It allows to group activities logically and execute them in an context. This activity introduces a substantial concept to the modelling of processes in BPEL, which is presented below.

#### Scopes and Handlers

The <Scope> in BPEL provides the means to model executional context for activities inside the scope. It allows to specify variables, partner links, message exchanges, correlation sets that are only visible inside the scope.

Each scope activity contains a primary activity (basic or structured) that describes its regular behavior. Further additional process logic can be defined for spontaneously occurring situations. This logic can be specified as handlers, that contains activities which execution are triggered dependent on the type of situation. The concept of handlers is essential for business processes as it provides the means for reacting on errors and events during runtime[OAS]. BPEL specifies four types of handlers[OAS, vLLN11]:

- <eventHandlers>: Allow to react on events (messages or time triggered) independently of the control-flow inside the scope
- <terminationHandler>: Allows to specify behavior for termination of scope activities
- <faultHandlers>: Provide the means to react on faults occurring inside the scope
- <compensationHandler> :Enable to undo or compensate work done inside the scope

A scope activity can define multiple event handlers, multiple fault handlers, a compensation handler and a termination handler. The handlers are subordinate to their scope and can access all variables, partner links, message exchanges and correlation sets defined in it[OAS].

When a scope is executed, all its elements (variables, partner links, etc.) are instantiated and initialized. Also all fault handlers, termination handler and event handlers are installed and activated to listen to they respective input. The composition handler is installed after the primary activity completes successfully. After all preparations are done, the execution control is passed to the primary activity of the scope. The handlers are invoked according to their specification[OAS].

Scope activities can have nested activities that contain other scope activities, thus creating an contextual hierarchy. The "root" of this hierarchy is the <process> construct. Faults occupying on a particular level in this hierarchy can be managed on the same level with an <faultHandler> or be passed upwards the hierarchy with the <rethrow> activity[OAS].

#### Extensibility

WS-BPEL provides the means to extent it with custom attributes, elements, data types and activities. For introducing new activity types BPEL provides an extension mechanism. Inside a process model, custom activities can be placed inside the <extensionActivity> element[OAS]:

```
1 <extensionActivity>
2
3    <anyElementQName standard-attributes>
4
5       standard-elements
6
7    </anyElementQName>
8
9 </extensionActivity>
```

**Listing 2.1:** Extenstion activity schema for the WS-BPEL 2.0 Specification

The extension mechanism of BPEL is used in the scope of this thesis to extend the language with an activity representing a place holder for a process fragment, that has to be injected during runtime.

### 2.1.3. Enterprise Service Bus

The flawless interaction between different applications, running on different platforms, is a big challenge in today's business reality[Jr.07]. It requires a highly reliable communication system which must also support different communication protocols. SOA stands up to

this challenge, by providing an integration environment that allows to integrate system components with very small effort and which supports reliable messaging.

The Enterprise Service Bus allows integration with respect to loose coupling and distributed deployment. Application logic is exposed to the ESB as a service described by an abstract logical endpoint. This supports the principles of SOA as it allows to change the implementation behind an service interface without additional adaptation of other components in the distributed system. Like most SOA concepts, also the ESB is based on standardized technologies. Applications integrated by an ESB don't need to consider message routing and data transformation. For the messaging inside the ESB XML is used. Mediation services can be used for the transformation between data formats. This allows to integrate application very easily as the programmer just has to implement the to the binding to the logical. An essential part of ESB is reliable messaging, which is performed by a reliable messaging router. Integrated applications don't need to handle message resending on failures. For example, if a message cannot be delivered due to a temporal unavailability of the receiver, the messaging system handles it by itself and delivers the message when the addressed application is available again. All this features allows to separate the orchestration logic (the business processes) from the underlying services.

Following the SOA principle, services can be integrated and registered to the ESB and then be used by other applications. The integration is intended to be es efficient as possible. The integration environment provides service containers, which include a framework for management and invocation of the hosted services. The user only has to configure integration and connectivity services based on predefined services. Depending on the requirements of the distributed system, the ESB should provide sufficient connectivity services supporting the necessary communication protocols (e.g SOAP) and integration services that could for example transform messages to capable formats. It it important to mention that multiple ESB instances can be connected together, thus create a distributed integration middleware. This allows to connect applications that are spread over multiple locations, for example applications of third party providers.

The ESB is essential for the realization on and SOA based system. In this diploma thesis the open-source ESB implementation Apache ServiceMix 4.3. will be used to connect the application required to to realize dynamic process fragment injection. the open-source ESB Apache

## 2.2. Adaptive Pervasive Flows

Workflow models must provide the flexibility to allow continuous adaptation in a easy manner. In the scope of this thesis the possibility of process fragment injection in running business processes is researched and provided in a prototype. The modelling potential resulting from such a capability can be used in the multiple ways. One of the best employments is presented with *Adaptable Pervasive Flows* which are described below. With fragment injection the adaptability of processes can be significantly increased, which fits the motivation of this work.

*Adaptable Pervasive Flows* (APFs) [BLMP10, MPS+09, HRKD08], are a concept of business processes and also process fragments that extend modeling capabilities known from traditional workflows languages like WS-BPEL[OAS]. This concept is a approach to make workflows suitable for execution in highly dynamic pervasive environments by introducing flexible adaptation during run-time. The difference between workflows and APFs are their approach in process modeling and in the interaction with the execution environment. The execution of BPEL process is only dependent on the workflow engine that it is executed on and on the services its using. With adaptive pervasive flows a different perspective for the interaction between the flow and the environment are introduced. The main idea behind it is that all entities that are involved in the process execution, have their own business processes or process fragments logically attached to them, which specify the behaviour intended for them. During run-time, an entity can share its workflows with the environment, so that adaptations can be made in order to properly interact with this entity. This significantly increases the flexibility of the model as the process instance is composed on-demand during execution to fit to the current situation requirements.

As the entities interact with the system or as they change their properties either by themselves or by external forces, it may be required to deal with them differently e.g. with a better suited process. This depends on the characteristics they have at a particular time during the process execution. In order to be able to reflect to this changes, the flows are made context-aware[HRKD08]. To realize this, the modeling approach introduces a *context model* to describe the current state of the environment and its entities that play a role in the process execution. The model is shared with the whole system and consists of a set of *context properties*, which describe operational aspects of all relevant entities in the system and a set of transitions defining possible changes in these aspects. The transitions have activities or events associated with them that model the source of the state change (e.g. A context property of an postal package can be altered from "intact" to "damaged" by an event *transportation accident*). There is also the term *context configuration* introduced, that represents the state of the context at a specific time. Its like a snapshot of current context property values at a given time and its used e.g by external context-aware services for decision making regarding problems with adaptation, availability of resource etc.

With the introduction of the context model the process developer gains the possibility to model the interaction with the operation environment[10] in relation with the process execution. It enables to model the workflow in a very simple but controlled manner, without limiting it to only few predefined stiff execution patterns. The specify the desired behaviour the activities are annotated with *preconditions* and *effects*. Preconditions restrict the execution of an activity to a number of specific context configurations, providing a very good method to monitor the process progression and trigger run-time adaptation if necessary, e.g. when a entity has to be in a different state to execute the activity. With *effects* the expected result of a activity execution can be specified in terms of the change in the context configuration. In simple words the objective of the activity. All these constructs are used for the decision making to achieve the outcome expected after the execution of an process or its fragment. There create the basis for run-time refinement of process fragments described below.

[10]Here the world environment is meant where the system is used.

One if not the most important construct in the APF model is the *abstract activity*. With such an activity the developer can define a place holder for other APF fragments to be used there. The injected fragments can also contain abstract activities. Each abstract activity specifies a goal it has to achieve. The goal is defined as a context configuration that have to be reached. Based on the current context configuration, the given goal and the available process fragments the abstract activities are recursively refined at run-time in to executable processes. The refinement is done automatically. For the decision making regarding which fragment to use a external component or service can be used that various planing methods. For example in [BMPR12] the context model is formalized allowing to transform it in AI planning problem, that is than solved by utilizing AI Technology.

In [BMPR12] the authors describe a set of adaptation mechanism used in their framework to meet the adaptation needs of context-aware pervasive systems. The first one is the *refinement mechanism* which handles the refining of an abstract activity in the running instance. This mechanism is intended to automatically replace abstract activities with executable process fragments composed from the fragments provided by other entities, so that after execution the goals(context configurations) specified in abstract activities are reached from the current context configuration. The fragments used for the refinement can also contain abstract activities. They also have to be replaced with appropriate process fragments, so that the process execution can be continued, thus the refinement must be incremental. The second is the *local adaptation mechanism* which targets execution problems, more precisely the violations of the context preconditions. To solve such an issue available process fragments are composted and executed in such a way, that the system is brought to a context configuration where the process execution can be resumed. In highly dynamic environments a predefined compensation is not always enough for the process to continue. The local adaptation mechanism enables the on-demand execution of process fragments to enter an acceptable system state. The third is the *compensation mechanism*. It creates and executes on-demand a compensation process, composed from available process fragments, so that after execution the specified compensation goal for the specific activity is fulfilled. This has the advantage that the compensation is computed dynamically based on the specific execution context, thus the process compensation can better adapt to the changes in the environment, than explicitly defined activity compensations. Further *adaptation strategies*[11] are introduced. They consist of multiple adaption mechanisms combined together, thus making it possible to adapt to very complex situations.

The capability of injecting process fragments in running business processes is crucial for the implementation of a concept similar to that above. With fragment injection is it possible to increase the flexibility and adaptability of processes considerably, making them capable to run in dynamic pervasive systems.

[11] For more information on adaptation strategies please read [BMPR12].

## 2.3. Technologies

In the following sections the technologies are described that realize the concepts of Service Oriented Architecture and the ESB described in the previous sections. The work developed in this diploma thesis builds up on these technologies to provide the means of process fragment injection that could be used to realize the adaptation functionality of adaptable pervasive flows.

### 2.3.1. Fragmento

In the scope of this thesis a approach is developed to dynamically inject process fragments to running process instances. This requires, that the process fragments for the injection are stored and made available in a way, that allows to automatically retrieve the required fragment during runtime. For this purpose *Fragmento*[DS, SDH+12, SKLS10] will be used. It allows to extract process fragments and share them over the Web. It provides also function for an easy search, retrieval and reuse of process fragments in the modelled process. The Fragmento framework provides a Web-based process fragment library together with a process design environment based on an extended Eclipse BPEL Designer[SDH+12]. Fragmentos extension of the designer is briefly described in 2.3.2). The Fragmento library[SKLS10] is a repository for reusable process fragments. It provides version management, XML-validation and further functions for the work with process fragments[DS]. The integration of the framework relays on Web service technology, an OSGi-based Eclipse plug-in and process design environment extensions. The process fragments can be accessed over the Web front and or by the design environment plug-in which also utilised web service technology[SDH+12].

The framework consists of for main components[SDH+12]:

- **Deployed Fragmento libraries**[SDH+12]: They include databases and are responsible for persistently storage and version management of process fragments.
- **Web Interface**: Provides the means to share and reuse process fragments through a simple Web Service. It allows to use Fragmento in different application in a loosely coupled way.
- **Process Design Environment**: Based on Eclipse BPEL Designer
- **Eclipse plug-in**: Integrates Fragmento with the process design environment and provides a web service connection to the Fragmento libraries. It adds a graphical component to the Eclipse designer, through which the user can easily access the stored fragment.

A description of Fragmento's graphical extension of the Eclipse BPEL Designer is provided in the next section (2.3.2). In short, with the extended designer the user is able to easily extract process fragments and store them as new process files. Fragmento automatically handles the extraction of data related to the selected process fragment, like variables, XML Schema, information about the process interface and the WSDLs of the services the fragments interacts

with. The extension also provides the means to easily publish the extracted fragment to a connected Fragmento library and later select and retrieve them on demand.

The Fragmento framework plays an essential role for the concept realized in this thesis. Fragmento's library will be used to withdraw the required process fragment for the injection. Together with Fragmneto's extension of the Eclipse BPEL designer, the future results of the injection extension developed in this thesis would provide an easy way to model flexible business processes, as the user could see which process fragments are already available and based on this decide on the criteria use for the dynamic fragment selection during injection.

### 2.3.2. Eclipse BPEL Designer

In May of 2005 the Eclipse BPEL Designer Editor was created mainly by IBM and Oracle Corporation. Today it is available under the open-source license and provides comprehensive support inside the Eclipse IDE to define, edit, deploy, test and debug business processes specified in WS-BPEL 2.0. The authors list the following key features of the BPEL Designer[Fouc]:

- **Designer**: A GEF[12]-based editor that enables to graphically model BPEL processes.
- **Model**. An EMF[13] model for the WS-BPEL 2.0 specification.
- **Validation**: A validator which operates on the EMF model and produces errors and warnings based on the specification.
- **Runtime Framework**: An extensible framework which will allow for deployment and execution of BPEL processes from the tools into a BPEL engine.
- **Debug**: A framework which will allow the user to step through the execution of a process, including support for breakpoints.

As figure 2.2 shows the BPEL Designer runs on top of the Eclipse IDE framework. The designer allows to model BPEL processes simply by drag&drop of the required elements and by setting the respective properties. For this diploma thesis a version of the Eclipse BPEL designer will be used that was extended in scope of the Mayflower[SHK12] and Fragmento[DS] project. As described in section 3.3.2 Mayflower provides the means to develope processes on-the-fly and perform corrections on the process structure and workflow context in an ad-hoc manner during runtime. Fragmento(Process Fragment-Oriented Library) provides a repository for reusable process fragments and their related artefacts together with versioning and XML-validation functionality[DS]. A description of Fragmento is provided is previous section(2.3.1). Fragmento provides a plug-in for the eclipse designer that provides the graphical support for an easy extraction and management of process fragments in cooperation with a external Fragmento repository. Therefore a repository view(button-left in figure 2.2) is added to the

[12]GEF stands for Graphical Editing Framework[Foue]. It is a framework encapsulating technology for the creation of rich graphical editors and views for the Eclipse Workbench UI.

[13]EMF is short for Eclipse Modeling Framework[Foud]. It describes a modeling framework and code generation facility to develope applications based on a structured data model.

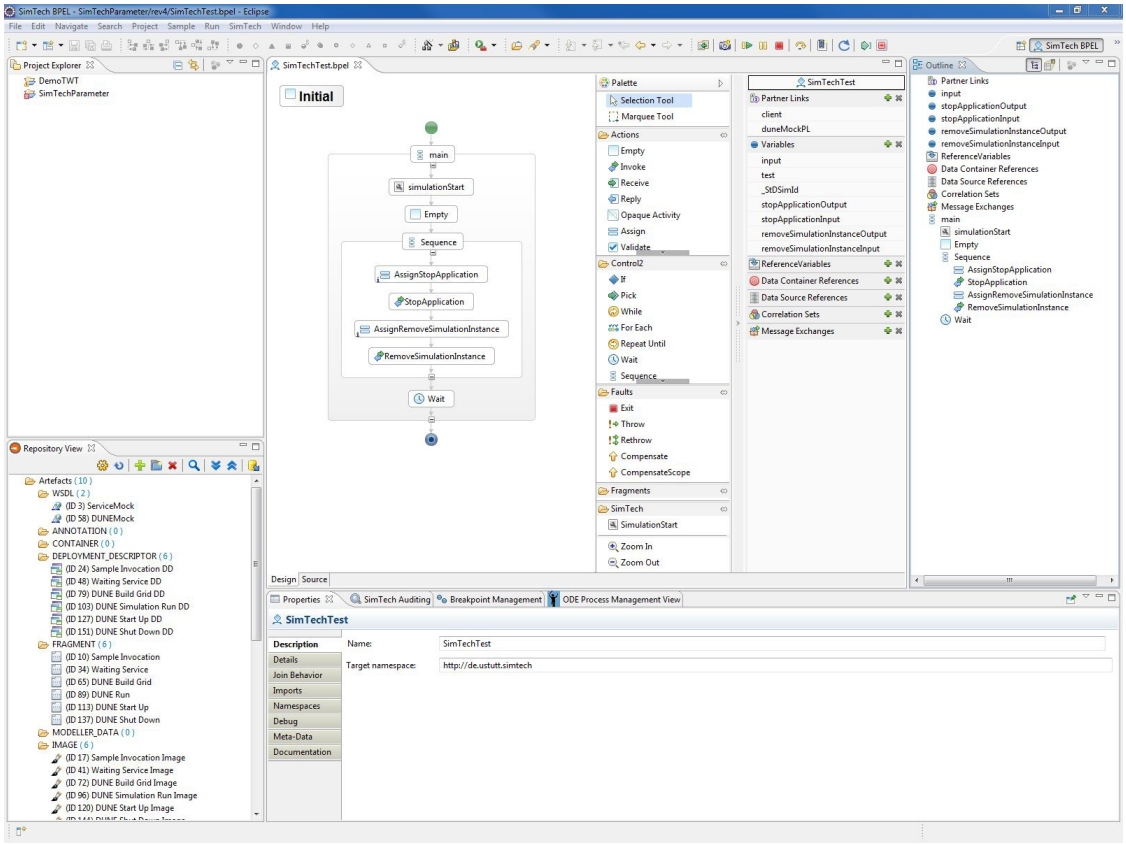


**Figure 2.2.:** Eclipse BPEL Designer User Interface

Eclipse designer, that displays the content of the connected fragment library and allows the easy publication, retrieve, search and selection of processes and process fragments. Also version control is supported through this view. The use of Fragmento is very comfortable, for example a user can mark the fragment directly on the graphical process model and perform extraction by selecting the corresponding operation in the context-menu opened by right-click. Process fragments can be easily imported by drag-and-drop.

The BPEL Designer is intended to be extensible to third-party vendors in multiple ways. For example it allows to add new activity types. Therefore it provides various extension points to extend the editor with additional (visual) elements, like property pages for extensibility of existing constructs and new elements for the extensible palette. The data model supports extensions to provide new activities or attributes and to validate them with an validator extension. Furthermore the editor's runtime deployment framework is designed to allow the extension to support different runtime engines[Fouc].

In scope of this diploma thesis the Eclipse BPEL Designer will be extended to provide the means to model processes containing a the new type of activity, the abstract activity, which also will be introduced to the WS-BPEL language in this thesis. The abstract activity represents a place holder for process fragments that may be injected dynamically during runtime. The abstract activity also includes information for the dynamic selection of the process fragment.

The process fragment for the injection are provided by the Fragmento repository. With such a extended version of the Eclipse BPEL Designer the user would be able see the available process fragments in Fragmento and specify the selection information in the abstract activity accordingly with respect to the goal to be achieved by it.

### 2.3.3. Apache Orchestration Director Engine

The Apache ODE (Orchestration Director Engine) is a implementation of a business process execution engine for processes defined in WS-BPEL[OAS]. This engine was created by the Apache Software Foundation[14]. The engine provide the means for orchestration of the web services, while handling the communication with the services, the data manipulation and error handling as specified in the process definition of the executed process. Apache ODE supports shot- and long living process executions[Apaa]. In this thesis an extended version of the engine is used. Apache ODE with Pluggable Framework Support (ODE-PGF) extends the original engine with a standardized BPEL event model[KHK+11] and functionality that allows to change the engine's behavior when an event was received[OK].

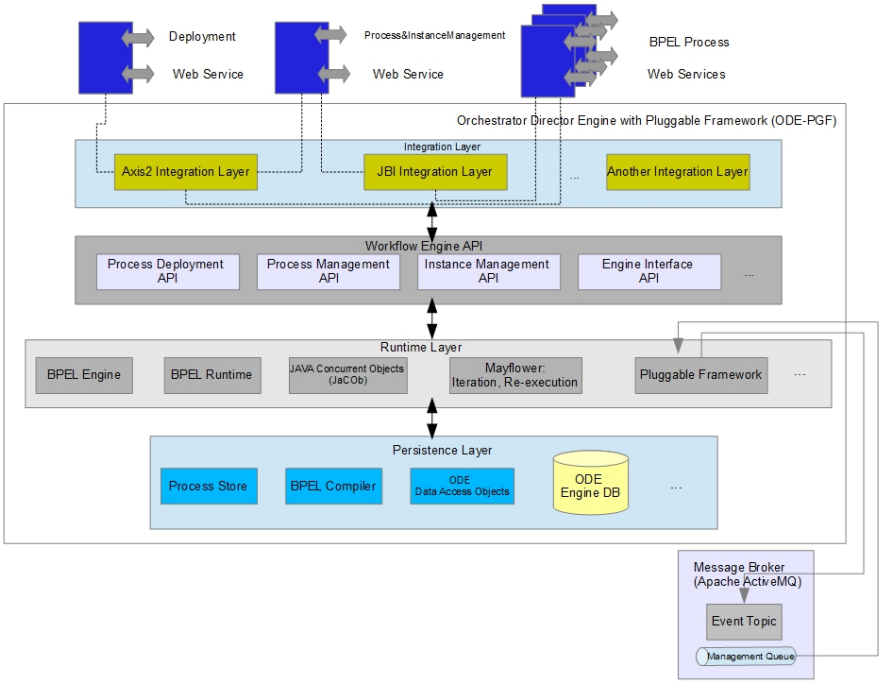


**Figure 2.3.:** Architecture of the extended Apache ODE[Hah13][15]

As described in [Hah13][15] the extended ODE is divided in four layers, each of them fulfilling a different purpose. A description for each layer, based on [Foua] and [Hah13], is provided

[14]The Apache Software Foundation `http://www.apache.org/`

[15]This architecture diagram and description is partially based on the ones presented in [Hah13]. The basic architecture and software are the same, as the engine mentioned in [Hah13] extends ODE-PGF.

below.

The fist layer is the *Integration Layer*. It provides the means for the interaction with the outside world, by allowing to embed the engine in different execution environments. Apache provides integration layers for AXIS2 and JBI, which expose the functionality of the engine though the web service interface and via a connection through the JBI's message bus respectfully. The exposed API covers process deployment along with the management of the deployed processes and their instances. Additionally the integration layer is responsible for the life-cycle management of the runtime. Further the integration layer can be extended with additional integration implementation for other execution environments[Foua].

The second laxer, *Workflow Engine API*, encapsules all generic interfaces that provide access to the internal functionality of the engine's runtime[Hah13]. This layer functions as a communication mediator between the implementations in the integration layer and the runtime layer. Requests incoming in the integration layer are forwarded to the respective component of the runtime layer providing the requested functionality. For example, requests to stop a particular process instance, received by the web service provided by the Axis2 integration layer, are passed through the *Instance Management API* to the BPEL Engine component implementing the required functionality inside the runtime layer. The *Engine Interface API* is used for the communication between the processes running inside the engines and the orchestrated services.

The *Runtime Layer* is the core of Apache's BPEL engine realization. It encapsules the classes implementing the engines functionality. It contains components allowing to compile and execute BPEL processes and handle the additional work related to the process execution, like process instantiation, management and message routing to the correct process instance. The *BPEL Engine* component provides classes that implement engine-specific functions. One example is the *BpelProcess* class, it represents the a model of a BPEL process and provides the means to create new process instances based on this model. The *BpelRuntimeContext* class is used to represent running process instances and provides functionality necessary for their execution. This functionality includes for example message correlation, operations on the variables of the process instance, external service invocation[Hah13]. The *BPEL Runtime* component consists of classes providing implementations of different BPEL constructs. To provide reliable process execution in unreliable environments, Data Access Objects (DAOs)[Mic02] are used in the runtime layer to handle persistence. The persistence facilities are described in more detail in the persistence layer explanation in the next text segment. The implementation of the BPEL constructs relays on the Java Concurrent Objects (JaCOb) framework. It provides a application-level concurrency mechanism together with a mechanism for allowing transparent execution interruption and persistence of the execution state. "Essentially, JaCOb provides a persistent virtual machine for executing BPEL constructs"[Foua]. JaCOb is essential for the reliability of Apache's engine as it allows to resume process execution after occurrence of unexpected problems, for example issues with the underlying execution environment. As mentioned before an extended version of Apache ODE is used for the purposes of this thesis. Based on the work described in [Ste08] the engine was extended with the Pluggable Framework (PGF)[OK], integrating a standardized event model for WS-BPEL 2.0[KHK+11] into the engine and providing additional event-based functionality. This extension allows

to observe and react upon engine-internal events signalized to the outside via messaging, thus improving debugging capabilities for executing process instances[Ste08]. To receive the events, wrapped as *Event Messages*, one can subscribe to the respective messaging topic(*Event Topic*) inside the message broker(Apache ActiveMQ[16]). Further a message queue (*Management Queue*) is provided to receive messages form outside the engine and by this w influence the execution of process instances. This allows other applications, for example to set values for variables in a process instance, by sending a corresponding management message. Also integrated in the extended version of Apache ODE is functionality from the Mayflower project described in section 3.3.2.

The *Persistence Layer* is responsible for storing process related data, form the runtime layer, permanently to an underlying data store. This data includes the deployed models, active process instances together with their execution state, the used partner links, message exchanges, and values of the variables. As mentioned above Data Access Objects (DAOs)[Mic02] are used for the interaction between the Runtime Layer and the Engine Database. Usially JDBC relational databases are used for this purpoce[Foua]. Per default Apache ODE provides DAO implementations for Hibernate[17] and OpenJPA[18][Ste08]. All files(BPEL process definitions, WSDLs and XML-Schemas) provided in a deployment bundle are compiled by the *BPEL Compiler* and converted into their corresponding internal representations inside the engine. The result of this compilation is an executable object model of the BPEL process, with resolved references to variables, partner links etc. Also some default objects are generated, e.g default fault handlers. The deployment bundle and the created object model are serialised as a *.cbp file. The deployment of the process models is handled by the *Process Store* component. It also triggers the compilation of the models and utilizes the DAOs to store the necessary data.

### 2.3.4. ServiceMix

The Apache Service Mix[Apab], provided by the Apache Software Foundation[19], is an open source ESB implementation. The Kernel of this integration container is built upon the OSGi[All12] Framework Implementation Apache Karaf[20]. Service Mix 4.3.0 is fully compliant to the JBI specification[Cor05] and provides the means to support users on JBI deployment and management. This environment makes possible to efficiently deploy components and applications in a loose coupled fashion.

As mentioned before ServiceMix 4.3.0 is JBI compliant, and provides JBI deployment and management functionality. The JBI container includes the Normalized Messege Routher (NMR), which provides the essential communication functionality expected in from an ESB. Interaction between the JBI and OSGi- container can be achieved, through the NMR, which makes it easier to reuse applications used with previous ServiceMix versions. Furthermore,

[16] Apache ActiveMQ: `http://activemq.apache.org/`
[17] JBoss Community, Hibernate: `http://www.hibernate.org/`
[18] The Apache Software Foundation, OpenJPA: `http://openjpa.apache.org/`
[19] The Apache Software Foundation. `http://www.apache.org/`
[20] `https://karaf.apache.org/`

the as Service Engines[21] or Service Assembles[21] packed endpoint-configurations or JBI-components are packed into OSGi bundles, on deployment in Service Mix.

Thanks to the extensible management command line console provided by Apache Karaf, a user friendly management of the components life-cycle (e.g. for OSGi, JBI) is possible. Furthermore, ServiceMix provides a hot deployment directory, where the OSGi bundles, JBI components wrapped in Service Assembles can be deployed just by copying. In reverse, to undeploy, it has just to be removed from the deployment directory.

Additional JBI components are provided alongside ServiceMix, that are already deployed as OSGi bundles. This includes for example components for different communication protocols (e.g. SOAP over HTTP) and the the integration framework Apache Camel.

In scope of this thesis the prototypical implementation of the extended ODE for process fragment injection will be integrated in the Apache ServiceMix 4.3 ESB.

[21] For a JBI component description please see:`http://docs.oracle.com/cd/E19316-01/820-4335/jbichapter/index.html`

# 3. Related Works

Dynamic process fragment injection is proposed as an approach to enable process flexibility. In this chapter a general overview is provided on different approaches enabling process adaptation and on related works that describe parts of the realization of the Adaptive Pervasive Flow concept, thus indirectly supporting the goal of the Allow Ensembles Project. In the first section the focus is directed to present existing on approaches for dynamic (on run-time) adaptation of business processes. These approaches modify the execution of a particular process instance in various ways, for example by replacing the invoked services with those better fitting the requirements.Process fragment injection works within a similar principle to provide adaptation. Subsequently, the BPEL language extensions are introduced, that provide similar and possible reusable functionalities for the realization of a framework for providing adaptation support in CAS. Finally, some projects using process fragment injection are described and compared with the approach presented in this thesis, taken into account the details of the realisation of the injection itself in these projects.

## 3.1. Dynamic Process Adaptation

Process flexibility is an important issue in large pervasive systems, in which a process must by able to adapt to the dynamic changes occurring in the operation environment. To improve the adaptability of BPEL processes, different adaptation approaches were developed. In this section some approaches are described, that provide the means for changes in the process during run-time, thus enabling on-demand adaptation of the workflow instance.

### 3.1.1. Ad-hoc iteration and re-execution

In [SK12] the authors describe an approach to enable iteration and re-execution of activities and process fragments in scientific workflows. This capability is intended to improve the work of natural scientists that frequently run simulations or experiments modelled and executed as workflows. Often in order to gain scientifically meaningful results, a part of an experiment must be repeated, taking into account different input variables or circumstances. Keeping the rest of the experiment constant is important, especially the results achieved from the process execution prior to the part that needs to be changed. To simplify such changes in experiments, additional process adaptation functionalities during execution must be supported. Therefore, such experiments can be partially or completely re-executed in order to minimize the overall necessary time to run the experiments By enforcing the re-execution of only the part that is

involved in the malfunction or needs other input, the data gathered in the execution is not lost.

Two repetition patterns are taken into account:: The iteration- and the re-execution operation. The iteration operation re-executes workflow segments without performing correcting actions or undoing already completed work. The part is run againmaintaining the actual state of the environment. The re-execution operation resets the context and execution environment with compensation techniques prior to the re-execute. It can for example undo already completed work. Figure 3.1 from [SK12] describes the interaction between a user and a workflow system implementing the here presented approach. Figure 3.1provides an overview of the supported operations in the system.

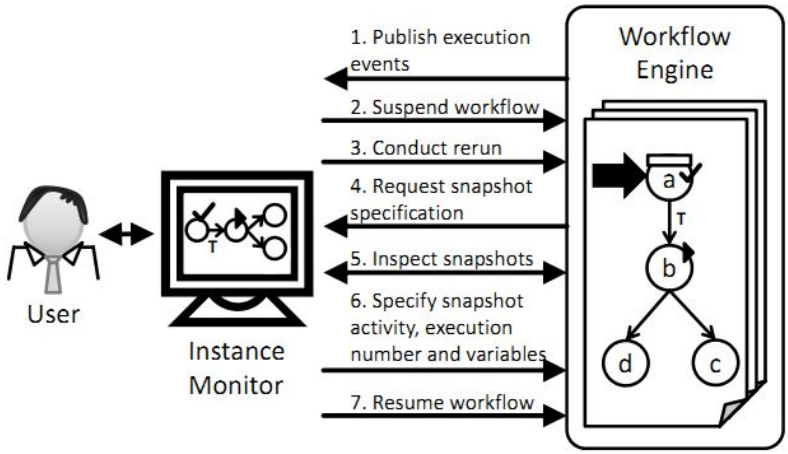


**Figure 3.1.:** User interaction with a workflow system providing repetition operation[SK12]

The engine publishes execution events, that are visible for the user. To modify the process instance the user can suspend the workflow execution, and choose an arbitrary activity as a start-point of a rerun. Than a re-execution or iteration operation can be manually triggered. To restore previous states for a re-execution, variables and snapshots from a audit component are used. Every activity that makes changes to variables gets an snapshot attached to it. The snapshots are minimized, so that they only store changes, not changed values are addressed by a reference to the prior snapshot. The user can choose which snapshot to use and which variables should be taken. Based on the snapshot the according state is entered. Than the workflow execution can be resumed[SK12].

Different aspects of the adaptation approach presented above will be used in the scope of this thesis, for example suspending the process execution for the adaptation and some methods regarding data and activity handling. Both the process fragment injection approach realized in this thesis and the iteration\re-execution approach presented in this subsection add activities to the process instance, just before they have to be executed. The Iteration and re-execution is designed for changes in process execution that are triggered and performed with human interaction. For injection on the other hand, the user only has to place the abstract activity in the process model during design time. As soon as the workflow engine recognizes the abstract activity during process run-time, the process instance is automatically suspended

and the injection of the process fragment performed. Process fragments for the scope of this thesis are provided by external components, developed in other projects. The choice regarding the right fragment to inject is also calculated externally. Process fragment injection provides more flexibility than the iteration\re-execution operations. With the latter only process fragments that were already executed are inserted again in the process instance. The approach from this thesis enables injection of new process fragments, that are not defined as part of the executed process. This way injection is providing more execution variants, as each process fragment changes the execution differently. The same applies for compensation, when placing a abstract activity in a compensation handler.

### 3.1.2. Manual ad-hoc process changes

In the ADEPT projects [PDMRa, PDMRb] methods were researched to enable quick business process implementation, deployment and adaptation. The technology, developed in scope of these projects, has been transferred in the industrial product called AristaFlow BPM Suite[1] which is currently offered as a BPM solution. In this subsection the process management system ADEPT2 is described, as for this software sufficient documentation is provided.

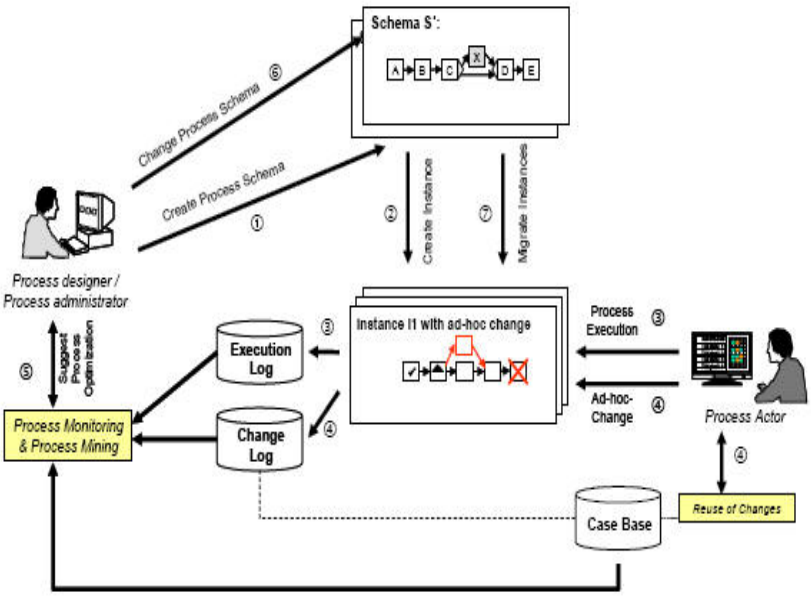


**Figure 3.2.:** Process lifecycle management in ADEPT2[PDMRb]

ADEPT2 is based on the concept of process-aware information systems(PAISs)[DVdATH05]. In PAISs the process logic is separated form application code and it is described as process templates or schemas. To create the process layer a process management system is used which provides generic functions for modelling, executing and monitoring workflow processes. The ADEPT2 system provides full process lifecycle support combined a set of adaptation

[1] www.AristaFlow.com

features (see figure 3.2). Beside the generic functions for processes, the software enables ad-hoc modifications of running process instances and supports process schema evolution[2] with the possibility to apply process schema changes to already running process instances. This software can be used to dynamically modify processes in both *process type level* (during design time) and *process instance level*(during run-time)[WRRM08]. ADEPT2 supports a set of different change patterns[RD09, WRRM08]. For example:

- *Insert Process Fragment*. It enables the user to add process fragments on a chosen position in a process schema. Its possible to add the fragment as a parallel branch o between two activities in a sequence.
- *Delete Process Fragment*. Removes a process fragment.
- *Move Process Fragment*. Moves process fragments from one position in the process schema to an other.

ADEPT2 provides process changing functionality for the process designer as well as for a user participating in the process. The software developed in scope of this thesis targets the automation of process adaptation in a highly dynamic executional environment. ADEPT2 does not support the execution of partially incomplete process models (schemas). The injection approach implemented in this thesis provides flexibility by allowing to specify place holders(abstract activities) in process models. Those activities are replaced during runtime with process fragments that are executed accordingly.

### 3.1.3. Run-time Web-Service Replacement

In the context of the ReFFlow project[DK] an generic approach was developed to provide flexibility of web service compositions. This approach supports adaptation by extending the generic mechanism of web service lookup and binding in BPEL engines, with means to replace web service instances used by process activities. This enables for example the adaptation to new quality of service (QOS) requirements, as an workflow engine can performing service adaptation by dynamically replacing a malfunctioning service[KHC+05, KB04]. In this section the extended mechanism is referred as the "find and bind" mechanism.

The ad-hoc replacement of participating web service instances is performed by dynamic binding of WS ports to process instances at run-time. The process designer can specify default selection policies in the deployment descriptor during design time. Further process administrators can assign activities alternative policies, which are used for port selection during run-time. For this purpose the BPEL language was extended with a the `<find_bind>` extension element, which can be placed within BPEL's activities modelling the interaction with web services. Namely the `<invoke>`, `<receive>`, and `<reply>` activities. The `<find_bind>` element contains a `selection_policy` attribute with which the process administrator can control the selection of compliant web service instances.[KHC+05]

[2] *process schema evolution* describes a concept where the the schema of a particular process type in continuously updated to meet new requirements and cope with environmental changes. Often it necessary for long-running processes instances to migrate to a new version of the schema[Rin04].

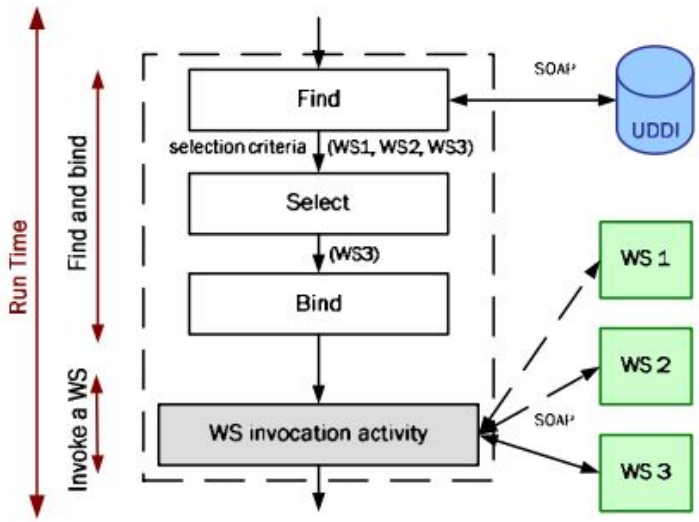


**Figure 3.3.:** The "find and bind" mechanism[KHC+05]

Im figure 3.3 the "find and bind" mechanism is presented. The approach is divided in three main steps performed during run time:

- In the first step (find), all available web services are listed, that are compliant to the portType specified for the executed activity.
- Next a port is selected from the list obtained in the first step. Here the user-specified selection criteria(e.g. QoS) are considered.
- In the last step(bind) the selected port is bind to the activity of the process instance. After this step the chosen web service is executed.

In [KHC+05] the authors mention two use cases of the presented approach. The first case describes the process instance repair during run-time. The "find and bind" mechanism enables a automatic repair of process instances, when a participating web service fails and no other compliant ports are available that are meeting the selection policies defined upon deployment. In this case, the selection criteria specified in the `<find_bind>` element are used. The second use case describes the modification of selection policies, due to new user requirements. To adapt the process instance to meet the new requirements, the `selection_policy` attribute can be modified with monitoring tools supporting viewing process instances and value changes at run time. This way its not necessary to stop the process instance to adapt its model, thus avoiding effecting all process instances of this model.

The approach presented above would provide similar adaptation possibilities to process fragment injection, if the ports changed during runtime would invoke other processes, that provide the same functionality as the process fragments considered in this thesis. Such a realisation would require that each possible selection alternative for the dynamic binding is

already deployed and offered as a web service. This implies, that only completely specified processes can be chosen for adaptation. The injection approach taken in this thesis, supports execution of partially specified process fragment models, without the need to expose them as already instantiated processes with web services interfaces.

## 3.2. Language Extensions

In this section BPEL extensions are presented that extend the language with constructs providing similar adaptation capabilities as the abstract activity introduced in thesis. Also included in this section is the "Context4BPEL" extension[WKNL07], as it provides process execution context awareness required in systems using adaptive pervasive flows and a valuable method for formalization of BPEL extensions.

### 3.2.1. Context for BPEL Language Extension

Workflows to be able to adapt to changes occurring in the physical world, need to be aware of these changes and be able to handle information from various sources about different situations in the physical reality such as position or state. Such information is referred to as context[WKNL07]. In this subsection the BPEL language extension "Context4BPEL" is presented based on the work described in [WKNL07].

In [WKNL07] the BPEL extension "Context4BPEL" were described, introducing concepts for modelling context-aware workflows. In the scope of the diploma thesis [Fon09], a prototype of the Context4BPEL execution environment were implemented based on the integration of the open global platform for location- and spatial-aware applications Nexus[3] and use of Apache Orchestration Director Engine[Apaa]. In [WKNL07] three conceptual elements were introduced:

- *context events* representing asynchronously occurring information, produced by the context management system, about changes in a environment state.
- *context query* enabling synchronous access to context data with a query language. The queries are executed at the context management platform. The results are transferred into the internal workflow data.
- *context decisions* providing the means to control the process execution flow considering context information acquired ether form internal process data or as a result of a context query. For context decisions context-aware operators are used. These operations provide comparison functions for all types of *primary contexts*, for example identity, location and time[GBH+05, WKNL07].

[3]Nexus Project: `http://www.nexus.uni-stuttgart.de`

In the diploma thesis [Fon09] the BPEL language has been extend with six new activity types. One for context decisions and context queries respectively, and four to represent the operations interacting with the context management platform to handle context events(receive, update, register, de-register). Further a set of context variables, were introduced, to handle context information internally in the process. With a context query, a process instance can obtain context data and store it in the context variables. The handling of context variables was not realized in the prototype.

The BPEL language extension for process fragment injection will be performed, with a methodology inspired by the formal BPEL extension for the "Context4BPEL" approach presented in [Fon09]. The formalization is done, be extending the abstract syntax meta-model for WS-BPEL 2.0 provided in [KML08].

### 3.2.2. Formalization of Adaptive Pervasive Flows

In [BLMP10] the authors present a formalisation for a language for adaptive pervasive flows called APFoL (Adaptable Pervasive Flow Language). It offers a formal model for APFs by extending the Blite[4] language. Blite provides formal semantics covering a significant part of the to WS-BPEL language. For this reason it was chosen as the base for the formalisation of adaptive pervasive flows[BLMP10]. As described in section 2.2 the concept of APFs presents workflows intended for dynamic pervasive environments. Each physical entity in such an environment has a process or process fragment logically attached to it, which describes the functionality and behavior of it's entity. The processes (and fragments) can contain place holders for process fragments of other entities. When it is required for two entities to interact with each other, the "holes" are filled with process fragments published by the entities providing the required functionality in order to fulfil a particular task[MPS+09].

In order for APFs to react on changes in the physical environment they must be context-aware and provide the means to preform necessary adaptations[MPS+09]. In [BLMP10] the constructs are formally defined that meet those requirements. Blite[4] as a formal basis for WS-BPEL was extended with constructs enabling context handling and vertical adaptation[5]. In the following some of the added constructs are presented, based on the description in [BLMP10].

- The *abstract activity* construct enables the modelling of vertical adaptation[5]. It "represents a partial design-time specification of a flow model or a abbreviation of a complex activity"[BLMP10, p. 7]. For the refinement of an abstract activity the process fragment definition is ether assumed to exist or is given at run-time.
- A *context event* is a special type of activities modelling the reception of events from the *Context Manager*[6] module. In this activity the process engine waits for a event to be

[4] `http://rap.dsi.unifi.it/blite/`

[5] "Vertical adaptation refines the flow or re-maps services to the flow, without affecting the flow structure."[MPS+09, p. 7]

[6] A module responsible of monitoring contextual constraints and providing information required to evaluate contextual conditions[MPS+09].

received.

- *Constrained scopes* specify scopes with defined entry and exit points and a constraint. Within the scope the constraint must be valid, otherwise a exception is raised and an adaptation is performed, for example to return in a environmental state, from which the process can continue its execution.
- A *contextual IF* allows to model alternate branches that are chosen based on *context conditions* each branch has. The first branch for which the context condition is valid is executed. Also it is possible do define a branch without a context condition to model default behavior.
- A *context handler*: is a alternate flow associated with the main ?ow or ?ow scope. The flow is executed if the condition in the corresponding scope is violated. It can be also triggered by an fault or event.
- The *contextual one-of* specifies a set of alternative branches with context conditions attached to them. The branches have also rollback flows associated to undo the work done by the already executed part of the branch. The first branch is exceeded, for which the associated context condition holds. The condition is monitored during the hole execution of the branch. When the condition is violated, the execution of the branch is stopped and its rollback flow is triggered. When the rollback is done a *context jump* is performed, where the execution is restarted at the branch selection (condition check).

In the scope of this thesis vertical adaptation is provided by dynamic process fragment injection. It is the realisation of the abstract activity construct introduced in [MPS+09] and formalized in [BLMP10], thus covering one functionality of adaptive pervasive flows. Dealing with the execution context and adaptation management in terms of process fragment selection is not covered in this thesis.

### 3.2.3. WS-BPEL Extension for Sub-processes

To enable modularization and reuse in large business processes a BPEL[OAS] extension for sub-processes was developed under the name BPEL-SPE[KKL+05]. The extension provides the means to execute business processes as a sub-process of a superior process. The sub-processes can be defined inline in a process or be modelled as a standalone (sub-)process. In deference to defining a separate business process an using it as a service with an invoke activity, the sub-processes introduced with the BPEL-SPE extension are coupled to their parent process in terms of life-cycle, interaction and data access. For example if the parent process is forcefully terminated or just compensated the invoked sub-processes are treated accordingly. To realize this a special *Coordination Protocol* between processes and sub-processes is also introduced[KKL+05, IA].

The interaction between the parent process and sub-processes is realized in therms of normal invocation of services, but is usually limited to the initiating request and the final reply message. A sub-processes defined inside of the parent process can be only used form within

the same process. Is the sub-process defined as a separate business process, than it can be reused across multiple BPEL processes[KKL+05].

The BPEL-SPE extension supports also the scoping and error handling known from simple BPEL processes. Is the sub-process defined inside a scope, the same scoping rules apply to the sub-process as they would apply to a standard BPEL activity. This includes also accessing variables from the scope where they are defined. Inline sup-processes can access global variables of the parent process, but not variables in other nested scopes. Sub-processes defined as separate processes cannot access variables from other processes. The extension also supports data exchange between the calling process and a called inline sub-process[KKL+05].

BPEL-SPE introduces a new type of activity the <call> activity, which specifies the invocation of a sub-process inside of the parent process. The new activity allows easily modelling of complex tasks inside of business processes while keeping the control provided inside of BPEL processes. When the <call> activity is executed, the referenced sub-process is invoked with the specified input variables. The execution control is passed form the <call> activity to the sub-process. It the sub-process completed, the output data stored inside the output variables inside of the parent process and the control is returned back to the branch containing the calling <call> activity. Fault are handled the same way as with standard BPEL activities[KKL+05].

The BPEL-SPE extension provides approximate similar functionality with the <call> activity, as the injection approach realized in this thesis does with the <abstract activity>. Both provide the means to execute business processes as a part of another process and not just as an service invocation. This allows to cover complex tasks as single activities in the main process thus providing multiple abstraction layer for modelling a business process.

But there are some significant differences between the BPEL-SPE extension and the one developed in scope of this thesis. One key difference is that process fragment injection provides dynamic selection of the fragment to execute during runtime, thus providing good flexibility. With BPEL-SPE the sub-processes need to be ether defined or referenced from inside of the parent process at design-time. So the decision regarding the which sub-process has to be used must be made before execution and it is static. An other difference is that process fragment injection provides more modelling flexibility, as it allows to run partially unspecified processes. The process designer can leave out same parts of the process and only specify the result to achieve within an <abstract activity>. The missing fragments are injected during run-time.

## 3.3. Fragment Injection

In this section projects are presented related to the runtime injection of process fragments. The approaches used in these projects are compared with the approach presented in this thesis, taken into account the details of the realization of the injection provided in these projects.

### 3.3.1. ASTRO-CAptEvo

The ASTRO-CAptEvo framework[RBK+12, BMP13] provides the means to define adaptable context-aware business processes that can be used in highly adaptable service-based systems (SBSs). The framework realizes the concept of APFs (presented in 2.2), where the interactions among entities is achieved through the interaction of their business processes. In ASTRO-CAptEvo the adaptations are performed automatically during run-time, using sophisticated AI planning techniques to compose the available services. The framework supports both vertical adaptation (process refinement) and horizontal adaptation (execution of additional steps to continue with the process).

The framework uses three adaptation mechanisms for the process adaptation[BMPR12]:

- The *refinement mechanism* is responsible to for refining of abstract activities in the running process instances. Abstract activities are refined with executable process fragments composed from the fragments provided by other entities[BMPR12].
- The *local adaptation mechanism* solves execution problem, by composing and executing available process fragments in such a way, that the system is brought to such a state, that the process can continue its execution from the point where it got stuck because of a precondition violation[BMPR12].
- The *compensation mechanism* is used when the process execution cannot be restored only by performing additional tasks, but require the compensation of previous process steps. It creates and executes on-demand a compensation process, composed from available process fragments, so that after execution the specified compensation goal for the specific activity is fulfilled. Such a goal can for example specify a state so that the execution can be continued at a point where the process splits into multiple branches, thus providing again the alternatives for execution[BMPR12].

As presented in figure 3.4[RBK+12], the framework is divided in three layers. The *Presentation layer* provides a graphical interface for the user interaction. The *Adaptation layer* is responsible for solving adaptation problems, occurring during process execution. In this layer selections are computed regarding what adaptation mechanism should be used and what process fragments need to be composed in what way, to solve the adaptation problem. The *Execution layer* contains all components enabling the execution of the adaptable context-aware business processes of the entities cooperating with each other. The processes are modelled with the CAptLang[BMP13] language. This layer is also responsible for the detection of execution problems and the initialisation of the adaptation. The adaptation of the process instance is preformed inside of the execution layer based on the solution provided by the adaptation layer[RBK+12, BMPR12].

The process *execution engine* inside the *Execution layer* is of special interest. It provides similar process fragment injection functionality as the engine to be developed in scope of this thesis. The engine inside the ASTRO-CAptEvo framework provides the functionality of a conventional process engine with additional features to enable process adaptability[Rai12, p.231]. It implements process execution according to the semantic rules described in [BMP13].

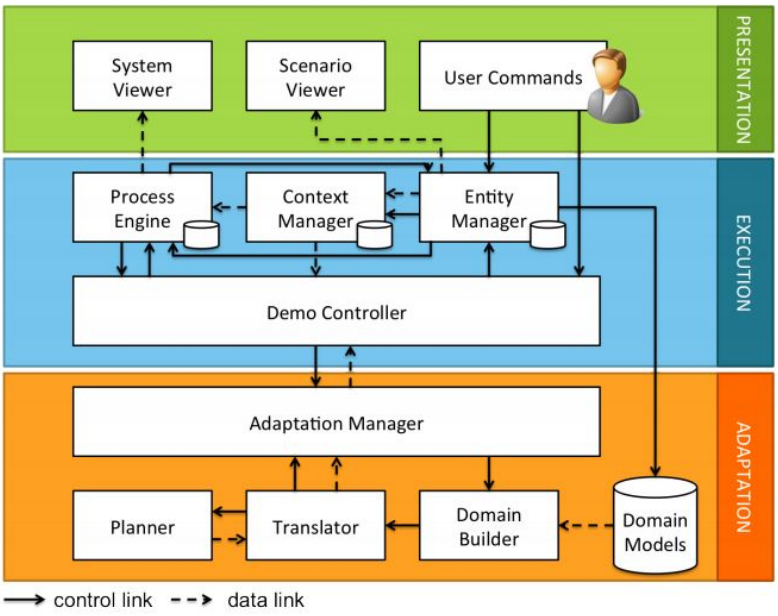


**Figure 3.4.:** Architecture of the Astro-CaptEvo Framework [RBK+12]

The engine can detect situations that require adaptation and in the case of such pass the necessary information about the adaptation problem to the adaptation layer. To detect such situations, the process execution is monitored with respect to constrains specified in the process model, information about the physical world (current context) and the availability of process fragments. Further the engine is able to suspend the execution of process instances. When adaptation needs to be performed, the process is suspended, the adaptation processes provided by the adaptation layer are executed and the execution of the process instance is resumed. Additional "jumps" inside of the suspended process instances can be performed to continue the execution form a different point, as it were suspended. Additional viewers were implemented in the ASTRO-CAptEvo framework, to provide the means to examine processes during or after execution[Rai12, p.231],[RBK+12].

The work described in this thesis strongly relays on the concepts presented in the ASTRO-CAptEvo framework. In the thesis the focus lays on the business process engine. The goal is to extent it to enable process fragment injection, in order to provide vertical adaptability. To reach this goal the WS-BPEL language will be extended with the <AbstractActivity>, which was introduced in [BMP13]. To be able to model processes and process fragments containing the new activity the Eclipse BPEL Designer will be extended with the appropriate data model, graphical representation, and conversion controls. Similarly to the work presented in this section, the abstract activity will contain information about the contextual goal to be achieved with the process fragment replacing the activity. Also the activity will contain information specifying security concerns and preconditions for the execution of the process fragment. Analogically to the ASTRO-CAptEvo framework the execution engine must be extended with functionality to suspend the process execution and perform adaptation. In this thesis the Apache Orchestration Direction Engine (ODE) will be used and extended accordingly, for the

execution of the enhanced process models.

In the scope of this thesis the process engine will be extended to provide better adaptability. For process fragment composition and solution of the adaptation problems external components will be used. The realization of such a composition and adaptation manager and the context management is out of scope of this thesis, as the focus lays on the execution engine and the injection functionality.

From the process engine description of the ASTRO-CAptEvo framework provided in [Rai12, p.231] can be concluded, that the process is suspended, the adaptation process is executed separately, and than the execution of the previous process is resumed. In the engine extended for this thesis a slightly different approach will be taken to realize the adaptation. When the engine detects an abstract activity as the next activity to be executed, the main process is suspended and the activities of the adaptation process fragment are directly injected in the execution queue of the main process instance. Then the execution of the enhanced main process is resumed, resulting in the execution of the activities of the adaptation process and a flawless transition to the main process activities. This approach has for example the advantage, that no additional life cycle management is required.

### 3.3.2. Mayflower

The *Mayflower*[SHK12] workflow environment was developed to support natural scientists in they research. The software was introduced in [SHK12] under the name "Mayflower" which is short for Model-as-you-go Workflow Developer. It allows to develop business processes on the fly with the trial-and-error approach that is typical in researches that require experiments and simulations to be performed very frequently.[SK10]. Mayflower provides a impressive collection of features and covers various approaches to handle business processes and increase flexibility while operating with them in a user friendly manner. Among the integrated approaches the authors mention instance migration, versioning and backwards loops. The workflow environment provides the means to handle runtime faults, perform corrections on the process structure and workflow context in an ad-hoc manner during runtime[SHK12].

The approach described in section 3.1.1 provides the means to perform ad-hoc adaptation on process instances, by allowing to iterate and re-execute parts of the process instance with and without compensation respectively[SK12]. This approach was prototypically implemented in the scope of work of the research cluster SimTech[7] and integrated in Mayflower[SHK12].

For the realization of the Mayflower workflow environment different components were implemented for auditing, resource management, process modelling and execution. For the implementation of the modelling framework the Eclipse BPEL Designer [Fouc] was extended with further elements to control the process execution (including the iteration and re-execution functionality form [SK12]). Further interfaces were added for the visualisation of the process execution inside the modelling framework and for assisting on the information

[7] `http://www.simtech.uni-stuttgart.de/index.en.html`

evaluation (e.g. snapshot review). For process execution Apache's Orchestration Director Engine (ODE) [Apaa] has been extended by various components, like the *Event Publisher* and the *Iterate/Re-execute* component. Also some already existing modules in Apache ODE were modified. For example the *Navigator*, which is responsible for the traversing along the workflow graph and for triggering of activity execution.

When the function preparing the ad hoc rerun is in invoked, the content of the execution queue inside Apache ODE is correctly and consistently adapted. The queue holds a list of all activity instances scheduled for execution in as part of a particular process instance and a list of completed activities.

The approaches for process fragment injection in the scope of this thesis are inspired by the realization approaches used in the implementation of the ad-hoc iteration and re-execution functionality in Mayflower. Similar to *Mayflower*, the workflow environment developed in this thesis will be based on standardised workflow technology. Both the Eclipse BPEL Designer [Fouc] and Apache's Orchestration Director Engine (ODE) [Apaa] will be used for the prototypical implementation of the injection functionality researched in this thesis. Further the injection itself will be also realized by suspension of process execution and modification of the lists inside of the execution queue object of Apache ODE. This approach was chosen because it allows to dynamically select and execute inferior processes or process fragment as part of a main process, without the necessity of explicit management and synchronization of the life-cycle of the inferior processes. Also it allows to benefit from the advantages of the standard workflow technology, with respect of the extensibility and already realized functionality.

The work of this thesis does not cover the visualisation of the process adaptation and fragment injection. Further the point in the workflow, where the process fragment has to be injected needs to be modelled with an abstract activity at design time. That means, that the injection approach in this thesis is intended to provide adaptability in an dynamic manner (performed at run-time without manual intervention), and not in an ad-hoc manner where the user initiates adaptation.

### 3.3.3. Framework from "Runtime Process Adaptation for BPEL Process Execution Engines"

In [TZ11] an approach for process adaptation is presented that is intended to be realisable with various existing BPEL execution engine implementations. To integrate an execution engine with the approach considerable small extension effort is needed. This way adaptable workflows can be supported with the framework presented by the author, without being tied to a specific process execution engine. The supported adaptation covers structural modification of process instances during runtime. This structural modification can be used to add process fragments to the process instance. The framework achieves this adaptation support for BPEL engines by providing an abstract process model for description of the executed process, an adaptation pattern model describing the adaptations to be performed on the process instance and an adaptation engine for the generic adaptation of the process

instance. The execution engine has to provide sufficient monitoring capabilities and has to be extended only with a small set of generic adaptation functions to provide adaptation support together with the framework[TZ11].

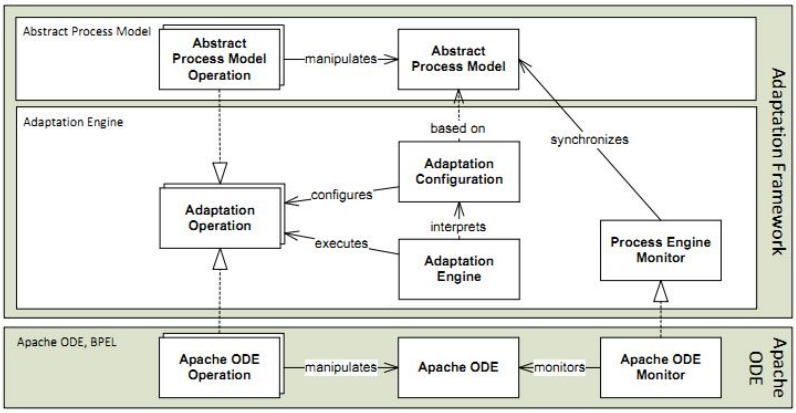


**Figure 3.5.:** Overview of the adaptation framework extending Apache ODE (from [TZ11])

Figure 3.5 shows an overview of the adaptation framework integrating the Apache Orchestration Direction Engine to provide adaptation support. The *Adaptation Framework* operates on a generic process model for an abstract representation of the process instance and its runtime information. The framework also includes models to describe adaptation. Based on an abstract process model and defined adaptation structures, the adaptation of a particular process instance can be specified. Specific adaptations for a process are represented with instances of the *adaptation pattern model*. An *Adaptation Configuration* contains further information for the adaptation, for example information about the position where a new process activity need to be added and the information about the added activity[TZ11].

The *adaptation engine* interprets the information provided inside the *adaptation pattern model* and *adaptation configuration* to coordinate the adaptation. The execution of the process instance is suspended for the adaptation. The adaptation engine simultaneously performs the adaptation on the abstract process model and delegates the adaptation to the integrated execution engine on which the process instance is executed. The adaptation itself consists of a set of basic adaptation operations (like inserting or deleting an activity), which the execution engine must support. It is necessary to extend the execution engine only with those operations to integrate it with the adaptation framework. So when an adaptation operation is invoked on the abstract process model its realization is triggered inside the process execution engine as well[TZ11].

As mentioned before, the execution engine needs also to support sufficient monitoring capabilities to provide information for the abstract process model. If it is not the case, the engine has to be extended with those capabilities. In the example presented in [TZ11] Apache ODE was extended using Aspect-Oriented-Programing so that the engine was not effected with the changes performed to add monitoring functionality[TZ11].

The framework described in [TZ11] could be used to inject process fragments in BPEL

processes. But in this thesis the focus lays on a dynamic approach, that allows the process fragment selection during runtime. In the presented framework an adaptation needs to be modelled in advice. Providing dynamic adaptation with the framework, would require to generate at run-time an adaptation pattern and adaptation configuration for each process fragment that needs to be inserted in to the process. The adaptation of the framework would take much effort. The generation of the required models would also be an additional step, that would increase the overhead during a single injection. Due to the previously mentioned reasons, the approach will be not used in scope of this thesis.

# 4. Concepts and Requirements

This chapter is provides a overview of the requirements and constrains for a framework implementing process fragment injection based on BPEL, the de-facto standard language for business process definition. A overview, describing the general idea for the realization of such a framework based on already existing BPEL technology(Eclipse BPEL designer and Apache ODE) is presented. Further included in this chapter is the formalization of the BPEL language extension and an example of the introduced abstract activity is provided.

## 4.1. Requirements Analysis

This section describes the requirements for a framework realizing dynamic process fragment injection in BPEL processes.

First general requirements for the framework are described. Then the requirement specification is described in a two level perspective. First the requirements at the model level are specified, covering requirements for the language as well as the design environment. The second is the execution level, which provides a requirement analysis at level of process instances.

### 4.1.1. General requirements

In this thesis the focus lays at enabling of the injection of process fragments and does not cover handling the execution context of the operational environment nor the automatic composition of "adaptation patterns" to react to a problematic situation in the operational environment. The goal of this thesis is to extend an a service orchestration engine in such a way, that it allows the execution of processes containing abstract elements. These abstract element do note specify concrete execution logic, but provide only information about the goal that needs to be achieved. When the engine recognizes the abstract element, it has to refine it with process logic fulfilling the specified goal and execute this logic directly during runtime.

The injection of the process fragment has to be performed dynamically at run-time to provide increased flexibility in order to adapt the process to the current situation occurring in the operational environment. This requires a formal language to define business processes and process fragments for the injection. BPEL will be extended for this purpose. Further an execution framework has to be provided to execute these processes. Such an framework requires an execution engine supporting fragment injection, a mechanism for the selection of

an appropriate process fragment as well as a repository storing available process fragments and providing them on demand.

Process fragment injection is developed in the scope of this thesis in order to allow to react on situations occurring in highly dynamic operational environments. Depending on the situation, the framework automatically must select a appropriate process fragment and inject it in the executed process, to solve a problem or handle the situation in a convenient way. This requires the execution environment to observe the state of the operational environment in order to create a executional context for the process execution. Based on this context information and the "goal description" modelled inside the abstract activity, the Adaptation Manager must compute the best choice regarding the process fragment to inject. In order to provide context information for the Adaptation Manager it must be considered how and where the context monitoring will be handled. This could be done similar to the concept described in section 2.2 where a graph-based model is provided describing the current state of the operational environment (the execution context).

As mentioned before this thesis focus on the engine and process fragment injection end execution. In the framework implemented in this thesis, the decision regarding the process fragment to inject is delegated to an external *Adaptation Manager* component. Furthermore to provide the context information to the Adaptation manager a context management functionality must be provided. Due to the limited scope of this thesis the realization of the context management must be provided by future projects. Process fragments for the injection will be also provided by an external component.

Please note, that the Adaptation Manager has yet to be realized in an different project. For the purpose of this thesis a dummy implementation of the Adaptation Manager will be created, that receives the data from the abstract element and returns a random ID of an process fragment.

For storing and providing process fragments, the process fragment-oriented library Fragmento(see 2.3.1) will be used.

### 4.1.2. Language and Modelling Requirements

In this section the requirements for the language and design environment are described.

#### Language extension

To enable injection in BPEL a well defined conclusive extension for the language is required that allows the designer to provide basic information for the injection and mark points in the business process model, where a process fragment should be injected. To enable this, a custom activity must be introduced to the BPEL language. The new activity must be compliant to the WS-BPEL specification and it should contain context information important for the refinement such as the pre-conditions and effects(as seen in the work described in 3.3.1). The new activity, named "abstract activity", will be integrated by using BPELs build-in extension mechanism.

This ensures the correctness of the extension in BPEL. To ensure the correctness in the abstract activity itself, a formalization is presented in section 4.4.

In the previous section it was mentioned, that process fragment injection requires the interaction between multiple components. Such an cooperation between the engine performing the injection, the Adaptation Manager(selecting the fragment) and Fragmentio(fragment repository) requires additional information, that has to be provided inside the process (or process fragment) model. This information must be considered in the language extension. Therefore the extended model must cover the following information:

1. Service endpoint for the communication with the Adaptation Manager.
2. Service endpoint for the communication with the Fragmneto.
3. The position in the business process where the process fragment should be injected. (This point is already covered by the abstract activity).
4. Information describing the task to be completed by the injected fragment.
5. Further information upon which the Adaptation Manager can base the selection of the fragment.

As mentioned in the previous section, this thesis focuses on the injection of process fragments. The realization of the Adaptation Manager goes beyond the scope of this thesis. Nevertheless, an prototypical data model (see 4.5) for the abstract activity is created, to coverer the Adaptation Manager-related requirements listed above. The model provides provides information about the goal to be achieved as well as other context-related information that could be used to select a appropriate process fragment. The data model is based on the work presented in [BMP13]. This model could be used in the future in a implementation of the Adaptation Manager.

Furthermore the behavior modelled with an abstract activity must be clear and situations have to be avoided, where an model could be interpreted in multiple ways. This is especially true for the control transition, when the control is passed from the parent process to the abstract activity and further to the injected process fragment. Also the transition back to the parent process is critical. For this reason constrains for the injected process fragments must be specified, so that the modelled behavior is clear and conclusive. This limitations are described in section 4.2.2.

#### Modelling environment\BPEL-Designer extension

To allow process designers to employ the flexibility advantages of process fragment injection, a modelling environment has to be provided that supports the BPEL extension from this thesis. The requirements related to such an modelling environment are described in this subsection.

The modelling environment has to support the BPEL language extension developed in this thesis. This means the environment must allow to model processes and process fragments

containing abstract activities in an easy manner. In order to provide this functionality, support for the abstract activity needs to be added to the extended version of the Eclipse BPEL Designer2.3.2. The user must be able to add and edit abstract activities as he would do with any other standard BPEL activity in the Eclipse BPEL Designer, so the corresponding functionality has to be added, including:

- A graphical representation of the abstract activity for the graphical modelling
- The underlying data model of the abstract activity
- Graphical elements to model the information inside the abstract activity
- Translation logic to the corresponding <abstractActivity> representation in the extended BPEL language
- Automatic changes to the related BPEL elements, to satisfy all requirements of the BPEL language[1]
- Automatic attachment of the required XML-schema files to the BPEL project
- Validation for the abstract activity definition inside the BPEL document.

Further the user needs the means to create and manage process fragments and the related artefacts (e.g. WSDLs) required for the injection, and add them to the Fragment repository, so that the execution engine can access them to inject them in running processes. This functionality is already covered by the Fragmento extension for the Eclipse BPEL Designer described in 2.3.2.

An important requirement is the backward compatibility to previous extensions of the Eclipse BPEL Designer. The functionality added in the scope of this thesis may not impact the functioning of other extensions or the original Eclipse BPEL Designer. For example process deployment directly from the designer must be supported. Other functions like stopping, pausing, setting braking points and the functionality from Mayflower and other projects must be kept. To ensure this, the other extensions must be analyzed regarding how they function, and if necessary workarounds must be provided for the extension introduced in this work. One example is the instance execution view from Mayflower, where the activities in the graphical model change colours, depending on their execution statute. Upon injection the model inside Eclipse BPEL Designer is not changing its structure. But it should be compensated, by showing that the a abstract activity is executed.

### 4.1.3. Requirements for the execution environment

The execution environment enabling process fragment injection is fundamental for the concept of Adaptive Pervasive Flows (see 2.2). In provides a great improvement for business processes in terms of flexibility. To achieve this, the injection functionality will be added to the extended version of Apache ODE (see 2.3.3). The engine will be integrated with other components

[1] For example it is important to set the mustUnderstand variable in the extension declaration in the BPEL document as true.

participating in the injection, building together the the execution environment. In this section the requirements are presented for the execution environment allowing process fragment injection.

#### Integration

As mentioned in the previous section, to allow dynamic process fragment injection during runtime, the execution framework requires some functionality responsible for the automatic selection of a fitting process fragment for the injection and a fragment repository allowing to share the fragments between the design environment and the execution environment. As stated before, the scope of this thesis does not cover the realisation of the Adaptation Manager (selection mechanism), but will use a external dummy implementation instead. The fragment repository was already implemented as part of the Fragmento project(see 2.3.1). Nevertheless, this adds new requirements to the framework in terms of integration:

- The must provide the means to integrate the Adaptation Manager and Fragmento in a loosely coupled manner, to allow the easy replacement of the their implementations in the future[2].

- The communion between the execution engine, Fragmento and the Adaptation Manager must be performed in a secure and reliable manner. This is important because all three participating components are necessary for the injection and the reliability of the whole system depends directly on the reliability of the connection between them.

- The execution engine must provide interfaces for the interaction with the Adaptation Manager and Fragmento.

#### Context Information

As described in the general requirements section (4.1.1) for the selection of a appropriate process fragment the Adaptation Manager requires data about the current execution context. The context management can be ether realized directly in the engine or be delegated to an external component. The realization of the context management and its integration in the engine is beyond the scope of this thesis, as the focus lays on developing approaches for the process fragment injection itself. Independent from that, interfaces for the communication between the engine and the Adaptation manager will be created. Future project realizing the context management can extent those interfaces to allow the transmission of context information to the Adaptation manager.

[2]This is especially relevant for the future Adaptation Manager realization, so it can be integrated with as less effort as possible.

#### Execution

The execution engine must support the BPEL extension introducing the abstract activity. For this purpose the extended version of Apache ODE(see 2.3.3) must be further extended. Below a list is presented summarizing the requirements for the such extended engine. The engine must be able to:

- Recognize abstract activities in the process models provided by the design environment
- Interpret abstract activities and compile them to a engine-internal representation
- Execute of processes containing abstract activities (compliant to the WS-BPEL specification)
- Provide the means to allow communication with other components required for the injection (Adaptation Manager, Fragmento)
- Transmit the information form the abstract activity to the Adaptation Manager and receive an ID of an process fragment
- Retrieve the required process fragment from Fragmento, together with all artefacts required by the fragment
- Inject and execute the process fragment in place of the abstract activity. This covers the injection of the process logic provided by the process fragment in the same process instance containing the abstract activity. Also, the artefacts o the process fragments must be added to the process instance. The execution of the added logic should be performed as it would be just an activity, so that after completion the process instance can continue it regular execution.
- If the injected process fragment contains further abstract activities, these must be also executed accordingly in an recursive manner.

#### Correctness

The injection of process fragments and utilization of the required artefacts has to be done in a correct manner to ensure consistency of the refined process. BPEL scopes of the process instance have to be considered as well as other atomic constructs. Especially the correct exception must be guaranteed, when the abstract activity is placed inside a <flow> activity, which models parallel execution of its child activities. Furthermore the injection may not influence other processes already running on the engine. It have also to be ensured, that the communication patterns of the parent process remain unchanged.

#### Visibility of adaptations

The injection and execution of process fragments must be made somehow visible for the user. The execution environment must provide the means to validate the execution. For this purpose already available auditing tools will be used.

#### Life-cycle

The execution of the process can be divided in to multiple levels, describing the depth ot the recursive process fragment injection. As figure xxx shows, each recursively injected process fragment is executed on an new logical execution level. An important requirement is, that the executed process fragments are tightly coupled in terms of life-cycle and error handling, to the invoking process or process fragment (called parent process). The injected process fragments should behave as like it were executed as part of a nested BPEL scope. For example, when a parent process is terminated (due to an error in a parallel executed branch), the injected process fragment have to be terminated too.

#### Error handling of injected process fragments

Process fragment injection adds new possibilities to dynamically handle errors occupying during business process execution. This extends BPEL's model for error and compensation handling. As mentioned in the life-cycle requirement above, each recursively injected process fragment is executed in a deeper execution level. In BPEL when an error occurs in an BPEL Scope nested inside an other BPEL Scope, the unhanded error is forwarded to the upwards the hierarchy. Similarly it has to be done for the execution levels of the injected process fragments. If an error is not handled on one execution level it is forwarded to the upper level.

For the compensation a different approach is intended. When a error occurs during the execution of an injected process fragment, the engine must allow to compensate it accordingly. It must be possible to execute compensation logic for each injected process fragment. This compensation logic can be defined inside the process fragment(when it defines a scope) or be injected.

#### Access to data of the parent process

Injected process fragments must be able to access variables of their direct parent process. This is necessary to enable data exchange (input and output) between injected process fragments and their parent processes.

#### Backward compatibility

The extension of the ODE engine for fragment injection have to keep the previously integrated functionality. The user must be able to follow the execution (execution tracking and auditing). Other functions like stopping, pausing, setting braking points and the functionality from Mayflower an other projects must be kept in the engine. For this the other ODE extensions must be analyzed on how they work. Some functionality like repetition marks for Mayflower or the events for the BPEL Designer might not for the injected process fragments. If possible workarounds or other solutions need to be provided to allow other extension to function properly, but without providing their functionality for the injected fragments or parts of the main process effected by the injection.

#### Concurrency

Apache ODE supports parallel execution of multiple process instances. Consequently the injection mechanism must be able to support simultaneous injection in multiple process instances. In order to fulfil this requirement, the engine extension must provide concurrent execution of the injection operation and communication functionality with a corresponding correlation mechanism. The correlation is important so the data(e.g process fragments) form the Adaptation Manager and Fragmento can be associated with the right process instance. Commutation issues like for example delays also have to be considered.

## 4.2. Identification of possible constrains

In this section possible constraints for the process fragment injection are identified. First the constrains from the WS-BPEL specification are described. Further other language and modelling constrains are presented and the constrains for the execution environment are described.

### 4.2.1. Constrains given by the WS-BPEL specification

The WS-BPEL 2.0 Specification[OAS] provides various mechanisms to extend the language with custom definitions. The language extensibility is described under the point 5.3[3]. The addition of new activity types is presented under the point 10.9. in the specification[4]. WS-BPEL allow to put namespace-qualified attributes on any WS-BPEL elements. Also elements from other namespaces can appear within WS-BPEL elements.

[3] WS-BPEL, Language Extensibility: `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html#_Toc164738482`

[4] WS-BPEL, Adding new Activity Types: `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html#_Toc164738511`

In the scope of this work we want to introduce our custom activity. For this, new elements expanding the language can be placed inside the <extensionActivity> element(see listing below). To improve the understandability the new elements will be referred as *extension elements*.

```
<extensionActivity>

   <anyElementQName standard-attributes>

      standard-elements

   </anyElementQName>

</extensionActivity>
```

**Listing 4.1:** Extenstion activity schema for the WS-BPEL 2.0 Specification

To ensure the right performance of WS-BPEL engines the following constrains regarding additional activities are specified:

- The <extensionActivity> element must contain a single element, the extension element, qualified with a different namespace than the WS-BPEL namespace.
- The extension element inside of the <extensionActivity> must provide WS-BPEL's standard-elements and standard-attributes.
- If the extension element is not a element recognized by the WS-BPEL processor and the mustUnderstand-attribute is not set as "yes", then the unrecognized activity must be treated like a an <empty> activity with the standard-attributes and standard-elements of the extension element. Other attributes and child elements of the extension element are ignored.
- The standard-attributes and standard-elements have to be treated as defined by the WS-BPEL 2.0 specification, regardless if the extension is understood or not.
- If an <extensionActivity> contains other activities, its highly recommended to set mustUnderstand="yes".
- The extension element must meet the WS-BPEL required semantics(explanation below).

In general the extensions in BPEL can be optional or mandatory. This is defined with the mustUnderstand attribute. If it is set to "yes" the extension in mandatory and has to be supported by the WS-BPEL implementation. If the BPEL processor does not support the extension, the process definition containing them must be rejected. Then the extensions that are optional and not supported by a BPEL engine have to be ignored.

If the mustUnderstand-attribute is not set to "yes", the WS-BPEL processor performs the static analysis after the non-standard-attributes and non-standard-elements of an unrecognized

extension activity are ignored. When one of the ignored elements contains a element or attribute essential for the consistency of the process model, violations can occur. In [OAS] examples for this are provided. For better understanding we will also present them below.

A requirement for a process model is that it has to contain at least one start activity. If we use a <customStartActivity> which contains WS-BPEL's standard start activity, we will get an validation error, because our custom start activity as an non-standard child of the <extensionActivity> would be ignored.

The other example presents the case, when a link is defined across the boundary of the <extensionActivity>. Links have exactly one source and target. If the non-standard child of the <extensionActivity> is ignored, the link has no source or target respectively.

### 4.2.2. Other constrains for the language

The WS-BPEL specification defines a model allowing to represent business processes. The modelled behavior can be very complex, and injection of further process fragments increases this complexity. To provide some control over this complexity, constrains must be set, which help to prevent errors in the modelling and execution of processes specified in the extended BPEL language.

#### An abstract activity as an join activity.

The BPEL extension should allow to place Abstract activities as join activities inside a <flow> activity. But to allow this in a conclusive manner some limitations must be set, so that the modelled behavior can be interpreted correctly and only a single way. To provide a better understanding of the matter the following example is presented:

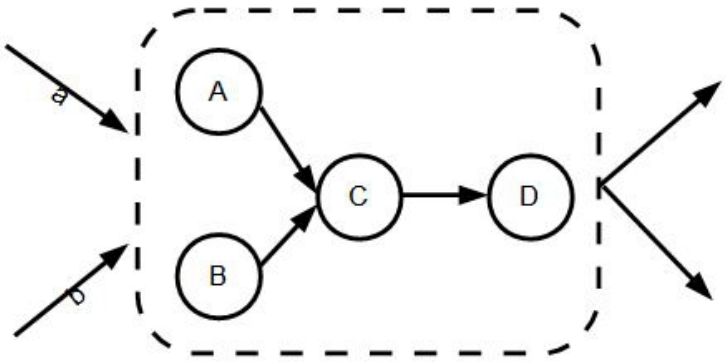


**Figure 4.1.:** Process fragment inside the abstract activity

The figure 4.1 presents a abstract activity inside a <flow> activity. The abstract activity has two incoming links it. Inside the activity is a process fragment displayed, that was selected

and injected in the process instance. The injected process fragment has two start activities. Now such a injection can be interpreted in multiple ways. One is, that when the link a was activated, the injected activity A should be executed. On activation of link b, the activity B should be activated. A other way to interpret such a injection is, that both activities (A and B) should be executed, when one incoming link is activated (or when both links are activated if the suppressJoinFaliures attribute is set).

To provide a clear model for the behavior upon injection, it was decided to limit the injected process fragments to such, that have only one start and only one end activity. Furthermore, the abstract activity can only be used with the suppressJoinFaliures attribute set to yes. This prevents unwanted behavior upon injection and limits the execution to one injection operation per abstract activity. Of course repeatedly executed abstract activities placed inside loops (e.g the <while> activity) allow that multiple injections.

#### Abstract activity scope

To ensure the atomicity of the injected process fragments, those have to defined inside BPEL scopes. This also ensures that injected process fragments are not influenced by the parent process and solves the problem of inconclusive behavior when the execution control is passed to the injected process fragment (described in the previous constrain). Normal lexical scoping rules apply to the process fragments. Process fragments injected directly into the main process can access global process variables. Variables of the process fragment are only accessible from within the scope it is defined in and its nested scopes. This is also congruent to the execution level concept, where injected process fragments are executed on logical execution lever representing the depth of recursive injection. The BPEL scopes also ensure the correctness of the process fragment on parallel execution inside a <flow> activity.

#### Process fragments consisting of just an other abstract activity

Process fragments consisting of just an other abstract activity are not allowed, in order to block the possibility of endless refinement loops. This could be considered as a drawback on the flexibility of the adaptation, because by injecting just an other abstract activity, the goal of the previous abstract activity could be changed during runtime. But in order to keep the modelling simple and avoid situations where process designers lose the overall overview of the expected behavior of the modelled process, it was decided to keep this limitation.

### 4.2.3. Constrains for the modelling and execution environment

In order to provide a conclusive and user friendly environment to work with process fragment injection, some constrains must be set that prevent unwanted results. Below a set of constrains is presented that are ether defined for the concept of process fragment injection or are given by the technology (Eclipse BPEL Designer, Apache ODE) on which the realization will be based on.

#### Interoperability with Mayflower

As described in section 3.3.2, Mayflower provides functionality visualizing the execution of a process instance and allowing to iterate and re-execute parts of the process instance. This functionality is not interoperable with the injection approach introduced in scope of this thesis.

Lets first consider the visualisation of process execution. When a process fragment in injected in a process instance, it's model is not changed. So the execution of the process fragment cannot be displayed by the usual means of Mayflower. Nevertheless, when an process fragment is injected and executed, the abstract activity representing the injection should be marked as "executing". When the injected process fragment is complied or changes its state otherwise, this should be also displayed accordingly.

Regarding the iteration and re-execution functionality, also constrains should be considered. The implementation of Mayflower(see 3.3.2) inside Apache ODE is based on manipulation of the content of the execution queue inside ODE, where the objects implementing BPEL activities are stored waiting for execution. The future realization of the injection functionality will also operate on the ODE'S execution queue. A injected process fragment might influence the iteration and re-execution operations performed by mayflower, resulting in malfunctions and even unwanted execution of process logic. For this reason iteration and re-execution of such parts of a process, that contain abstract activities will not be allowed.

#### Constrains of Eclipse BPEL Designer

Abstract activities must contain information, based on which an appropriate process fragment can be selected for the injection. This information can describe the goal to be achieved with the injected fragment and other relevant aspects. This requires a complex data model of the abstract activity. One constrain regarding this is related to the Eclipse BPEL Designer. Custom elements placed inside the extension activities are not supported by the automatic serialization and deserialisation mechanism inside Eclipse BPEL Designer. For this reason ether a data model for the abstract activity must be provided with no custom elements or a workaround must be considered, to serialize and deserialize custom elements.

#### Behavior regarding non-standard attributes and elements

The behavior of the engine regarding non-standard attributes and elements must be implemented according to the WS-BPEL specification. This means that not supported elements and attributes must be ignored, when there are "not subject to a mustUnderstand="yes" requirement from an extension declaration"[OAS]. Furthermore unknown activities contained within the <extensionActivity> must to be treated as <empty> activities that have the standard-attributes and standard-elements of the unrecognized activity.

## 4.3. System Overview

In this section a general overview of the system is provided describing the main components and their interaction. The goal of the system is to support dynamic process fragment injection during runtime. This concept allows to define processes which an incomplete specification of its business logic. Instead of providing a concrete specification of the behavior, a process designer can define places in the process, where the missing business logic should be automatically selected during runtime, based on an abstract goal, and executed directly. This dynamic selection and execution of business logic during runtime is referred to as process fragment injection. The incomplete parts of the process are modelled with abstract activities. Inside an abstract activity the modeller can specify the goal that needs to be reach with the business logic (process fragment) injected in the process instance. The selection and retrieval of process fragments for the dynamic execution inside a process instance is referred to as refinement. The extended version of the BPEL language, which introduces the abstract activity and the injection functionality is called BPEL-Inject.

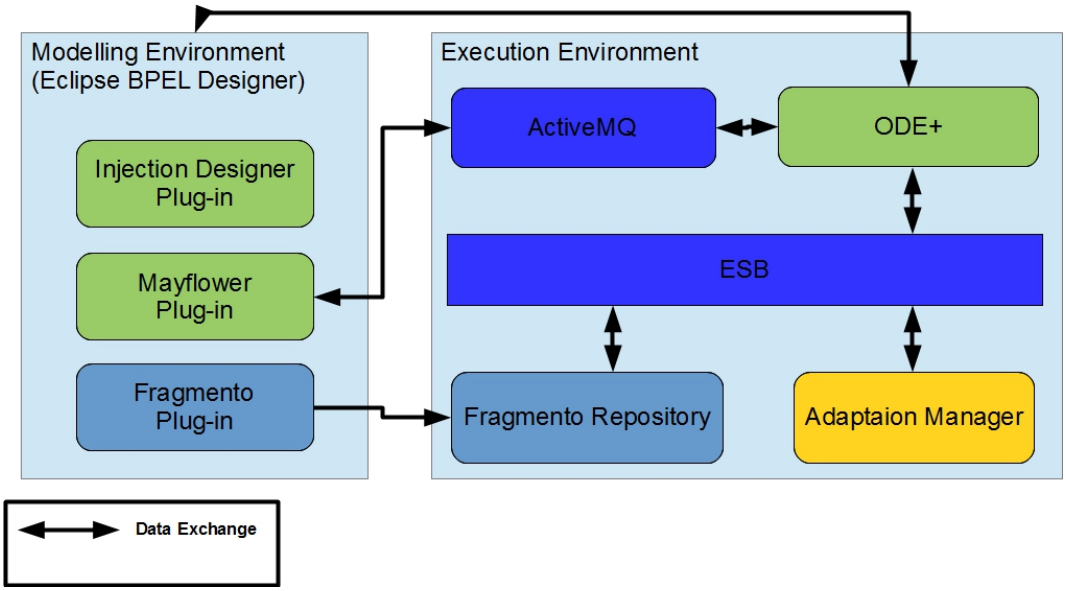


**Figure 4.2.:** Architecture of the Injection Framework

Figure 4.2 presents an overview of the framework supporting process fragment injection. The framework consists of two main components: The *Modelling Environment* and the *Execution Environment*. Below a description of the components is provided:

The Modelling Environment, allows to model business processes specified in the extended version of BPEL, BPEL-Inject, which introduces abstract activities and the injection concept. The Modelling Environment is based on the extended version of Eclipse BPEL Designer(see 2.3.2). The extension includes the Fragmento Plug-in which provides functionality to create process fragments defined in BPEL and allow to store them in the Fragmento Repository. Also included in the designer extension is the plug-in from the Mayflower project, which together with the extension inside Apache ODE allows to iterate and re-execute parts of the process.

The *Injection Designer Plug-in* is developed and integrated to Eclipse BPEL Designer in the scope of this thesis. This plug-in extends the designer with the means to support process modelling with the abstract activity.

The Execution Environment allows to execute processes containing abstract activities and provides the necessary means for the injection of process fragments. In order to accomplish this, the Execution Environment contains multiple components connected together in a loosely coupled way. The communication is realized with use of standardized messaging protocols like SOAP and the ESB realization Apache ServiceMix (described in 2.3.4). The Execution Environment integrates the three components required for dynamic process fragment injection during runtime: The Fragmento Repository providing the process fragments for the injection, the Adaptation Manager responsible for the selection of an appropriate process fragment for the injection, and the extended Apache ODE proving an execution engine for the BPEL-Inject processes containing abstract activities. The engine is the core of the framework. Not only it is responsible for the execution of the BPEL-Inject processes, but it also coordinates the injection and provides the means for further adaptation of the process instance in case of errors or other unexpected situations. When a process needs to be refined, the engine suspends the execution of the process instance and transfers the data specified inside the abstract activity to the Adaptation Manager[5], which selects a fitting process fragment and returns its Fragmneto-ID back to the engine. The engine uses the provided ID to retrieve the corresponding process fragment form the Fragmento repository. After the fragment is successfully obtained, the engine injects it in the refined process instance and resumes its execution. In result the engine executes the injected business logic and directly continues with behavior specified in the process model.

Also included in the Execution Environment is the message broker Apache MQ[6], which is part of the Apache ODE extension introducing the Pluggable Framework (described in section 2.3.3, that allows event-based interaction and control of the engine. ActiveMQ is also used inside the Mayflower extension, for example to realize the visualisation of the process execution.

## 4.4. Formalization

This section presents the formalization of the abstract activity extending the WS-BPEL 2.0 specification. The extension consists of the abstract activity and few components representing information, based on which external services will provide the process fragments for the injection. At first a small part of the textual notation used in the syntax formalization of WS-BPEL 2.0[KML08] is introduced. This work was chosen as the basis for the formalization presented here, because it is compact, provides good readability and is suitable for extension. The formalization presented in this section is inspired by the work shown in [Fon09]. Further

[5] At this point also information about the current execution context (described in 4.1.1 and 2.2) could be transmitted to the Adaptation Manager, if context monitoring would be realized inside the engine.

[6] Apache Software Foundation, Apache ActiveMQ: `http://activemq.apache.org/`

for the purpose of this thesis the syntax for the abstract activity is specified and the semantic is explained.

### 4.4.1. Introduction of the notation

In order to enable process fragment injection in BPEL, the functionality of the abstract activity has to be realized, provided as an extension activity. First it must be formally defined. To provide better readability, the necessary definitions from [KML08] are presented in this section.

The mentioned abstract syntax formalization of WS-BPEL describes it's workflow meta-model as a tuple $M$, including all required process information defined by the WS-BPEL 2.0 standard[OAS]. It contains a set $\mathcal{A}$ consisting of the process activities, the `process` element representing a specific process model, the function $\mathsf{type}_{\mathcal{A}}$ specifying the types $t \in \mathcal{T}$ of the activities $a \in \mathcal{A}$. The process definition also includes workflow variables $v \in \mathcal{V}$, and functions $\mathsf{name}_{\mathcal{A}}$ and $\mathsf{type}_{\mathcal{V}}$ assigning respectively the name and the type of the BPEL variable $v$. The tuple $M$ contains more constructs, that are not described here to avoid unnecessary confusion and improve the readability of this thesis. All definitions used and extended in the scope of this work are properly explained below. For the whole process definition please see definition 3.1.1 in [KML08].

The WS-BPEL meta-model defines a number of different activity types $t$. It the referenced formalization there are specified in the set $\mathcal{T}_{\mathcal{A}}$ presented below:

$$\begin{aligned}\mathcal{T}_{\mathcal{A}} = \{&sequence, flow, if, pick, while, repeatUntil, forEach, scope,\\ &invoke, receive, reply, wait, assign, empty, throw, rethrow,\\ &compensate, compensateScope, exit, validate, opaqueActivity, extensionActivity\}\end{aligned}$$

Definition 3.2.1 from [KML08]

Further in definition 3.2.2 from [KML08] the function $\mathsf{type}_{\mathcal{A}}$ is specified which assigns each activity $a \in \mathcal{A}$ in the process model $M$ to a particular activity type $t \in \mathcal{T}_{\mathcal{A}}$:

$$\mathsf{type}_{\mathcal{A}} : \mathcal{A} \rightarrow \mathcal{T}_{\mathcal{A}}$$

A set $\mathcal{A}_t$ consisting of all activities $a$ of a given type $t$ is defined in [KML08] as follows (definition 3.2.3):

$$\forall t \in \mathcal{T}_{\mathcal{A}} : \mathcal{A}_t = \{a \in \mathcal{A} \mid \mathsf{type}_{\mathcal{A}}(a) = t\}$$

Not only activities and their types must be specified, but also their content. A activity can contain further elements of different types. As we can read on page 8 in [KML08], the set `STRUCT` is specified in definition 2.1.3 as "the set of types, which are defined or included in the process definition".

Further on page 9, in definition 2.1.4, it is specified, that the set STRUCT also includes the set $\mathtt{XML}_{\mathtt{STRUCT}}$, which is "the infinite set of XML-Schema types that can be defined using XML-Schema and are included in the process definition"[7]. This definition also specifies that:

$$\mathtt{STRUCT} \subset \mathtt{XML}_{\mathtt{STRUCT}}$$

The set TYPES is specified in Definition 2.1.11 as the "infinite set of all types of all programming languages including all XML Schema types"[KML08, p. 9]. A formalization of the WS-BPEL types is provided in the abstract syntax formalization of WS-BPEL[KML08] in section 2.1.

The last definition which is needed in this section is the definition of the children function, which "returns the set of immediate descendants of an activity"[KML08, p. 20].

$$\mathsf{children}: \mathcal{A} \rightarrow 2^{\mathcal{A}}$$

The whole specification of this function and the hierarchy relation, this function relays on, is presented in [KML08, Section 3.5].[8]

### 4.4.2. Extension of the meta-model

The activity type *extensionActivity* from $\mathcal{T}_{\mathcal{A}}$ is of special interest for this work. It allows the extension of the WS-BPEL meta-model with customized activities, according to the WS-BPEL 2.0 standard. For the purpose of modelling processes with adaptable elements, which are replaced on runtime via injection, the WS-BPEL meta-model needs to extended with the abstract activity. To provide a formalization at the same time, the formalization from [KML08] is extended as described below.

#### Addition of abstract activities

For an extension to be done according to the WS-BPEL specification, an custom activity must be of type extension activity. More precisely, the new activity needs to be defined as the only child element of the extension activity. To distinguish between extension activities containing abstract activities among other possible extension activities, the set $\mathcal{T}_{inject}$ of extension activity subtypes is defined:

$$\mathcal{T}_{inject} = \{abstractActivity\}$$

[7] For information about XML-Schema please see http://www.w3.org/XML/Schema

[8] The mentioned definitions are long and would distract from the topic of this section, thus there are not included here

Please note, that the set $\mathcal{T}_{inject}$ could also contain other extension activity subtypes, which would represent other extension activity implementations. For the purpose of this thesis, only one subtype needs to be introduced, the *abstractActivity* subtype.

Further a function is required, in order to associate custom activities to their subtypes. The function $\texttt{subtype}_A$ returns the subtypes for the introduced activities $a \in \mathcal{A}$ a particular extension activity subtype $t \in \mathcal{T}_{inject}$:

$$\mathsf{subtype}_A : \mathcal{A} \rightarrow \mathcal{T}_{inject}$$

The abstract activity introduced in this work, belongs to it's own *abstractActivity* subtype. The set $\mathcal{A}_{abstractActivity}$ is created, which contains all activities $a$ of the subtype *abstractActivity* from $\mathcal{T}_{inject}$:

$$\mathcal{A}_{abstractActivity} = \left\{ a \in \mathcal{A} \mid \mathsf{subtype}_A(a) = abstractActivity \in \mathcal{T}_{inject} \right\}$$

The WS-BPEL 2.0 specification, specifies constrains for the extensions that can be made. Those limitations were already described in 4.2.1. Among other restrictions, it is specified that the contend of the extension activity can only be a single element(simple or complex). The extension activity itself cannot contain any further element or attributes. Also it is required, that the child element of the extension activity provides standard-elements and standard-attributes required by the meta-model in an activity definition. To represent that a abstract activity $a'$, as the child element of the extension activity $a$, provides the required elements and attributes, we define the relation:

$$\forall a' \in \mathcal{A}_{abstractActivity} \exists a \in \mathcal{A}_{extensionActivity} : a' \in \mathsf{children}(a) \wedge a \neq a'$$

Please note that $a$ itself contains no standard-attributes and standard elements, and it contains only one immediate child element($a'$). That means, that the function $\mathsf{children}(a)$ is bijective. So with this relation we meet the requirements described above, as this is a bijective relation. There might be also other extension activities besides our abstract activity, for them an additional definition would be needed.

The $\mathsf{children}(a)$ function provides the means to define a the set $\mathcal{A}_{inject}$ containing only extension activities, that have a abstract activity as a child:

$$\mathcal{A}_{inject} = \left\{ a \in \mathcal{A}_{extenstionActivity} \mid \mathsf{children}(a) \in \mathcal{A}_{abstractActivity} \right\}$$

Finally the abstract activity needs to be added to the meta-model defined in [KML08]. Again custom activities need to be specified as a subtype of the *extensionActivity* type. So the set $\mathcal{A}_{extenstionActivity}$ from [KML08] is redefined to a new set $\mathcal{A}'_{extenstionActivity}$, additionally containing the above defined set of all abstract activities $\mathcal{A}_{abstractActivity}$:

$$\mathcal{A}'_{extenstionActivity} = \mathcal{A}_{extenstionActivity} \bigcup \mathcal{A}_{abstractActivity}$$

The set $\mathcal{A}_{inject}$ is a subset of $\mathcal{A}_{extenstionActivity}$, thus it does not need to be explicitly added to the new extension activity set $\mathcal{A}'_{extenstionActivity}$.

#### Contend of the Abstract Activity

In the previous section the formal framework for the abstract activity was created, now it's content needs to be specified. All elements that are required to properly represent information necessary for the refinement of process fragments, have do be defined as types. Through the extension created in this work, the following types are introduced:

$$\{goalType, preconditionType, effectType, pointType\} \subset \texttt{STRUCT}$$

It's planned that the information represented in the above mentioned data types are forwarded to a external service for the decision making regarding which process fragment should be used for the refinement. Specifying the whole context model for the implementation of adaptable pervasive flows and for the external service is out of scope of this work. For the purpose of this thesis a example data model will be used, based on the work presented in [BMP13]. In definition 2.1.3, 2.1.4 and 2.1.11 from [KML08] it is specified, that it is possible to specify the required data types using XML-Schema with a different namespace. To define the types needed in this thesis, this option is exploited. In appendix A the definition of the abstract activity as well as the type definitions of the elements is provided.

### 4.4.3. Semantics of the abstract activity

After the meta-model extension formalization is defined in the previous section, in this section the necessary semantical extensions are analyzed and specified. The abstractActivity is defined by means of providing a place holder for runtime process fragment injection. The execution of the abstract activity is described and the meaning of the example information is shortly introduced:

When an abstractActivity is identified during the process execution, the process instance execution is paused to perform the following tasks on a new inferior execution level:

1. Creation of a new BPEL-Scope for the execution of the process fragment
2. Reading refinement information contained in the abstract activity's attributes and elements
3. Send refinement information to the external Adaptation Manager
4. Receiving reference information (Id) about the process fragment to use
5. Sending a request containing the received reference information to Fragmento[DS] (the component providing process fragments)
6. The returned process fragment is dynamically deployed on the engine.
7. The process fragment is executed. The execution may be performed follow one of the behavioural cases described below:
    a) The process is performed without any further adaptation.

b) The process fragment is further refined due of an nested abstract activity, then executed.

c) The process fragment is re-refined(replaced) because of an error during execution. Here several strategies are taken into account. Afterwards the new process fragment is executed.

8. After the injected process fragment is successfully executed, the engine resumes its higher execution level and continues.

If the process fragment also contains one or more *abstractAtivities*, these are recursively executed the same way as described above. Generally, an abstractActivity is refined dynamically during run-time, when the activity should be executed by the engine. In the case of multiple abstracActivities, in the main process or the used fragments, the refinement also done, according to the progress of the workflow execution. Always one at a time.

There are two mechanisms to handle errors occurred while executing abstractActivity:

**Process Refinement at the execution level** One way to deal with the error is with further refinement. For example a process fragment can be injected and executed to compensate the already executed part and then restore a state, where the process can be continued. These approaches are called *adaptation strategies* and are briefly described in 2.2. In this work the injection of process fragments in running processes is enabled. This capability can be further used to implement adaptation strategies.

**The Modeling level** When en error cannot be handled with adaptation, it has to be managed within the BPEL model. In this case a BPEL-exception is thrown. The process designer needs to model the exception handling at design time. The exception may be handled for example with a compensation handler.

There are multiple points, along the execution process described above, where errors can occur. In the following part it is described how these errors will be handled:

When an error during the invocation of the Adaptation Manager occurs, a BPEL exception is thrown, which can be later handled i.e. with a fault handler. Independent from that if the Adaptation Manager is unavailable due communication problems or internal issues, without a decision regarding which process fragment to use, the process execution cannot be continued and also adaptation strategies cannot be performed. That is the reason, why the error needs to be considered during design time.

Errors with the interaction with the software Fragmento, providing the process fragments, need to be considered depending of their type:

**Technical issues** If we are dealing with technical issues (i.e connection problems), we cannot complete the refinement. I this case we throw a BPEL-exception, as the execution of the process cannot be continued.

**Missing process fragment** In the case, that a process fragment cannot be found, the possibility has to be provided to choose a alternative process fragment. In this case a new request to the Adaptation Manager component is send, with the additional information about

the unavailability of the previous process fragment. Since the context can change very quick, the decision about how many attempts to make and how to avoid infinite loops of retries, lays also in the responsibility of the Adaptation Manager. When the manager communicates to abort the execution, a BPEL-exception is thrown.

In the case of an error during dynamic process fragment deployment, we provide the same alternatives as with the process fragment software. Ether an attempt can be made to deploy other process part. Or an BPEL exception can be thrown. Again, only the means for injecting process fragments are provided. The decision making must be done by the Adaptation Manager.

With errors during execution of the process fragments will be dealt in the same way as with errors in normal BPEL processes. This means, that they have to be considered during design time and handled accordantly, ether in the main process or a other process fragment. This does not decrease the flexibility of the process, since it is possible to dynamically adapt handers for specific process fragments. To model such a dynamic handling of errors, it is enough to place an abstract activity in the compensation handler or error handler[9]. When an error occurs during execution, the abstractActivity in the compensation handler is refined is such a way, that it meets the specific compensation needs of the particular process fragment throwing the error.

## 4.5. BPEL-Inject Example

In this section a example is provided for the use of the BPEL-Inject extension. To provide a overall picture of how the extension elements will look inside a BPEL process, all changes to a basic BPEL process model are described. Furthermore, the prototypical data model for the abstract activity is briefly described in this section. The scope of this thesis does not cover the realization of the mechanism selecting the process fragments. Nevertheless, a example data model is introduced, that could be used in future projects, to represent information needed for the selection process.

Each extension of the standard WS-BPEL 2.0 language that is used in a BPEL process model must be specified properly, so a BPEL engine executing the process can recognize and handle the extension accordingly. Below a specification for the BPEL-Inject extension is presented:

```
1 <bpel:process name="ExmapleProcess"
2   targetNamespace="http://exmaplenamespace"
3   suppressJoinFailure="yes"
4   xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
      executable"
5   xmlns:dain="http://www.allow-ensembles.eu/adaptation/
      abstractActivity">
6
```

[9]Both handlers are described in the fundamentals in section 2.1.2

```
7     <bpel:extensions>
8         <bpel:extension namespace="http://www.allow-ensembles.eu/
              adaptation/abstractActivity" mustUnderstand="yes"></
              bpel:extension>
9     </bpel:extensions>
10
11    <bpel:import namespace="http://www.allow-ensembles.eu/
          adaptation/abstractActivity" location="
          DaInjectAbstractActivitySchema.xsd"
12  importType="http://www.w3.org/2001/XMLSchema"></bpel:import>
13
14  <!-- Definition of partner links, message exchanges, variables,
          correlation sets, handlers, activities according to the WS-
        BPEL 2.0 specification and the BPEL-Inject extenstion. -->
15
16  ...
17
18  <bpel:process>
```

**Listing 4.2:** Specification of the BPEL-Inject extension inside a BPEL process

In line five the prefix "dain" for the namespace of the extension is defined. Line eight specifies that a BPEL extension the process model is used with the given namespace of the extension. The "mustUnderstand" attribute defines that a execution engine must support the extension in order to execute the process. Line eleven imports the data model definition for the <abstractActivity> element, representing the custom activity added by the BPEL-Inject extension. In this line the namespace for the import is specified, which obviously matches the namespace of the extension. Furthermore the location of the file containing the model definition is provided as an attribute and the type of the import is specified inside the importType attribute. Please note that the suppressJoinFailure attribute of the process (line three) is set to "yes". This is important for the right execution of the process, as it prevents multiple injection of process fragments, when the abstract activity is used as a join activity.

As mentioned before, a prototypical data model is used for the representation of possibly selection-relevant data inside the <abstractActivity> element. The model is based on the work presented in [BMP13]. In this work the state of the operational environment is represented by a collection of graph based models. Each object has a graph associated with it. The nodes of the graph represent the possible states of the object. The transitions between the notes have information logically attached to them, so transition conditions\triggers can be modelled. For example, the state of an object can change depending on an occurring event. Furthermore, goals can be represented as a collection of states of multiple objects, that need to be reached after the injected business logic is executed, in order to achieve the goal. The XML-Schema definition of the example data model is attached in the appendix A. Below a example of the <abstract activity> element is presented:

```
<bpel:extensionActivity>
    <dain:AbstractActivity name="ticketPayment" annotation="basic
       " securityConcerns="high">
      <dain:entities>
      <dain:Entity>
        <dain:entityName>PassengerA</dain:entityName>
        <dain:entityUUID>PA424</dain:entityUUID>
        <dain:entityLocationURL>http://flexibus.dain/passengers/
           pa424/location</dain:entityLocationURL>
        <dain:entityContextReferenceURL>http://flexibus.dain/
           passagers/pa424/entitycontextreference</dain:
           entityContextReferenceURL>
      </dain:Entity>
      <dain:Entity>
        <dain:entityName>Bus B1</dain:entityName>
        <dain:entityUUID>B1</dain:entityUUID>
        <dain:entityLocationURL>http://flexibus.dain/bus/B1/
           location</dain:entityLocationURL>
        <dain:entityContextReferenceURL>http://flexibus.dain/bus/
           B1/entitycontextreference</dain:
           entityContextReferenceURL>
      </dain:Entity>
    </dain:entities>

    <dain:preCondition>
      <dain:point>
        <dain:object o_name="Credit Card Terminal">
          <dain:state>Available</dain:state>
          <dain:state>Ready</dain:state>
        </dain:object>
        <dain:object o_name="Wireless Network Connection">
          <dain:state>Available</dain:state>
          <dain:state>Established</dain:state>
        </dain:object>
      </dain:point>
      <!--Other points -->
    </dain:preCondition>

    <dain:effect>
      <dain:object o_name="Bus Ticket">
        <dain:event>Bus ticket paid with Credit Card</dain:event>
      </dain:object>
      <dain:object o_name="Bonus points">
        <dain:event>Points granted</dain:event>
```

```
38        </dain:object>
39      </dain:effect>
40
41      <dain:goal>
42        <dain:object o_name="Bus Ticket">
43          <dain:state>Paid</dain:state>
44        </dain:object>
45      </dain:goal>
46
47      <dain:compensationGoal>
48        <dain:object o_name="Bus Ticket">
49        <dain:state>Awaiting payment</dain:state>
50        </dain:object>
51      </dain:compensationGoal>
52
53      </dain:AbstractActivity>
54  </bpel:extensionActivity>
```

**Listing 4.3:** Example of an <abstractActivity>.

Listing 4.3 presents an abstract activity that models the credit card payment for a Flexibus ticket. The abstract activity is part of a larger process describing the purchase of Flexibus tickets. The abstract activity introduced in this thesis contains five main elements:

- The <entities> element contains information about the entities participating in the executing process. In this case entities are the bus where the ticket is bought and the passenger buying the ticket.

- The <preCondition> element defines states, that the specified objects must be in, so that the injected business logic can be executed successfully without further adaptation. In the example two preconditions are modelled: Firstly, the bus must have a functioning credit card terminal on board. Secondly, a working wireless network connection must be available in order to successfully pay with the credit card.

- The <effect> element specifies which objects should be effected by the injected process logic. The effects are described as events, that trigger the transition of objects from one state to an other. In our example the bus ticket must be paid, so respectfully the object representing the bus ticket must change is state form "Awaiting payment" to "Paid". Also other objects can be effected. One example are bonus points that are granted for some payment methods. This is represented accordingly in the <effect> element.

- The <goal> element is the most important element in the <abstractActivity> element and it is the only one that is not optional. The <goal> element specifies states of multiple objects, that have to be reached after the injected business logic was executed. The <goal> in the presented example simply described, that the ticket must be paid.

- The <compensationGoal> specifies goal that should be achieved, if the default goal cannot be reached. This allows to specify that further process logic should be injected and executed, in order to return to a conclusive execution state of the process. From this state, other approaches can be to successfully execute the process. In our example the specified <compensationGoal> ensures, that the ticket does not accidentally change it's state to "Paid" when issues with the payment occur.

Beside the above described elements, the abstract activity has two attributes. The "securityConcerns" attribute are used to specify that the information handled with the injected business logic is sensible and requires additional security policies. The "annotation" attribute defines the annotation type used inside the <abstractActivity>.

# 5. Implementation

In this chapter the architectural solution and the realization of the system extensions enabling process fragment injection is presented, with respect to the requirements and concepts specified in chapter 4. First the solution of the Modelling Environment is described, including its architecture and extension. As the focus of this thesis lays on the orchestration engine and its extension regarding process fragment injection, in section 5.2 the architectural solution for the engine extension is presented as well as the description of the components and the injection approach.

## 5.1. Modelling Environment

In this section the architectural and technical approach fo the Modelling Environment is explained. First the architecture of the Eclipse BPEL Designer is described. Subsequently the design for the extension and its realization is presented.

### 5.1.1. Architectural Overview

The modelling environment developed in scope of this thesis is based on Eclipse BPEL Designer. In order to extend the software with the support for abstract activities the respective functionality must be added in a way consistent with the architecture of the Eclipse BPEL Designer. Below a description of the architecture of the designer is provided, based on the work presented in [Fer07, Höh08].

As figure 5.1 shows the architecture of Eclipse BPEL Designer is divided in five main components each encapsulating one of the main functions of the designer, described in section 2.3.2. Each component contains one or two java-packages. To add the functionality required for this thesis the package org.eclipse.bpel.model of the Model component and the org.eclipse.bpel.ui package of GUI component are extended. To provide better understandability of the provided extension and how they work, the two extended packages are described in more detail.

#### org.eclipse.bpel.model

The org.eclipse.bpel.model package is important for this thesis, as it contains the EMF-model representing the constructs of the BPEL language and the generated classes providing java-object-representations of those constructs. Furthermore the package provides classes used for the automatic translation of the java-objects to their respective form inside a bpel-file.

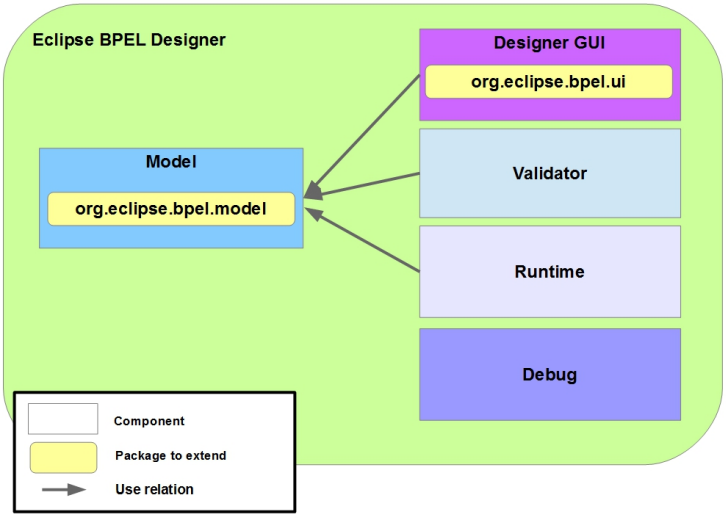


**Figure 5.1.:** Eclipse BPEL Designer architecture[Höh08]

This translation is called serialisation. The opposite translation from the BPEL language constructs to the java-objects in referred to as deserialisation. The BPELReader class and the BPELWriter class are responsible for the deserialisation and serialisation respectfully. During the transformation the Document Object Model(DOM)[1] is used as a mediate format. Each java-object describing BPEL objects has a DOM-Node logically attached to it, which represents the object. The BPELWriter creates the DOM-representations of the java-objects. Those DOM-representations are passed to a DOM-parser which generates the respective XML-elements inside the bpel-file. Analogically, the BPELReader handles the deserialisation by creating java-objects for the DOM-representations form the parser. An other important class in the model package is the ReconciliationHelper class, which is responsible for updating the DOM-representation, when its java-object is modified.

#### org.eclipse.bpel.ui

The design of Eclipse BPEL Designer is based on the model-view-controller concept. The org.eclipse.bpel.ui provides the graphical components for the GEF-based BPEL-Editor and also contains some control logic. It contains the graphical representations of the BPEL constructs as well as dialogues, wizards, property windows together with helper classes and classes required by the eclipse framework. This package uses the model provided in the org.eclipse.bpel.model package. Each graphical object in the graphical process editor is associated with an EditPart. This provides a connection between the graphical object and the model element is represents. For each graphical element an adapter class is provided. This class describes which factories provide graphical objects(e.g. icons) for the graphical representation. The adapter also allows to specify additional behavior for an EditPart, for

[1] W3C, Document Object Model: `http://www.w3.org/DOM/`

example that a wizard is opened, when an new object is added to the graphical process model.

### 5.1.2. BPEL-Designer extension for BPEL-Inject

As mentioned in the fundamentals, the Eclipse BPEL Designer provides various ways for extension. To extend the designer with the support for a new custom activity, the org.eclipse.bpel.ui package must be extended with the necessary graphical elements and a model for the activity must be provided that are consistent with the model provided within the org.eclipse.bpel.model package. To add new graphical elements to the Eclipse BPEL designer provides multiple extension points[Fou05]. For the extension developed in this thesis the org.eclipse.bpel.common.ui.paletteAdditions extension point is used to add the new abstract activity element to the pallet of available BPEL constructs. Furthermore the org.eclipse.bpel.ui.uiObjectFactories is used to define the factory for the graphical objects.

#### Model (de.uni.stutt.da.inject.model)

In order to provide support for the BPEL-Inject extension inside the Eclipse BPEL Designer, the EMF-Model was created based on the abstract activity model described in section 4.5:

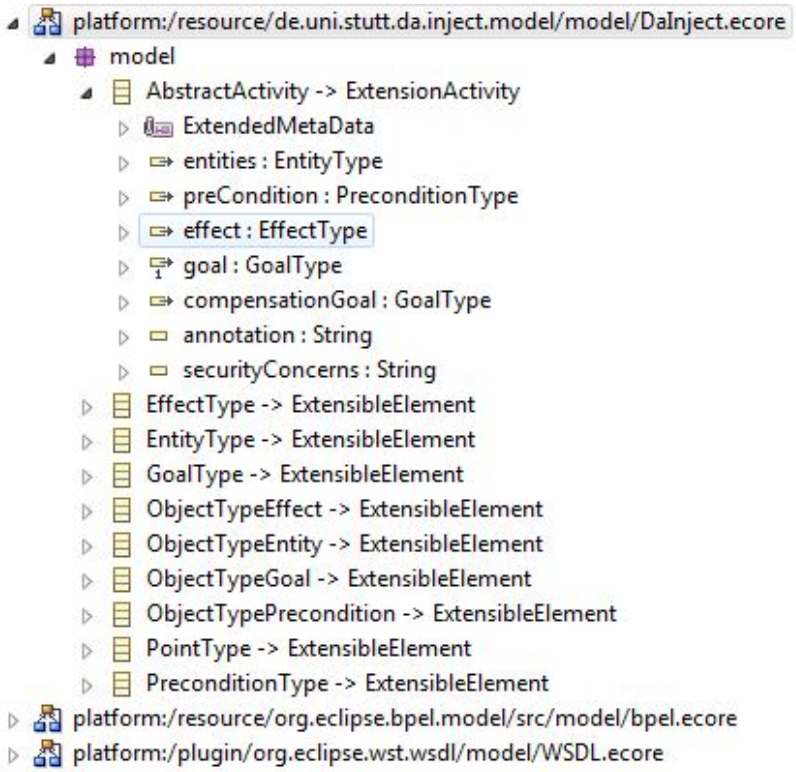


**Figure 5.2.:** EMF-Model of the abstract activity

The Eclipse Modelling Framework provides the means to generate classes for the modelled objects. Nevertheless the generated code was additionally modified to ensure the right reconciliation support. To enable the serialisation and deserialisation of the abstract activity the DaInjectExtensionsSerialiser and DaInjectExtensionsDeserialiser classes are created. Listing 5.1 presents the method for the deserialisation of the <goal> element of the <abstractActivity>.

```
1  GoalType toGoalType(Element goalElement) {
2      if (hasChild(goalElement)) {
3        GoalType result = ModelFactory.eINSTANCE.createGoalType();
4        Element objectElement = (Element) goalElement.getChildNodes
             ().item(0);
5
6        ObjectTypeGoal goal = ModelFactory.eINSTANCE.
             createObjectTypeGoal();
7
8        // OName Attribute
9        String attOName_Name = ModelPackage.eINSTANCE.
             getObjectTypeGoal_OName().getName();
10       if (objectElement.getAttribute(attOName_Name) != null) {
11         goal.setOName(objectElement.getAttribute(attOName_Name));
12       }
13
14       // States elements
15       if (hasChild(objectElement)) {
16         NodeList statesDOMNodes = objectElement.getChildNodes();
17
18         for (int i = 0; i < statesDOMNodes.getLength(); i++) {
19           goal.getState().add(statesDOMNodes.item(i).getNodeValue
                 ());
20         }
21       }
22       result.setObject(goal);
23       return result;
24     }
25     return null;
26   }
```

**Listing 5.1:** Method for the deserialisation of the <goal> element.

Furthermore, the created serialization and deserialization classes must be integrated with the internal serialization mechanism of the Eclipse BPEL Designer. For this purpose the designer provides the BPELExtensionRegistry class which allows to register custom serializers and deserializers which are invoked, when an instance of the ExtensionActivity class is recognized.

The ExtensionActivity class represents the BPEL's <extensionActivity>. The registration of the DaInjectExtensionsSerialiser and DaInjectExtensionsDeserialiser is presented in the listing below:

```
1  private static void registerSerializerAndDeserializer() {
2    // Initialize BPEL extension registry
3    BPELExtensionRegistry extensionRegistry = BPELExtensionRegistry
        .getInstance();
4
5    // Initialize our own serializers and deserializers
6    DaInjectExtensionsDeserializer deserializer = new
        DaInjectExtensionsDeserializer();
7    DaInjectExtensionsSerializer serializer = new
        DaInjectExtensionsSerializer();
8
9    // AbstractActivity
10   String name = AbstractActivity.class.getSimpleName();
11   extensionRegistry.registerActivityDeserializer(new QName(
        ModelPackage.eNS_URI, "AbstractActivity"),
12       deserializer);
13   extensionRegistry.registerActivitySerializer(new QName(
        ModelPackage.eNS_URI, name),
14       serializer);
15   }
```

**Listing 5.2:** Registration of the serializer and deserializer classes for the abstract activity.

#### GUI Extension (de.uni.stutt.da.inject.ui)

To extent the graphical editor with support for the BPEL-Inject extension, graphical components are added, for the representation of the abstract activity, and new property pages to edit information inside the elements of the abstract activity.

As mentioned in section 5.1.1 the Eclipse BPEL Designer provides various extension points to add elements to the editor. In order to add the new components, a new eclipse plug-in is created that makes contributions to the user interface and uses extension points to integrate with the Eclipse BPEL Designer. First the necessary factory and adapter classes are created. The DaInjectExtensionsUIObjectFactory provides the graphical objects (icons) for the representation inside the editor. This factory is integrated by using the org.eclipse.bpel.ui.uiObjectFactories extension point:

```
1  <extension id="DaInjectExtensionsUIObjectFactory"
2             name="DaInject"
```

```
3            point="org.eclipse.bpel.ui.uiObjectFactories">
4            <factory categoryId="not.used"
5                    class="de.uni.stutt.da.inject.ui.factories.
                        DaInjectExtensionsUIObjectFactory"
6                    id="de.uni.stutt.da.inject.ui.
                        DaInjectExtensionsUIObjectFactory"
7                    specCompliant="false">
8          </factory>
9  </extension>
```

**Listing 5.3:** Adding the DaInjectExtensionsUIObjectFactory to the BPEL Designer.

The DaInjectExtensionsUIAdapterFactory extends the ModelAdapterFactory class from the previously created model of the abstract activity. This step integrates the model with the graphical plug-in. For the adapter class the default ActivityAdapter is reused. It is extended inside the AbstractActivityAdapter class. The DaInjectExtensionsUIAdapterFactory returns an instances of the AbstractActivityAdapter class, as the adapter for the abstract activity. The factory must be registered in order to be used in the BPEL Designer. The listing below shows its registration:

```
1  BPELUtil.registerAdapterFactory(ModelPackage.eINSTANCE,
2        new DaInjectExtensionsUIAdapterFactory());
```

**Listing 5.4:** Registration of theDaInjectExtensionsUIAdapterFactory.

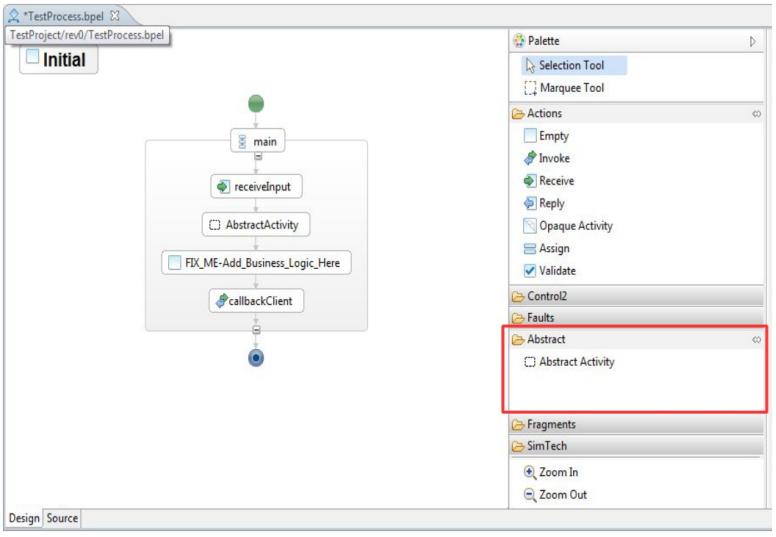


**Figure 5.3.:** Pallet extension

To add the abstract activity to the pallet of available BPEL constructs, the extension point org.eclipse.bpel.common.ui.paletteAdditions is used. This extension point allows to specify

classes contributing to the to the pallet. The realization of such a class is the DaInjectPaletteProvider class, which introduces a new category to the pallet and adds the abstract activity inside the new category(see figure 5.3). The DaInjectExtensionsUIObjectFactory provides the graphical representation for the added abstract activity element. For the data model representation the abstract activity java-object is taken from the de.uni.stutt.da.inject.model data model described above.

In order to enable the user to modify the content of the abstract activity, appropriate GUI-elements must be provided. The Eclipse BPEL Desinger provides the property view where the properties of selected elements are displayed. The property view contains multiple tabs. New property tabs can be added to the view, by extending the org.eclipse.ui.views.properties.tabbed.propertyTabs extension point. In total six property tabs (marked in figure 5.4) are introduced with the extension from this thesis. The abstract activity tab provides an overview over the contents of the selected abstract activity. Each of the remaining tabs displays one of the main components of the abstract activity. Each tab can contain multiple sections, which are automatically added or removed by the designer depending on the java-object type associated with the selected graphical element. Sections can be described as groups of graphical elements (e.g, labels, text fields, buttons) that together represent a particular element or attribute. To add sections the Eclipse BPEL Designer provides the org.eclipse.ui.views.properties.tabbed.propertySections extension point. The tabs are also made visible depending on the java-object type of the selected object. On figure 5.4 the property tab for the pre-condition element is selected to demonstrate the composition of the realized property views. The views are essentially lists representing the elements contained inside the selected property. The user can modify the information inside the elements, add new sub-elements(e.g. states) or remove them.

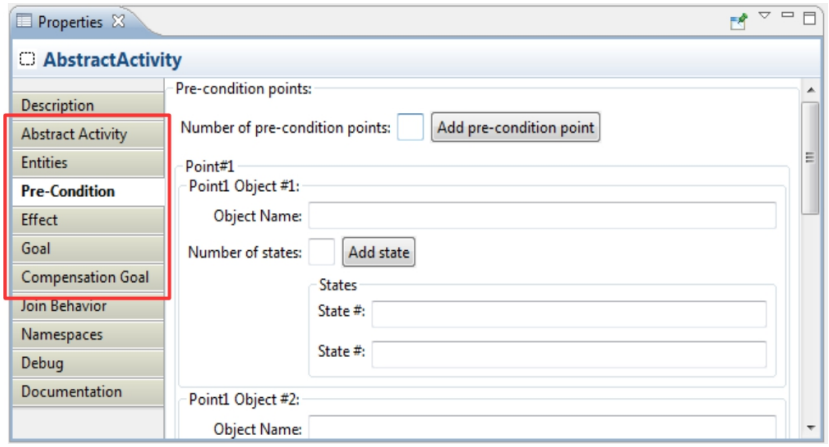


**Figure 5.4.:** Extended tabbed properties

#### Difficulties during realization

The data model of the <abstractActivity> element is relatively large and complex. To provide a user-friendly way to modify the content of the abstract activity an intuitional presentation of the content must be provided. The main elements of the abstract activity can contain a

large number of sub-elements (e.g states), so the content of the abstract activity is divided and presented in six separate tabs. During the realization of the property view it becomes clear, that the Eclipse BPEL Designer only provides limited support for complex types inside custom activities. Both in terms of automatic serialization\deserialization and in the GUI when selecting an element, only the custom activity type are considered. This results in issues on handling the complex types inside the abstract activity. To provide a workaround for this, both problems had to be solved without the internal mechanisms provides ba the Eclipse BPEL Designer.

To display the required sections and tabs for the abstract activity, all sections are associated with the abstract activity class but placed inside different tabs:

```
1  <extension point="org.eclipse.ui.views.properties.tabbed.
       propertySections">
2     <propertySections contributorId="org.eclipse.bpel.ui.
         bpeleditor">
3        <propertySection
4          id="de.uni.stutt.da.inject.ui.sections.
             AbstractActivityPropertySection"
5          class="de.uni.stutt.da.inject.ui.sections.
             AbstractActivityPropertySection"
6          tab="de.uni.stutt.da.inject.ui.
             AbstractActivityPropertyTab">
7             <input
8                   type="de.uni.stutt.da.inject.model.model.
                       AbstractActivity">
9             </input>
10         </propertySection>
11         <propertySection
12               class="de.uni.stutt.da.inject.ui.sections.
                    EntityPropertySection"
13               id="de.uni.stutt.da.inject.ui.sections.
                    EntityPropertySection"
14               tab="de.uni.stutt.da.inject.ui.EntityPropertyTab">
15            <input
16                  type="de.uni.stutt.da.inject.model.model.
                       AbstractActivity">
17            </input>
18         </propertySection>
19         <propertySection
20               class="de.uni.stutt.da.inject.ui.sections.
                    PreConditionPropertySection"
21               id="de.uni.stutt.da.inject.ui.sections.
                    PreConditionPropertySection"
22               tab="de.uni.stutt.da.inject.ui.
```

```
                PreConditionPropertyTab">
23          <input
24                type="de.uni.stutt.da.inject.model.model.
                    AbstractActivity">
25          </input>
26        </propertySection>
27
28    ...
29
30   </propertySections>
31 </extension>
```

**Listing 5.5:** Registration of the sections for the abstract activity.

For each used section the data reading as well as the composition of the graphical elements(labels, text fields, etc.) is realized independently of the mechanism of the BPEL Designer. The control logic for this is realized inside the section classes. Furthermore, the serialization\deserialization of the complex elements of the abstract activity couldn't be realized by using the Designer's internal mechanism. So the main part of the serialization\deserialization must been realized inside the GUI extension. The following listing presents a fragment of the serialisation of the <entities> element:

```
1  btnAddEntityButton.addMouseListener(new MouseListener() {
2    @Override
3    public void mouseUp(MouseEvent e) {
4
5      EntityType entities = null;
6      if (aactivity.getEntities() == null) {
7        entities = ModelFactory.eINSTANCE.createEntityType();
8      } else {
9        entities = aactivity.getEntities();
10     }
11
12     ObjectTypeEntity entType = ModelFactory.eINSTANCE
13       .createObjectTypeEntity();
14     entities.getEntity().add(entType);
15
16     Document document = aactivity.getElement().getOwnerDocument()
           ;
17
18     Element allEntitiesDOMElement = document.createElementNS(
19         aactivity.getElement().getNamespaceURI(),
20         ModelPackage.eINSTANCE.getAbstractActivity_Entities()
21             .getName());
```

```
22      allEntitiesDOMElement.setPrefix(ModelPackage.eINSTANCE
23          .getNsPrefix());
24
25      Element entityDOMElement = document.createElementNS(aactivity
26          .getElement().getNamespaceURI(), ModelPackage.eINSTANCE
27          .getEntityType_Entity().getName());
28      entityDOMElement.setPrefix(ModelPackage.eINSTANCE.getNsPrefix
            ());
29
30      allEntitiesDOMElement.appendChild(entityDOMElement);
31
32      entities.setElement(allEntitiesDOMElement);
33    }
34  }
```

**Listing 5.6:** Sersialisation of the entitiy element.

## 5.2. Execution Engine

As described in chapter 4 multiple aspects must be covered, in order to add injection support to Apache ODE. In this section the necessary extensions to the engine are described. First an architectural overview of the engine extension is provided and the added components are described. Further the realization approach for the injection is presented.

### 5.2.1. Architectural overview of the engine extension

Figure 5.5 presents the Apache ODE extended with components for enabling process fragment injection. Almost every layer of the engine is effected by the adaptation. The core of the extension is the Injection Runtime. It is responsible for the coordination of the injection process. When an abstract activity is recognized, the execution of the process instance must be suspended. In order to achieve this, the Injection runtime interacts with the Process and Instance Management component. For the retrieval of the fragment choice and the process fragment itself, the Workflow API is extended with new interfaces. The JBI Integration Layer is also extended in order to provide the necessary connectivity. The extension of the JBI Integration layer is divided in two integration components. One is the Inject Adaptation Management integration component, which provides the means to communicate with the external Adaptation Manager, in order to acquire a selection of the process fragment intended for injection. The second integration component is the Inject Fragment Management component which enables the interaction with the Fragmento repository, so the required process fragments can be obtained. Both integration components use web services for communication.

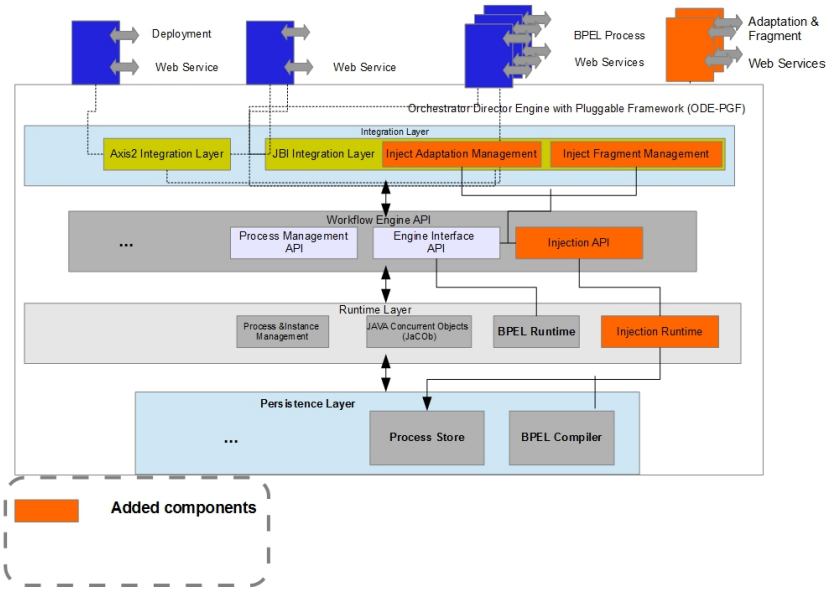


**Figure 5.5.:** Extension of Apache ODE for injection support.

For the compilation of the received process fragments the BPEL Compiler from the Persistence Layer is used. When the ODE-internal representation of the BPEL objects are available, there are injected in in the process instance.

### 5.2.2. Injection concept

The realization approach for the injection concept is inspired by the approach used in the Mayflower project (see section 3.3.2). When the injection needs to be performed, the compiled constructs of the chosen process fragment must be inserted in the place of the abstract activity. In order to achieve this the internal execution queue inside Apache ODE is correctly and consistently adapted. The queue holds a list of all activity instances scheduled for execution as part of a particular process instance and a list of completed activities. The execution queue object contains also a list of channels used for communication between activities. To perform the injection, these three lists are modified, depending in the activity in which the abstract activity where placed. For example if the abstract activity was placed inside sequence activity, than the object representing the sequence needs to modified and the injected business logic must be placed inside the sequence activity. For each structured activity type an appropriate adaptation must be performed. The injected process fragments are encapsulated in scope activities. The defined event handlers must be initialized prior to the execution. This is handled inside the Injection runtime.

### 5.2.3. Validation

The validation of injected process logic is not trivial. Due the fact, that no functioning implementation could yet be created, it could not be determined, if the Pluggable Framework generates events of injected business logic. Depending on the outcome, ether this or other monitoring solutions will be used for the validation. Also the validation by monitoring the behavior of the orchestrated web services is considered.

# 6. Outcome and Future Work

In the scope of this thesis an general overview on existing approaches for enabling process fragment injection was provided. Requirements were analyzed for the language as well as the execution environment and possible constrains were identified. Furthermore, a concept for a framework enabling process fragment injection was presented and the formalisation for the BPEL language extension was provided. Also a prototypical data model for the introduced abstract activity was created and described with an example. A description of concepts and technologies necessary for the realization of the injection framework were presented. Also architectural solutions for the realisation of the Modelling Environment and Execution Engine were presented and details of their implementation were provided.

Future project can extent the work of this thesis to provide further adaptability of business processes. For example by realizing the adaptation strategies mentioned in section 2.2.

# Appendix A.

# Abstract Activity Schema

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <schema targetNamespace="http://www.allow-ensembles.eu/adaptation
      /abstractActivity" elementFormDefault="qualified" xmlns="http
      ://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.allow-
      ensembles.eu/adaptation/abstractActivity">
3
4  <!--
5    BPEL namespaces have to be included, also the standart element
        and attributes in the abstract activity.
6    extension namespace= http://www.allow-ensembles.eu/adaptation/
        abstractActivity
7   -->
8
9      <complexType name="Entity">
10       <sequence>
11         <element name="entityName" type="string" minOccurs="0"
12           maxOccurs="1">
13         </element>
14         <element name="entityUUID" type="string" minOccurs="0"
15           maxOccurs="1">
16         </element>
17             <element name="entityLocationURL" type="string"
                 minOccurs="0" maxOccurs="1"></element>
18             <element name="entityContextReferenceURL" type="
                 string" minOccurs="0" maxOccurs="1">
19         </element>
20         </sequence>
21     </complexType>
22
23   <!-- ##### Imported and adapted from the CAptLang specification
        for defining preconditions and effects ##### -->
24
25   <complexType name="ObjectTypePrecondition">
26     <sequence>
```

```
27        <element name="state" type="NMTOKEN" minOccurs="1"
             maxOccurs="unbounded"></element>
28      </sequence>
29      <attribute name="o_name" type="NMTOKEN" use="required"></
           attribute>
30    </complexType>
31
32    <complexType name="ObjectTypeEffect">
33      <sequence>
34        <element name="event" type="NMTOKEN" minOccurs="1"
             maxOccurs="unbounded"></element>
35      </sequence>
36      <attribute name="o_name" type="NMTOKEN" use="required"></
           attribute>
37    </complexType>
38
39    <complexType name="PointType">
40        <sequence>
41          <element name="object" minOccurs="1" maxOccurs="unbounded
               " type="tns:ObjectTypePrecondition"/>
42          </sequence>
43      </complexType>
44
45    <complexType name="PreconditionType">
46      <sequence>
47        <element name="point" minOccurs="1" maxOccurs="unbounded"
             type="tns:PointType"/>
48      </sequence>
49    </complexType>
50
51    <complexType name="EffectType">
52      <sequence>
53        <element name="object" minOccurs="1" maxOccurs="unbounded"
             type="tns:ObjectTypeEffect"/>
54      </sequence>
55    </complexType>
56
57    <!-- ########################### -->
58
59    <complexType name="AbstractActivity">
60      <!-- might want to include some other info here in attributes
61      BPEL- Standart elements and attributes has to be included
62      -->
63      <sequence>
```

```
64        <element name="entities" type="tns:Entity" minOccurs="0"
65          maxOccurs="unbounded">
66        </element>
67        <element name="preCondition" type="tns:PreconditionType"
68          minOccurs="0" maxOccurs="1">
69        </element>
70        <element name="effect" type="tns:EffectType" minOccurs="0"
71          maxOccurs="1">
72        </element>
73      </sequence>
74      <attribute name="annotation" type="string"></attribute>
75      <attribute name="securityConcerns" type="string"></attribute>
76    </complexType>
77
78    <element name="AbstractAdaptationActivity" type="tns:
        AbstractActivity"></element>
79  </schema>
```

**Listing A.1:** XML-Schema definiton for the <absrtactActivity>

# Bibliography


[ABS+13] V. Andrikopoulos, A. Bucchiarone, S. G. Sáez, D. Karastoyanova, and C. A. Mezzina. Towards Modeling and Execution of Collective Adaptive Systems. In *Proceedings of the 9th International Workshop on Engineering Service-Oriented Applications (WESOA'13)*, pages 1–12. Springer, Dezember 2013.

[All12] O. Alliance. OSGi Release 5. http://www.osgi.org/Specifications/HomePage, 2012.

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

[Apab] Apache. ServiceMix 4.3. http://servicemix.apache.org.

[BLMP10] A. Bucchiarone, A. L. Lafuente, A. Marconi, and M. Pistore. A formalisation of adaptable pervasive flows. In *Proceedings of the 6th international conference on Web services and formal methods*, WS-FM'09, pages 61–75, Berlin, Heidelberg, 2010. Springer-Verlag.

[BMP13] A. Bucchiarone, C. A. Mezzina, and M. Pistore. CAptLang: A Language for Context-aware and Adaptable Business Processes. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, pages 12:1–12:5, New York, NY, USA, 2013. ACM.

[BMPR12] A. Bucchiarone, A. Marconi, M. Pistore, and H. Raik. Dynamic Adaptation of Fragment-Based and Context-Aware Business Processes. In *Web Services (ICWS), 2012 IEEE 19th International Conference on*, pages 33–41, 2012.

[Cor05] O. Corporation. JSR-000208 Java Business Integration 1.0 (Final Release). https://jcp.org/aboutJava/communityprocess/final/jsr208/, 2005.

[DK] A. B. Dimka Karastoyanova. ReFFlow - Reusable Flexible WS Reusable Flexible WS-flows. http://www-old.dvs.informatik.tu-darmstadt.de/research/refflow/.

[DS] D. D. M. H. M. S. David Schumm, Tobias Anstett. Fragmento. http://www.iaas.uni-stuttgart.de/forschung/projects/fragmento/start.htm.

[DVdATH05] M. Dumas, W. M. Van der Aalst, and A. H. Ter Hofstede. *Process-aware information systems: bridging people and software through process technology*. Wiley-Interscience, 2005.

[Fer07] J. V. Fernandez. BPEL with Explicit Data Flow: Model, Editor, and Partitioning Tool. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Juli 2007.

[Fon09] C. Fonden. Konzeption und Entwicklung von Kontexterweiterungen für Workflows. Diplomarbeit, Universität Stuttgart : Sonderforschungsbereich SFB 627 (Nexus: Umgebungsmodelle für mobile kontextbezogene Systeme), Germany, August 2009.

[Foua] A. S. Foundation. Apache ODE Architectural Overview. http://ode.apache.org/developerguide/architectural-overview.html.

[Foub] A. S. Foundation. JaCOb - Virtual Processing Unit. http://ode.apache.org/developerguide/jacob.html.

[Fouc] T. E. Foundation. BPEL Designer Project. http://eclipse.org/bpel/.

[Foud] T. E. Foundation. Eclipse Modeling Framework Project (EMF). https://www.eclipse.org/modeling/emf/.

[Foue] T. E. Foundation. GEF (Graphical Editing Framework). http://www.eclipse.org/gef/.

[Fou05] T. E. Foundation. BPEL Project Extension Points. http://eclipse.org/bpel/developers/extension_points.php, December 2005.

[GBH+05] M. Grossmann, M. Bauer, N. Honle, U.-P. Kappeler, D. Nicklas, and T. Schwarz. Efficiently Managing Context Information for Large-Scale Scenarios. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pages 331–340, March 2005.

[Gro] T. O. Group. Service Oriented Architecture : What Is SOA? http://www.opengroup.org/soa/source-book/soa/soa.htm.

[Gro04] W. W. Group. Web Services Architecture. http://www.w3.org/TR/ws-arch, February 2004.

[Hah13] M. Hahn. Approach and Realization of a Multi-tenant Service Composition Engine. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, November 2013.

[Höh08] B. Höhensteiger. Konzipierung und Implementierung eines BPEL light Editors mit Unterstützung für Message Exchange Patterns. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Oktober 2008.

[HRKD08] K. Herrmann, K. Rothermel, G. Kortuem, and N. Dulay. Adaptable Pervasive Flows - An Emerging Technology for Pervasive Adaptation. In *Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on*, pages 108–113, 2008.

[IA] IBM and S. AG. WS-BPEL Extension for Sub-Processes (BPEL-SPE). http://xml.coverpages.org/ni2005-10-13-a.html#overview, October 13, 2005.

[Jr.07] S. O. Jr. Getting on Board the Enterprise Service Bus. *Computer*, 40:15–17, April 2007.

[KB04] D. Karastoyanova and A. Buchmann. Extending web service flow models to provide for adaptability. In *Proceedings of OOPSLA*, volume 4, 2004.

[KHC+05] D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann, and A. Buchmann. Extending BPEL for run time adaptability. In *EDOC Enterprise Computing Conference, 2005 Ninth IEEE International*, pages 15–26, Sept 2005.

[KHK+11] 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.

[KKL+05] M. Kloppmann, D. König, F. Leymann, G. Pfau, A. Rickayzen, C. Von Riegen, P. Schmidt, and I. Trickovic. WS-BPEL extension for sub-processes–BPEL-SPE. *Joint white paper, IBM and SAP*, page 49, 2005.

[KML08] O. Kopp, R. Mietzner, and F. Leymann. Abstract syntax of WS-BPEL 2.0, 2008.

[Mic02] S. Microsystems. Core J2EE Patterns - Data Access Object:. http://www.oracle.com/technetwork/java/dataaccessobject-138824.html, 2001-2002.

[MPS+09] A. Marconi, M. Pistore, A. Sirbu, F. Leymann, H. Eberle, and T. Unger. Enabling Adaptation of Pervasive Flows: Built-in Contextual Adaptation. In L. Baresi, C.-H. Chi, and J. Suzuki, editors, *Service-Oriented Computing, 7th International Joint Conference,ICSOC-ServiceWave 2009, Stockholm, Sweden, November 24-27, 2009*, volume 5900 of *Lecture Notes in Computer Science*, pages 445–454. Springer, November 2009.

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

[OK] D. K. R. K. F. L. M. P. R. S. M. S. T. S. S. S. M. W. Oliver Kopp, Michael Hahn. Apache ODE with Pluggable Framework Support. http://www.iaas.uni-stuttgart.de/forschung/projects/ODE-PGF/.

[PDMRa] P. D. P. D. Prof. Dr. Manfred Reichert. ADEPT1 - Flexible Workflow Management. http://www.uni-ulm.de/in/iui-dbis/forschung/projekte/abgeschlossene-projekte/adept1.html.

[PDMRb] P. D. P. D. Prof. Dr. Manfred Reichert. ADEPT2. http://www.uni-ulm.de/in/iui-dbis/forschung/projekte/abgeschlossene-projekte/adept2.html.

[Rai12] H. Raik. *Service Composition in Dynamic Environments: From Theory to Practice*. PhD thesis, University of Trento, Italy, 2012.

[RBK+12] H. Raik, A. Bucchiarone, N. Khurshid, A. Marconi, and M. Pistore. ASTRO-CAptEvo: Dynamic Context-Aware Adaptation for Service-Based Systems. In *Services (SERVICES), 2012 IEEE Eighth World Congress on*, pages 385–392, June 2012.

[RD09] M. Reichert and P. Dadam. Enabling Adaptive Process-aware Information Systems with ADEPT2. In J. Cardoso and W. van der Aalst, editors, *Handbook of Research on Business Process Modeling*, pages 173–203. Information Science Reference, Hershey, New York, March 2009.

[Rin04] S. Rinderle. Schema Evolution in Process Management Systems. 2004.

[Sat01] M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, 2001.

[SDH+12] D. Schumm, D. Dentsas, M. Hahn, D. Karastoyanova, F. Leymann, and M. Sonntaga. Web Service Composition Reuse through Shared Process Fragment Libraries. In *Proceedings of the 12th International Conference on Web Engineering (ICWE 2012 Demos), Berlin, Germany*, Lecture Notes in Computer Science. Springer-Verlag, 2012.

[SHK12] M. Sonntag, M. Hahn, and D. Karastoyanova. Mayflower - Explorative Modeling of Scientific Workflows with BPEL. In *Proceedings of the Demo Track of the 10th International Conference on Business Process Management (BPM 2012), CEUR Workshop Proceedings, 2012*, pages 1–5. CEUR Workshop Proceedings, September 2012.

[SK10] M. Sonntag and D. Karastoyanova. Next Generation Interactive Scientific Experimenting Based On The Workflow Technology. In R. Alhajj, V. Leung, M. Saif, and R. Thring, editors, *Proceedings of the 21st IASTED International Conference on Modelling and Simulation (MS 2010), 2010.* ACTA Press, Juli 2010.

[SK12] M. Sonntag and D. Karastoyanova. Ad hoc Iteration and Re-execution of Activities in Workflows. *International Journal On Advances in Software*, 5(1 & 2):91–109, Juli 2012.

[SKLS10] D. Schumm, D. Karastoyanova, F. Leymann, and S. Strauch. Fragmento: Advanced Process Fragment Library. In *Proceedings of the 19th International Conference on Information Systems Development (ISD 2010), 25 August 2010, Prague, Czech Republic*. Springer-Verlag, 2010.

[Ste08] T. Steinmetz. Ein Event-Modell für WS-BPEL 2.0 und dessen Realisierung in Apache ODE. Diploma thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, August 2008.

[TZ11] S. Tragatschnig and U. Zdun. Runtime Process Adaptation for BPEL Process Execution Engines. In *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011 15th IEEE International*, pages 155–163, Aug 2011.

[vLLN11] T. van Lessen, D. Lübke, and J. Nitzsche. *Geschäftsprozesse automatisieren mit BPEL*. dpunkt Verlag, Heidelberg, January 2011.

[W3C07] W3C. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). http://www.w3.org/TR/soap12-part1/, April 2007.

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[Wei01] M. Weiser. What is Pervasive Computing? 2001.

[WKNL07] M. Wieland, O. Kopp, D. Nicklas, and F. Leymann. Towards Context-Aware Workflows. In B. Pernici and J. A. Gulla, editors, *CAiSE 07 Proceedings of the Workshops and Doctoral Consortium Vol.2, Trondheim, Norway, June 11-15th, 2007*, pages 577–591. Tapir Acasemic Press, Juni 2007.

[WRRM08] B. Weber, M. Reichert, and S. Rinderle-Ma. Change patterns and change support features Enhancing flexibility in process-aware information systems. *Data & Knowledge Engineering*, 66(3):438 – 466, 2008.

All links were last followed on March 19, 2014

## Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, March 19, 2014 _____

(Name)