Institute of Architecture of Application Systems

Process Fragment Libraries for Easier and Faster Development of Process-based Applications

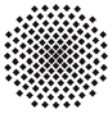
David Schumm, Dimka Karastoyanova, Oliver Kopp,
Frank Leymann, Mirko Sonntag, Steve Strauch

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

BibTeX:

```
@article {ART-2011-02,
    author = {David Schumm and Dimka Karastoyanova and Oliver Kopp and Frank
              Leymann and Mirko Sonntag and Steve Strauch},
    title = {{Process Fragment Libraries for Easier and Faster Development of
              Process-based Applications}},
    journal = {Journal of Systems Integration},
    volume = {2},
    number = {1},
    pages = {39--55},
    year = {2011},
    issn = {1804-2724}
}
```


The original publication is available at:
http://www.si-journal.org/index.php/JSI/article/view/83



Universität Stuttgart
Germany

Process Fragment Libraries for Easier and Faster Development of Process-based Applications

David Schumm, Dimka Karastoyanova, Oliver Kopp,
Frank Leymann, Mirko Sonntag, Steve Strauch
Institute of Architecture of Application Systems, University of Stuttgart,
Universitätsstraße 38, Stuttgart, Germany.
lastname@iaas.uni-stuttgart.de

Abstract: *The term "process fragment" is recently gaining momentum in business process management research. We understand a process fragment as a connected and reusable process structure, which has relaxed completeness and consistency criteria compared to executable processes. We claim that process fragments allow for an easier and faster development of process-based applications. As evidence to this claim we present a process fragment concept and show a sample collection of concrete, real-world process fragments. We present advanced application scenarios for using such fragments in development of process-based applications. Process fragments are typically managed in a repository, forming a process fragment library. On top of a process fragment library from previous work, we discuss the potential impact of using process fragment libraries in cross-enterprise collaboration and application integration.*

Key words: Process Fragment, Process Design, Reusability, Process Library.

1. Introduction

Several concepts have been proposed in the field of process-based application development to provide different granularities of reusable process artifacts. The atomic entities in process design are language constructs like activities, control and data connectors, routing gateways, business rules, and variables. Sub-process is the established reuse concept with larger granularity. A sub-process represents a self-contained and functionally complete artifact for process design and execution [12]. Process variants [9] and configurable process models [34] represent the largest units for reuse. These artifacts enable reusability and customizability of whole processes. The reuse of parts of a process is not covered by any of the above mentioned concepts.

We argue that there is a need of another unit of reuse, which should allow fine-grained reuse of process logic within the range from atomic language constructs to sub-processes and whole processes. The concept of process fragments is a promising candidate to fill this gap. We understand a process fragment as a connected process structure, which has relaxed completeness and consistency criteria compared to an executable process. We claim that the process fragment concept allows for an easier and faster development of process-based applications, including for instance applications based on Web service compositions. In order to provide empirical evidence for this claim this article introduces a collection of reusable building blocks for use in process design based on the concept of process fragments. The process fragments we describe stem from different domains. We identified these real-world fragments during our research in the field of compliance management, simulation technology, and software service research. Due to this diversity these process fragments have individual characteristics that differentiate them from one another and they reveal different application scenarios in which they can be used. One of the main objectives of this article is to demonstrate the applicability and the benefit of the use of process fragments in development and integration of applications. We envision that the infrastructure for development of process-based and service-based applications will feature in future novel infrastructure components: *process fragment libraries*. A process fragment library provides management functions for versioned storage, retrieval, and sharing of collections of process fragments.

The article's further structure is the following: In Section 2 we introduce a general concept of process fragments and exemplify domain-specific extensions of this concept. Section 3 provides real-world process fragment examples. Following that, we discuss common application scenarios for process fragments, which ease and accelerate the development of process-based applications (Section 4). In Section 5, application scenarios for the use of process fragment libraries in cross-enterprise

collaboration are presented. In Section 6 an example of the application of process fragments during process design is described. Related work from academia and industry is presented in Section 7. Section 8 concludes our work with a discussion about the advantages of using process fragments in the development of process-based applications.

2. Process Fragments

The term “process fragment” is recently gaining momentum in business process management research. Many different definitions and interpretations have been proposed. Often, a process fragment is defined as a connected sub-graph of a process graph [35], whereas a process graph comprises nodes which represent activities to be performed and edges which represent control dependencies between them (e.g. [7, 14]). In other works a fragment is understood as a connected process structure [13]. Section 7 summarizes the most prominent approaches. In this section, we briefly describe our understanding of the concept of process fragments. At first we describe the general aspects of the concept, and then we show how it can be refined for a specific domain.

2.1 General Concept of Process Fragment

We started working with the concept of a process fragment using the definition of a connected sub-graph of a process graph. Work on use cases in the field of compliance management has shown that this definition does not capture all aspects. As discussed in [29], a process fragment can be created top-down and bottom-up. In the top-down approach a fragment is created by extracting connected structures from a given process. Using this approach, a process fragment is indeed a sub-graph of a process graph. In the bottom-up approach, however, a process fragment is created from scratch. It is designed to implement a set of requirements and thus is not a sub-graph of a pre-existing process graph. Therefore, we follow the more general definition which has been proposed in [29]: “A process fragment is defined as a connected graph with significantly relaxed completeness and consistency criteria compared to an executable process graph. *A process fragment is made up of activities, activity placeholders (so-called regions) and control edges that define control dependency among them.* A process fragment may but is not required to: define a context (e.g., variables) and contain a process start or process end node. It may contain multiple incoming and outgoing control edges for integration into a process or with other process fragments. A process fragment has to consist of at least one activity and there must be a way to complete it to an executable process graph. Therefore, a process fragment is not necessarily directly executable and it may be partially undefined.” Incoming control edges are referred to as fragment entry, outgoing control edges are called fragment exit. The definition of a graph is loosened in the case of entries and exists: Entries and exists of process fragments are represented by a regular control flow connector that has either no source or no target.

This definition assumes that a graph-based process language is used to define the control flow of process fragments. There is, however, no assumption of the particular process language used. The general definition needs to be refined when mapping it to a concrete language. For example, further language characteristics such as events, control flow gateways or data edges representing data dependencies might have to be considered. Furthermore, the definition does not limit the expressiveness to single entry and single exit (SESE) structures.

2.2 Domain-specific Refinement

It is possible to extend or limit the general concept of a process fragment when applying it in a particular domain. In our research, one of our focus areas is compliance management in automated business processes. To put it simply, compliance refers to all measures that need to be taken in order to conform to requirements deriving from laws, regulations and internal policies. For example, a process for the approval of vacations has to conform to a certain approval procedure in which a manager is involved. Such a requirement may be modeled using the concept of process fragments. Previous work has shown that for the compliance domain the general concept has to be both limited and extended [29]. The restriction is that the design of a process fragment may not be changed, i.e. that no activities or control dependencies may be removed or reordered when reusing the fragment. Only particular parts of the fragment may be changed, and only in a prescribed manner. This restriction is required to ensure that the compliance requirement implemented by the process fragment is still captured after integration into a process. For this domain-specific refinement new constructs have to be added to the process language, if the language does not support parameterization or the definition of constraints. The activity placeholders introduced in the general concept (regions) can then also be used in a constrained manner, in order to state how they may be replaced. This domain-

specific refinement is an example of the fragment concept (and its corresponding influence on the process language used) that shows how the general concept may be adapted to a particular application domain. In other domains a different refinement might be required.

2.3 Mapping to a Process Language

The general concept of process fragment, as well as any of its domain-specific refinements, has to be mapped to a concrete process language in order to allow working with that concept in practice. A process language has to be extended with additional language constructs to account for the fragment concept and domain-specific extensions. The fragment concept demands for a way to represent one or more incoming control edges for integration (entries), outgoing control edges (exits), and placeholders (regions). For instance, the refinement for the compliance domain requires constraints, annotations and parameters.

In our work on concrete process fragments we use the Business Process Model and Notation (BPMN [21]) to provide a graphical notation for process fragments. Figure 1 shows a subset of the language constructs defined in BPMN. In addition to the standard constructs we added a cloud shape to represent a region. Regarding the domain-specific refinement for compliance, we added the annotation construct, which is connected to a construct by the shown dashed line. The annotation construct can be used to express a constraint or a parameter. The concrete parameterization is performed at the integration of a fragment into a process. We exemplify the use of these constructs in the sample collection of process fragments presented in Section 3. In the following, we also present process fragments specified in the Business Process Execution Language (BPEL [23]). These fragments are block-structured and they do not make use of regions and constraints. Therefore, they can be represented with native language constructs. A discussion of extensions to BPEL for representing process fragments is presented in [27].

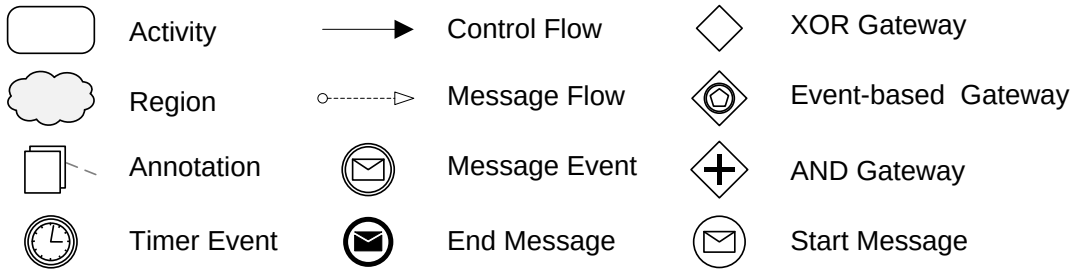


Fig. 1: Process fragment constructs based on BPMN

3. Sample Collection of Process Fragments

In this section a collection of process fragments is presented. The fragments stem from different domains and were identified during our research in compliance management, simulation technology and software service research. These include two industrial case studies with a telecommunication company and a company in the field of defense and security. We selected rather simple fragments for this collection in order to illustrate key concepts of our approach. Concrete application scenarios are discussed in Section 4. All fragments are stored in a process fragment library presented in Section 5. The first example is an approval fragment (Section 3.1). This fragment is extended in Section 3.2, where constraints are put on the possible usages in processes. Section 3.3 presents a security fragment involving two partners. A fragment for user authentication is presented in Section 3.4. Sections 3.5 and 3.6 present fragments for scientific workflows: acquisition and utilization of a scientific service are described. The relation between fragments and model transformation is illustrated by the fragments for completion delay (Section 3.7) and avoidance of infinite waits (Section 3.8).

3.1 Approval

Approval steps (or quality gates) are a reoccurring pattern in processes. The underlying construct is a check of a document followed by an approval. An approval itself may also be used in other situations such as checking for mistakes and authorization [29]. Figure 2 presents a process fragment for approval in BPMN: in case a certain condition is met (“Check Required?”), a particular situation is assessed (“Perform Check”). Depending on the result, one of the two exists is selected. In case

a check is necessary and the check result is not OK, the exit denoting a negative result is taken, otherwise the exit denoting the positive result is followed. The result is also positive if no checking at all is required. When using the fragment in a process model for different scenarios it needs to be adapted at two points: Firstly, the condition “Check Required?” forming the activation condition of the check. Secondly, the concrete Web service or human participant (a.k.a. staff query) triggering a staff including information about the object to be approved. The BPMN diagram of the fragment shown in Figure 2 does not contain this information. In Section 3.2 we show how it can be included in a fragment with the aid of annotations.

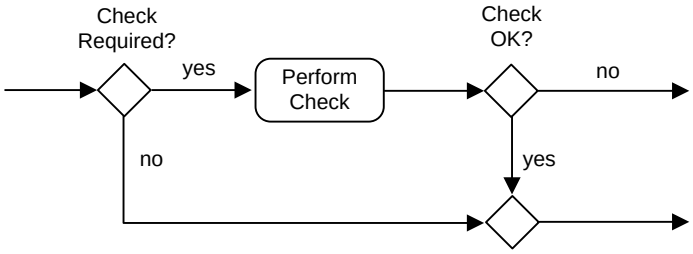


Fig. 2: Process fragment for performing an approval

3.2 Approval with Constraints and Parameters

Section 2.2 presented the motivation of process fragments restricting their embedding in processes in the context of compliance. Figure 3 presents an extended version of the fragment presented in Section 3.1. Here, annotations are used to explicitly state the parameters and to constrain the activities that may be added between the first gateway and the activity executing the checking (“Perform Check”). The constraint put on the region states that the process may not be exited.

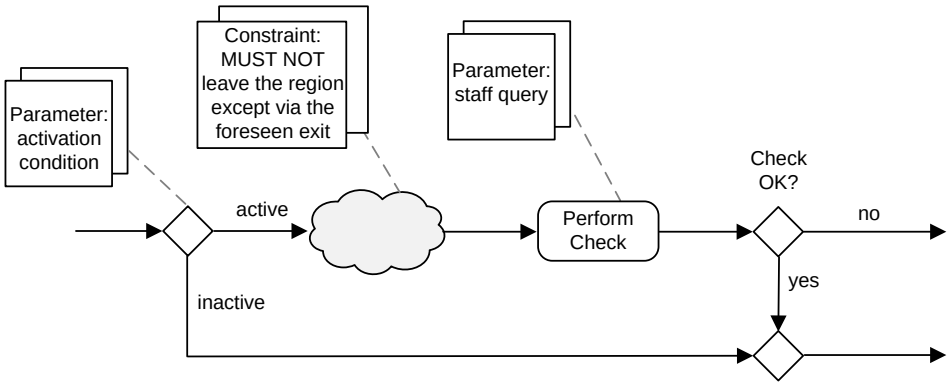


Fig. 3: Constrained and parameterized process fragment for performing an approval

3.3 Secured Service Invocation

Security and integrity are desired in B2B integration scenarios. In general, there are two ways to achieve security and integrity of a sent message: (i) use middleware functionalities or (ii) direct integration in the process. In case the middleware approach is used, the middleware automatically signs and encrypts the message at the sender’s side and decrypts and checks the signature at the receiver’s side. Typically, messaging activities that should exchange messages in a secure and trusted manner are annotated by this requirement, which in turn triggers the respective middleware capabilities. The steps executed by the middleware may be documented by a fragment. If one opts for direct integration in the process, these steps are executed by the process engine. Figure 4 shows an example fragment. This fragment is split up into two roles: the sender A and the receiver B. A message is first signed, then encrypted and then signed again at the sender’s side. Messages are signed first and then encrypted to prevent collision attacks. As decryption is more expensive than checking a signature an additional signing step is added to enable a pre-check of the message at the receiver’s side to be able to quickly reject forged large messages. The final signature validation is necessary to exclude forged messages relying on hash collision. The fragment presented includes multiple roles. A general discussion on fragments used in collaborations is presented in Section 4.3.

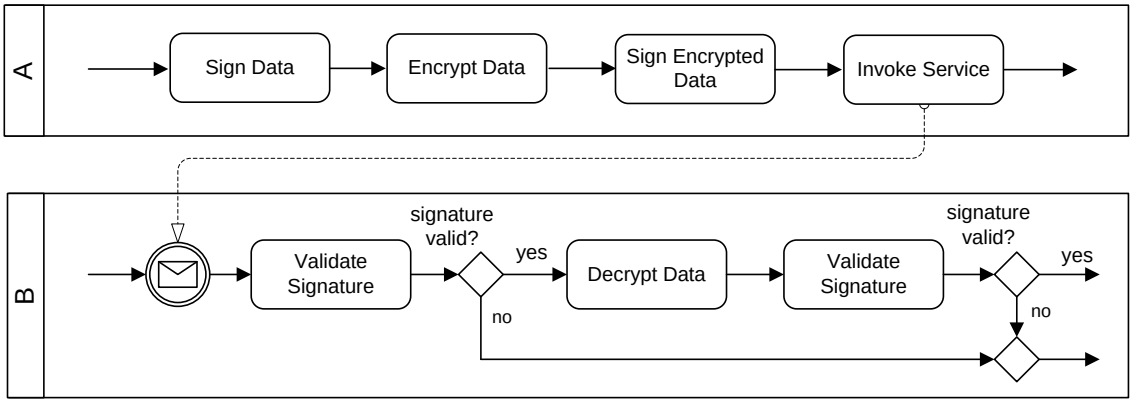


Fig. 4: Process fragment for secured service invocation

3.4 User Authentication

User authentication is frequent requirement for processes offered as a service. For instance, the usage of a travel booking service that is offered over the internet should only be allowed for registered users. As a consequence, the user has to authenticate himself. Figure 5 shows a process fragment for user authentication. The fragment receives a login request from a user and invokes a service to check the user's login data ("Check User Data"). If the information provided by the user is valid, a session identifier is generated and returned to the user. Otherwise, the user is notified about the login failure.

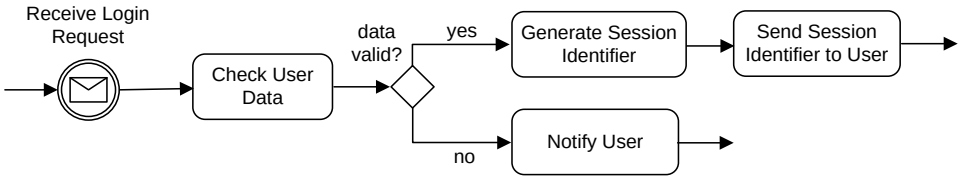


Fig. 5: Process fragment for user authentication

3.5 Acquisition of a Scientific Service

In scientific environments services are often resource-demanding: services that calculate complex simulations can easily allocate a complete processor for a long period of time (ranging from a few hours to several weeks). Such services are therefore usually managed by a resource management component, where users can signal interest in a service. Figure 6 shows a process fragment for the acquisition of a scientific service. First, the acquisition request has to be prepared (stating which service should be allocated for what time frame). The resource management is then requested for an endpoint that provides the requested service. If no service is available, the resource management sends a fault message in response. If an idle service could be acquired, the endpoint reference (EPR) pointing to the service location is sent to the requester. The service is then reserved exclusively for the requester. The requester stores the EPR and can then communicate with the service (represented by the region). Finally, the service is released by sending an appropriate message to the resource management. The scientific service can then be used by another interested party. This fragment can be refined by putting a constraint on the region that states the region must not be exited except via the foreseen exit.

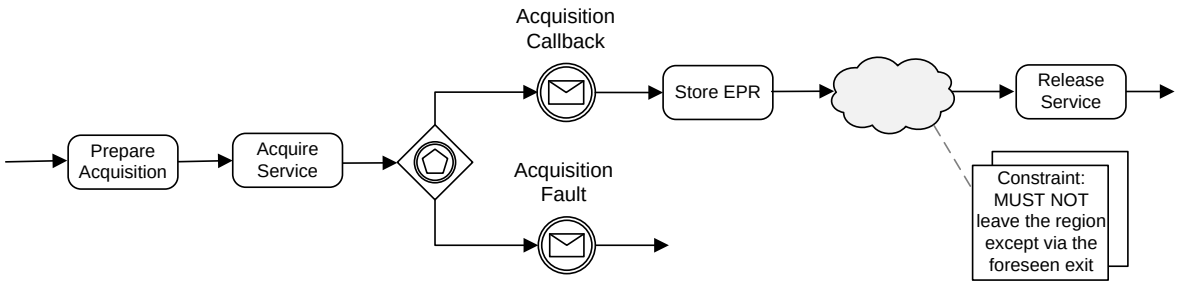


Fig. 6: Process fragment for scientific service acquisition

3.6 Usage of an Acquired Scientific Service

Services that participate in an environment for scientific calculations and simulations are controlled by a resource manager. Invocation of such services therefore differs from usual services. Figure 7 shows a fragment for invocation of a scientific service revealing these differences. As usual, the request message to the service has to be created first. Then, the service is invoked asynchronously. Asynchronous service invocation is used because of the time-consuming nature of scientific calculations. Three cases can be distinguished and therefore should be implemented by the requester. Firstly, the scientific service finishes and sends its response back to the requester. Secondly, the resource manager detects that the service has become unavailable in the meantime (e.g. due to a network partition) and sends the message "Service Unavailable" to the requester. Thirdly, the service acquisition ticket obtained by the requester has expired. In this case, the resource manager sends a message "Service Ticket Expired" to the requester. The scientific service is released and has to be acquired prior to further usage.

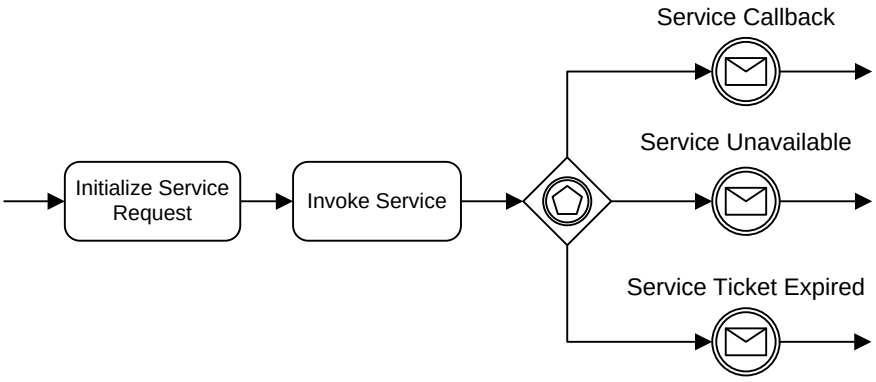


Fig. 7: Process fragment for usage of an acquired scientific service

3.7 Activity Completion Delay

Fragments may be used in model transformation scenarios. These fragments are typically of simple structure. They add a particular behavior to a process before it is deployed on a process engine for execution. Using the model transformation approach, even simple fragments may decrease modeling time, as the transformed code does not have to be manually inserted. For these scenarios, we want to show a simple but still useful fragment that models the delayed completion of an activity. Having such a behavior is for instance useful in some distributed systems with data replication, in order to avoid a dirty read in subsequent activities of a process. This fragment can be generated automatically (using the model transformation approach [32]) for activities which are annotated to complete with a particular delay. Figure 8 illustrates this model transformation in BPMN. A task that is annotated to complete with a particular delay is transformed into a task, followed by an intermediate timer event, which implements the completion delay.

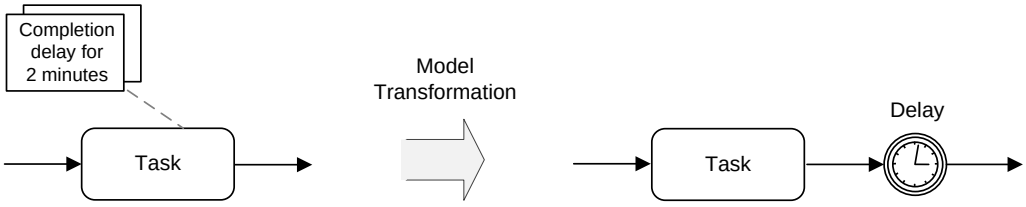


Fig. 8: Transformation fragment for delay after execution

This fragment and the respective transformation can be used in BPEL in the same manner. Listing 1 shows an annotated `<invoke>` activity in BPEL code. An `<invoke>` activity is used to invoke a Web service. The `name` attribute is a standard BPEL attribute, whereas the `ext:completionDelay` attribute is the extension.

Listing 1: Activity annotated with completion delay

```
<invoke name="Task" ext:completionDelay="PT2M" .../>
```

The fragment that is generated out of the annotated activity consists of a <sequence> containing the <invoke> activity whose completion should be delayed for a particular time, followed by a <wait> activity which implements the delay. Listing 2 shows the result of the model transformation.

Listing 2: Transformed activity

```
<sequence>
   <invoke name="Task" .../>
   <wait name="CompletionDelay" for="PT2M" />
</sequence>
```

3.8 Avoidance of Infinite Waits

Process fragments may be used to implement control structures, such as the fragment for avoidance of infinite waits in asynchronous service invocations. A representation of the fragment in BPMN is shown in Figure 9 (right). It defines a control structure that assures that a process does not wait for infinite time for a reply of an invoked service, which eventually will never send a response. The control structure implemented by this fragment is simple, but it still represents common knowledge about process design. A general discussion on the relationship between fragments and model transformation is presented in Section 4.4.

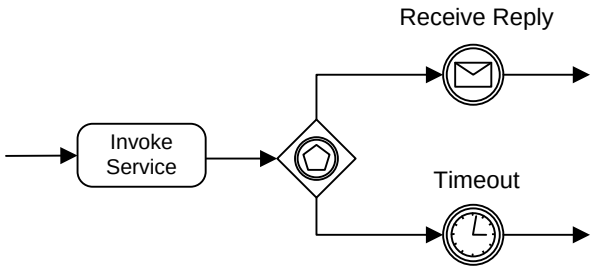


Fig. 9: Process fragment for avoidance of infinite waits

In [15] it was proposed to use similar fragments in a model transformation scenario, as described in Section 3.7. According to that work, a process can be annotated with instructions to enhance its fault tolerance. In particular, service invocation activities (<invoke>) are annotated with information that states whether they should be ignored, skipped, retried, or if an alternate service should be invoked in case a fault occurs during invocation. The fragment for avoidance of infinite waits may be used in the same way.

The fragment is also shown in Listing 3. In BPEL code, this fragment can be modeled with a <pick> construct that is placed after the service invocation. The nested receiving construct <onMessage> represents the callback for the invoked service. Within the <pick> there is also an <onAlarm> construct nested. This construct registers a timeout if a response is not received within a particular period of time.

Listing 3: Process fragment for avoidance of infinite waits

```
<sequence name="tryInvocation">
   <invoke name="invokeService" .../>
   <pick name="awaitResponse">
      <onMessage ...>
         <!-- handle the reply -->
      </onMessage>
      <onAlarm for="P1DT00H">
         <!-- handle the timeout -->
      </onAlarm>
   </pick>
</sequence>
```

4. Application Scenarios of Process Fragments

The collection of process fragments presented in Section 3 allows us to formulate common application scenarios for process fragments which ease and accelerate the development of process-based applications. Firstly, process fragments can allow for *reuse of process logic*. Secondly, they can be used as an *annotation to a process or service* in order to state how to interact with the service or process. Thirdly, fragments and their counterparts realize a particular *collaboration* which itself is reusable. A fragment counterpart is a fragment designed for interaction with another fragment. Fourthly, *process fragments in model transformation* scenarios add particular functions to a process, or slightly change its behavior. These application scenarios are presented next in this section in more detail.

4.1 Process Fragments for Reuse of Process Logic

Reusing process logic is the most common application scenario for process fragments both in academic and industrial research. As illustrated in Figure 10, in order to reuse process logic, at first a fragment of process logic needs to be either extracted from a process (see Figure 10) or created from scratch. Through the creation and subsequent generalization (e.g. removal of process-specific attributes) a new business asset is generated. This asset can be stored in a versioning system which we call *process fragment library*. This library serves as management platform for process fragments and provides functions for storage, search, retrieval, update etc. The assets stored in that library can be used in other processes by formulating queries and retrieving a process fragment that realizes particular process logic. The process fragment can then be integrated into another process, i.e. be reused. The advantages of fragments in this application scenario are basically similar to those in code reuse in traditional programming: One advantage is that the same logic does not need to be specified over and over again. Another advantage lies in an improved quality of the process design, which can be better assured when the process fragments that are used in the process have an efficient design. In case a better fragment is available for a particular task it replaces the less efficient version stored in the library. Over time, the quality of the process logic that is reused increases with this approach.

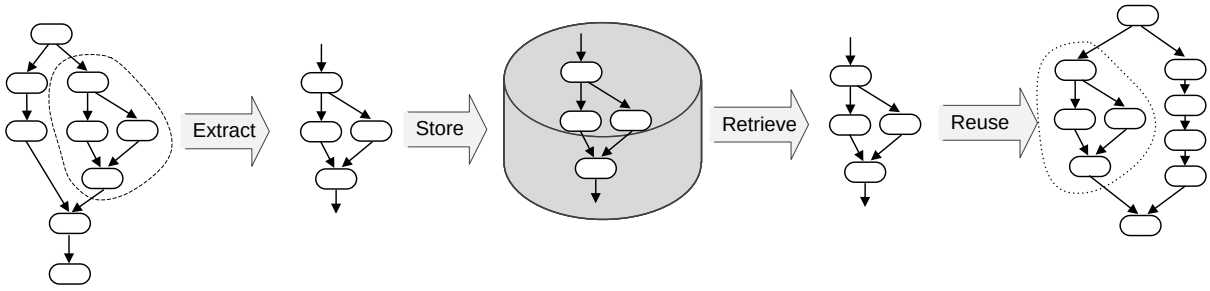


Fig. 10: Process fragments for reuse of process logic

4.2 Process Fragment Annotation to a Process or Service

In this application scenario we explain the use of annotating a process fragment counterpart to a process or service. A fragment counterpart is a fragment that is designed for interaction with the process or service from the partner's point of view. This kind of annotation eases integrating with the process or service that is annotated. First of all, we explain why we distinguish between the terms "process" and "service":

- A *process* is a set of activities connected with control connectors, which define their control dependencies. A process can be instantiated, and a process instance can be identified by a particular set of attributes (correlation sets [3]). Furthermore, in a service-based environment a process can be provided as a service. For instance, a BPEL process is exposed to the outside as a Web service.
- A *service* is an entity which provides particular functionality to the outside via a stable interface. An interface description language (such as WSDL [36]) is used to state which functions the service offers. There is no information about how the service is implemented. The service can be implemented by a program written in Java, by a BPEL process etc. In other words, a service can be a process, but it does not have to be a process. However, in order to integrate with a service, its functions have to be invoked in a well-defined manner, conforming to particular process logic.

In the following, we illustrate the application of process fragments as annotation of a process or service in order to define how to integrate with it, i.e. the annotation describes what the interaction pattern with the service or process must be. The fragments which are used in the annotations are basically process fragments for reuse which interact with the functionality provided by the process or service in a well-defined manner. In other words, these fragments interact with the process or service. To illustrate that, we can make use of concepts of process views in the context of outsourcing [8]. This work describes “White Box”, “Gray Box”, and “Black Box” as terms to describe the different visibility of a process to a business partner. In Figure 11 (left) the concept of the White Box is shown. A process is exposed to the outside as a service. This process might offer particular functions to integrate with another application. The other application has to conform to a particular behavior. This behavior can be expressed by a process fragment that is annotated to that service. If an application would like to use the service, it either has to integrate the fragment (incorporate the fragment in its own process) when it is a process-based application, or it has to implement the behavior described by the fragment in another form. Figure 11 (center) illustrates the Gray Box visibility which is common in outsourcing scenarios. In order not to expose internal details about a process to a business partner, structures irrelevant for that partner are hidden. The fragment counterpart is applicable to that scenario in the same manner. The number of counterparts depends on the particular interaction scenario. Figure 11 (right) illustrates the use of process fragments in service-based applications in general. Independent of its inner (unknown) implementation, the process fragment states how the service can be used. The advantages of having process fragments annotated to a service are that business partners, i.e. service consumers, are provided with a precise description how the service can be used. As we have seen in the process fragment collection in Section 3.6 this is also useful to provide detailed instructions for exception handling. Furthermore, a good design of such fragments reduces integration time due to less required design efforts. An alternative concept to describe the publicly visible behavior of a service is the operating guideline concept [16]. This approach generates an automaton describing the messages allowed to be sent and received. Operating guidelines, however, do not provide information on the internal behavior of a process. Thus, they do not serve as guideline for implementing a process using the service.

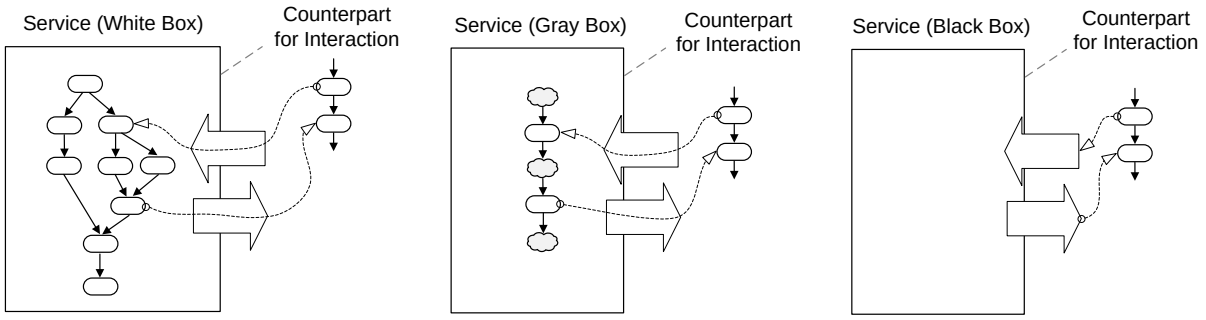


Fig. 11: Process fragment annotation to ease process-based and service-based application integration

4.3 Reusable Collaboration through Process Fragment Choreographies

If we generalize the process fragment annotation scenario discussed in Section 4.2, we obtain the concept of process fragments and process fragment counterparts. In BPM terminology, “choreography” describes the complex interaction between multiple processes. In the field of choreography modeling, there are two paradigms to capture a choreography: interaction models and interconnection models [5]. Interaction models regard interactions as atomic building blocks, whereas interconnection models capture the publicly visible behavior of each participant and connect the interaction activities. In our work, we build on the interconnection model paradigm and call the interconnection between the process fragment and its counterpart “process fragment choreography”.

On the one hand, a choreography describes sequences of message exchanges. On the other hand, Message Exchange Patterns (MEPs, developed by W3C [36]) can also define the exchange of messages between different parties. An MEP includes the sequence and cardinality of messages, the sender, and the recipient of the message. For example, “Request-Response” is a prominent message exchange pattern. A general discussion on limitations and opportunities of these patterns is given in [20]. In fact, these patterns can also be applied in process design, but still they are quite abstract forms of reusable building blocks. Process fragment choreographies are more concrete, and the

fragments contained therein may also contain activities to be performed by a participant of the choreography. In our process fragment collection we showed one fragment with one counterpart in Section 3.3 (secured service invocation). In complex choreographies, however, multiple participants may be involved, see Figure 12 (top). Based on these choreographies, fragments reflecting the interaction can be extracted to describe a particular structure of collaboration (see Figure 12 (bottom)). Such a set of fragments can bring advantages in multiple situations. For instance, when one participant in choreography needs to be exchanged, an alternative participant can be involved by implementing the corresponding fragment counterpart. A set of interacting fragments can also be used to implement particular protocols that the participants of a choreography have to comply with, e.g. a "Request for Bid" protocol in an auction choreography may be described this way.

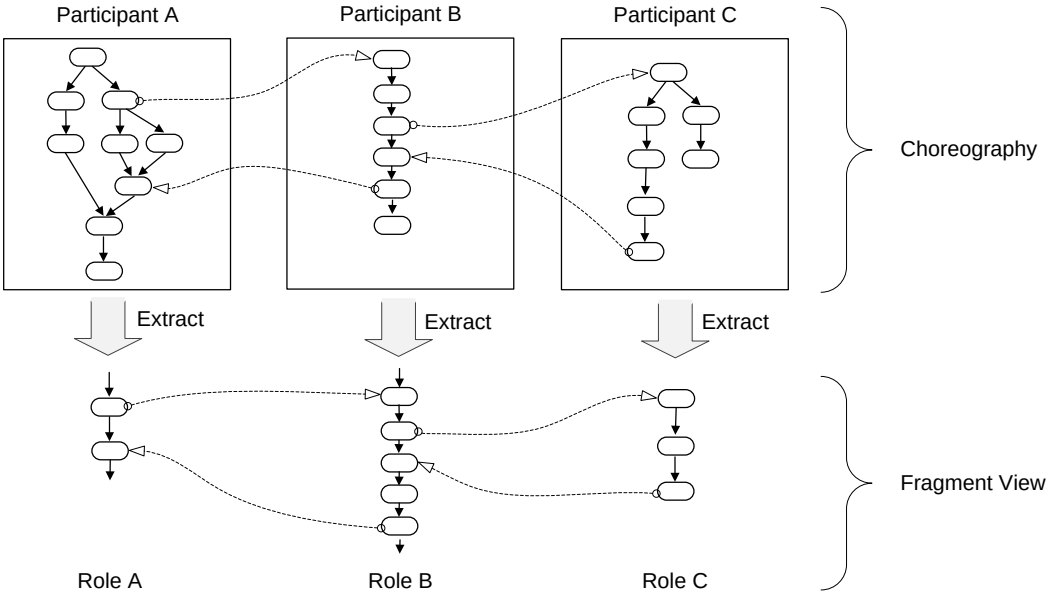


Fig. 12: Process fragment choreographies to describe a collaboration scenario

4.4 Process Fragment Integration by Model Transformation

In this application scenario simple process fragments are integrated into a process model during a transformation step which is performed either during design time or during deployment time. Integration of such fragments may change the behavior of the process. For example, an activity may be transformed into a more complex structure to avoid infinite waits (see Section 3.8). In the process of integration additional activities may also be injected into the process model, e.g. enabling logging functionality. Typically, process fragments used in this scenario implement non-functional requirements or cross-cutting concerns which are not supported by the process engine that is used for execution. The additional requirements are integrated into the process logic in terms of fragments and can then be executed by a standard process engine. According to [29], a disadvantage of this approach is that the process model is "polluted" with additional activities, which do not represent the actual work that should be performed in the process. However, it has been shown how the concept of "process views" can be used to absorb this negative effect on complexity [30]. A process view is an abstraction of a process that hides irrelevant process aspects. For instance, it can be used to show the process in its original form (before the model transformation).

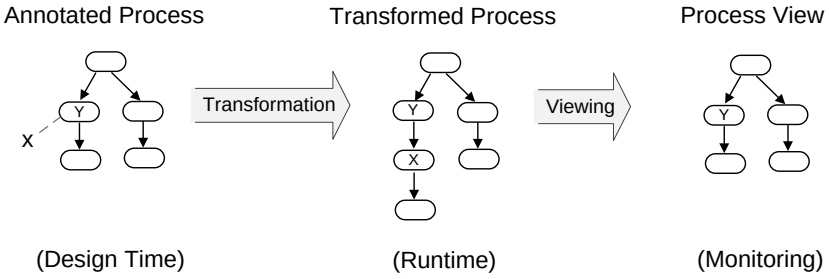


Fig. 13: Process fragment integration by model transformation

The model transformation application scenario is illustrated in Figure 13: A process is annotated in order to state that some activities should be executed in a particular manner. The subsequent model transformation integrates process fragments implementing this behavior. Such a transformation function may inject process fragments into a process, surround particular structures, extend available functionality, etc. The process that is augmented with these fragments is then executed. Next, process view techniques can be applied to show the execution of original process model by hiding the process fragments which have been integrated in the previous transformation.

5. Applying of Process Fragment Libraries

In computer science libraries for versioned management of artifacts are often referred to as repositories [4]. Various works concerning the application of repositories exist. For example, repositories are used in the area of agent systems [31] and they have also been proven useful in the field of semantic business processes [18]. In traditional programming, libraries of code fragments are an established approach for reuse of source code. Furthermore, in domains not directly related to computer science, libraries of reusable building blocks are also used, e.g. in chemistry [19]. In our previous work a repository with advanced functions enabling the management of process fragments has been proposed [28]. We call such an infrastructure component a process fragment library. This work also presented an architecture and a prototype[1] that can be used as technical enabler for using process fragment libraries in development of process-based and service-based applications.

Building on the concept of a fragment library, we introduce three main categories of applying process fragment libraries in cross-enterprise collaboration. These categories are namely *private*, *public*, and *hybrid* fragment libraries. As shown in Figure 14 and explained in the following, these scenarios consider different forms of visibility of the libraries and the business assets stored in them.

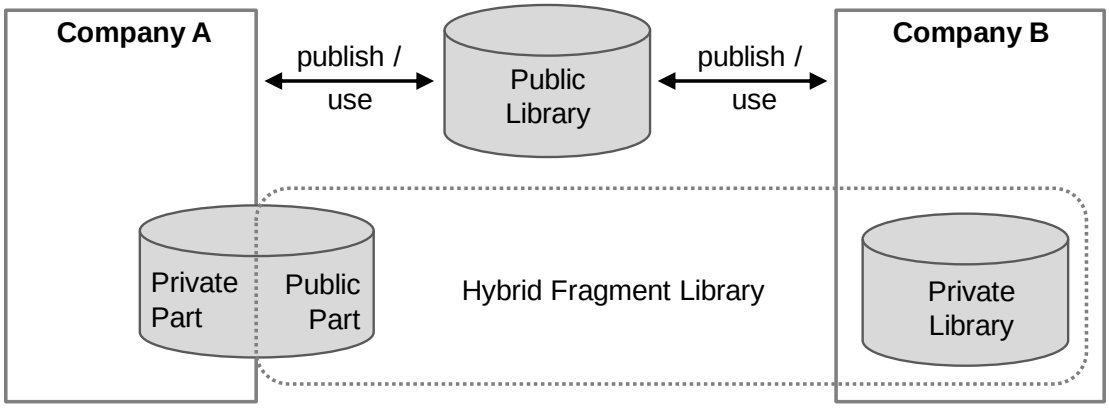


Fig. 14: Application scenarios of process fragment libraries

5.1 Private Process Fragment Library

In today's economy flexibility is essential for the competitiveness of a company. This is of special importance regarding the continuous improvement and adaptation of a company's products and services to changing market and consumer conditions. The company's internal business processes are directly related to the offered products and services. Therefore, flexible adaptation has to entail a fast and easy adaptation of the internal business processes. In order to be best prepared for the future, the management of a company needs to continuously identify potential future trends and define corresponding reactions through adaption of the business processes. For the purpose of reducing the response time to new market conditions, the identified actions and alternatives of adapting the business processes can be modeled and stored as process fragments in the company's internal *private process fragment library*. When market conditions change, the corresponding fragments implementing the action are retrieved from the company's private library and integrated into the business processes. This fragment library has to be kept internal (private) as it contains business secrets. Further scenarios of process fragments contained in a private fragment library are conceivable. For example, fragments from the domains security and trust could be stored there. These fragments implement confidential company internal policies, for instance the process fragments

[1] http://www.iaas.uni-stuttgart.de/forschung/projects/fragmento/start.htm

discussed in Sections 3.2, 3.3, and 3.4 implement such policies. Furthermore, process fragments which describe how the company's internal (Web) services are to be used should not be publicly available. In summary, private process fragment libraries are only to be used internally, by one specific company as depicted in Figure 14.

5.2 Public Process Fragment Library

Public process fragment libraries are available to all interested parties. The contained process fragments might be provided with or without particular license, with costs or free of charge. Public libraries may contain fragments providing templates, which implement best practice in process design. For instance, such fragments address process modeling problems like "how to avoid infinite waits in case there will be no response when calling an external Web service". An example fragment that implements such a best practice is described in Section 3.8. Public fragment libraries can also bring up new kinds of business models based on reusable process logic. For instance, an accountant company could provide its consulting services in form of process fragments which implement particular regulations. Also, a public fragment library could be supplied by a vendor of a business process modeling tool. In order to demonstrate leadership or for better customer support, the library could be made publicly available on the web. The process fragments contained in it can for instance be used in teaching (e.g. during tutorials) to reduce the time to get acquainted with modeling tools. RosettaNet [25] is comparable to a public process fragment library. It provides a collection of Partner Interface Processes (PIPs) which are similar to process fragment choreographies. RosettaNet, however, does not yet use a standardized format to describe the process fragments. Furthermore, the fragments are not yet specified in a manner that allows for their direct integration into a process-based application [11].

5.3 Hybrid Process Fragment Library

Hybrid process fragment libraries are logical federations of multiple libraries from different providers. They contain both publicly available process fragments and also private fragments which are not provided for the public. One reason to build up such federations is when an integration of processes from different companies is required. In order to save costs many companies outsource those parts of their business that can be performed cheaper by another company. The collaboration with the business partner chosen for outsourcing requires that the processes from Company A (the service consumer) and Company B (the provider) are integrated by creating a choreography, i.e. by integrating multiple processes. Each of the companies could provide the fragment counterparts for interacting with their interfaces of the internal business process in order to ease the design of the choreography. These fragment counterparts are also public to both partners within the hybrid libraries. All other assets make up the private parts of the hybrid fragment library. The public and private fragments are likely to be hosted in one library component because they are semantically related to each other. Regarding the public parts of a hybrid library, we can distinguish different levels of visibility. For example, in many cases it is feasible to show the public parts only to a restricted group of business partners. Furthermore, some business partners may be allowed access to "better" fragments, meaning that a "gold customer" has access to fragments performing faster, cheaper etc.

6. Example Application during Process Design

Figure 15 shows an example of use of process fragments in a business process. The process implements *order placement* functionality for the Web shop of a jewelry store. The process is used to check the credit worthiness of a customer. In the process also a check is made if the ordered product is available on stock. In case both checks are successful, the jewelry store frontend is notified and an order fulfillment process is being invoked. If either the credit worthiness check or the availability check fails, the customer is informed accordingly and the process ends. Both checks are executed in parallel as it is assumed that mostly they do not fail.

The process is made up of four process fragments (highlighted by a gray box) plus control structures which glue them together to form a complete process. On the left-hand side there is a secured invocation of a credit bureau service (F2), see also Section 3.3. This fragment may be provided by the credit bureau service as process fragment counterpart for interaction with the service. The security-related tasks are executed by the engine itself using vendor-provided extensions. On the right-hand side, the stock service is invoked. This invocation is made more robust by the fragment for avoidance of infinite waits (F3), see Section 3.8. This fragment is used twice (F3, F4) and thus implements a retry functionality. The process is finalized by sending the respective response messages to the jewelry

store frontend. The jewelry store frontend uses a counterpart of F1, which is provided by the description of the order placement process. This counterpart fragment is comparable to the fragment described in Section 3.6.

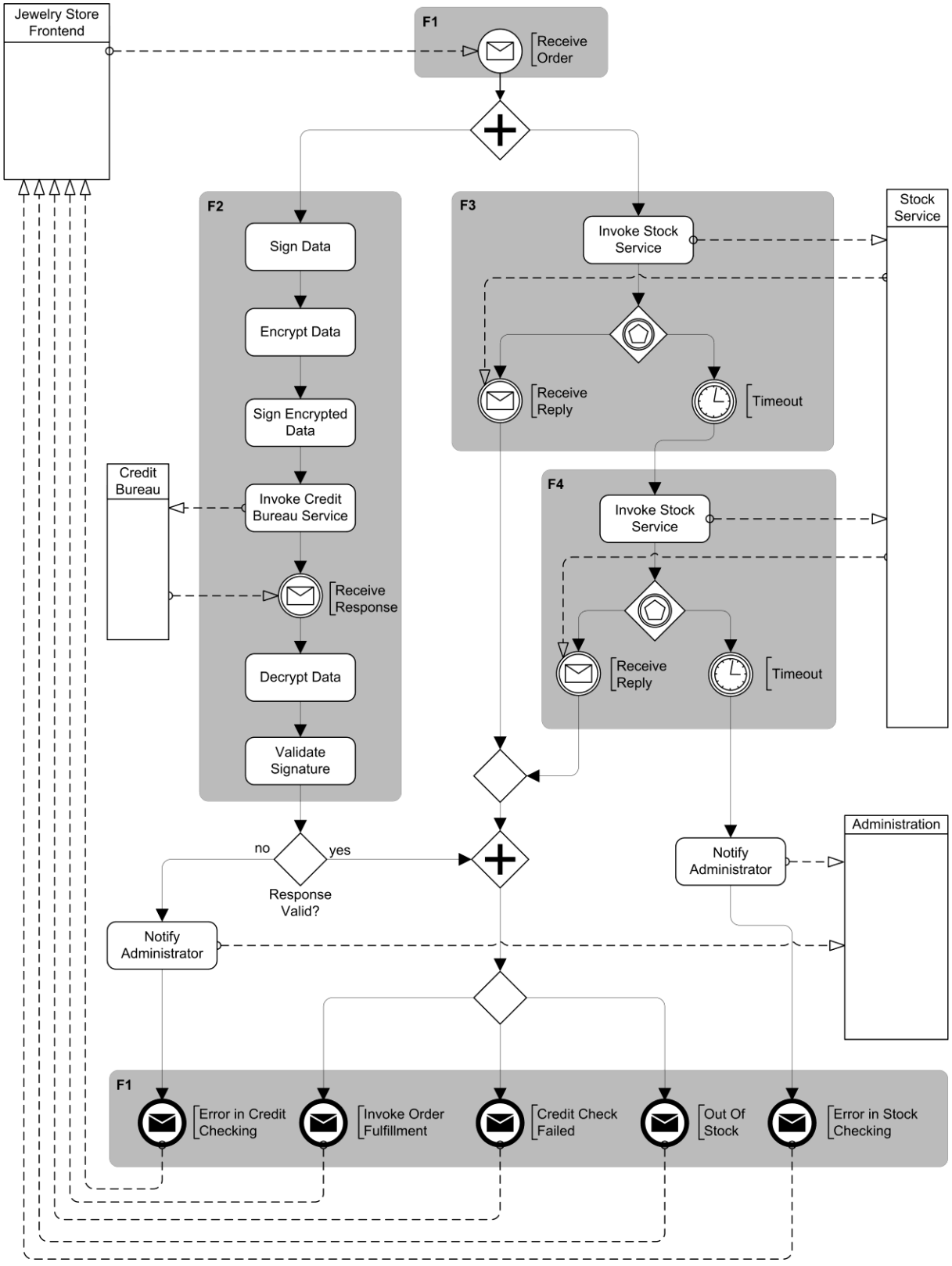


Fig. 15: Example of use of process fragments in a business process

7. Process Fragments in Academia and Industry

In this section, we present works related to our approach: Firstly, we examine academic works on reusability of process structures. After this, we discuss product features from BPM vendors in this field.

7.1 Related Work from Academia

Fragmentation of a process into smaller parts is a fundamental technique for reuse of process structures which already has been thoroughly investigated. To name an early work on this, in [2] software processes represented by Petri Nets are split into sub-graphs for later reuse and composition. In [35] a technique is presented for automatically decomposing a process into single entry and single exit (SESE) fragments for simplification of process analysis. In [7] process fragments are employed for representing local knowledge of process participants. These fragmentation techniques are essential parts of a holistic approach on process fragments. However, they focus on reuse and distribution of process logic and do not consider the other application scenarios we proposed in this article.

Besides the approaches for process fragmentation, further approaches addressing reusability of process structures exist. The authors of [24] for instance introduce the notion of “Process Chunks” for reuse in the requirements engineering process. A process chunk captures generic process knowledge that is useful in particular situations. The concept of “Pockets of Flexibility” [26] allows the flexible definition of processes which are completed to individual instances at runtime. A Pocket of Flexibility consists of a set of activities and sub-processes (called process fragment in this work) which can be composed according to particular composition rules. By placing Pockets of Flexibility within a process, an instance can be individually tailored to the specific needs at runtime. Basically, the notion (and metamodel) for process fragment described in Section 2 may also be applied in this approach to make it even more powerful. The concept of Pockets of Flexibility has some similarities to a more recent approach called “Worklets”. The authors of [1] describe a Worklet as a small, self-contained and complete process that handles one specific task within a larger process. The most notable difference to current implementations for sub-processes is that a Worklet for a particular task can be dynamically selected and used in a running process instance. A Worklet has a single entry and a single exit and is thus still similar to the concept of a sub-process, as for instance discussed in [12]. The concept of process fragments described in Section 2 might be seen as a superset of worklets, as it includes SESE structures, but is not limited to it.

Workflow Patterns [33] is a work related to our approach. A workflow is the technical implementation of a process [14]. There, however, is a significant difference between the concept of a workflow pattern and a fragment. The different Workflow Patterns that have been proposed describe different elementary language constructs of workflows on an abstract level. For example, the control flow pattern “Synchronization” describes the reconvergence of two or more parallel branches into one branch. The patterns states that the subsequent branch is activated when all input branches have been completed. Such patterns are very useful to measure the expressiveness of a workflow language, or to assess the functionality of a process engine. Compared to the process fragments presented in Section 3, the workflow patterns are significantly abstract, whereas a process fragment represents concrete functionality. A process fragment can be compared to a concrete, possibly generalized code snippet of a particular program, whereas a workflow pattern is more like describing a construct of the language used to write such code, such as `if` / `then` / `else`.

Several concepts and methods to support the use of process fragments have been developed. In [17] a technique based on graph matchmaking algorithms is discussed which provides efficient search of process fragments within a fragment library. A user can specify a query in form of a process fragment. The fragment library can then suggest process fragments which are similar to the one specified by the user. Further techniques have been proposed in [30]. In this work, a technique based on process view transformations is described which allows to extract a process fragment from a given process. This technique can also be used to hide a process fragment. This is useful in scenarios where a multitude of process fragments is contained in a process which is not important for understanding of the work that is actually performed in that process. The work does also discuss process fragment highlighting which is useful in analysis scenarios. Another technique related to process fragment use is discussed in [6]. This work suggests composing a process out of process fragments. For this approach, an operation to integrate one fragment with another fragment is described. A technique to integrate a process fragment into a given process is proposed in [27].

7.2 Related Work from Industry

Oracle JDeveloper 11g [22] is a development framework which includes a modeling tool for developing BPEL processes. This modeling tool offers the integration of pre-defined process fragments via drag and drop to ease invocation of particular services. These are, a service for sending e-mails, a service for sending short messages to mobile phones (SMS), a service for integration of human beings in a workflow, and a service providing business rules functionality. An inspection of the code of these fragments reveals that they all have a similar structure. All activities are nested in a `<scope>` construct which stores variables required for invocation. The child element of this `<scope>` is a `<sequence>` that contains one or more `<assign>` activities for preparation of the invocation, an `<invoke>` activity for the actual service invocation, followed by a `<receive>` activity for the callback. Figure 16 shows an illustration of the process fragment which is integrating human beings in a BPEL process, visualized by using BPMN. Depending on the particular service, the invocation is asynchronous for integrating human beings, and synchronous for sending e-mails, SMS, or evaluation of business rules. All these fragments have a single entry and a single exit.

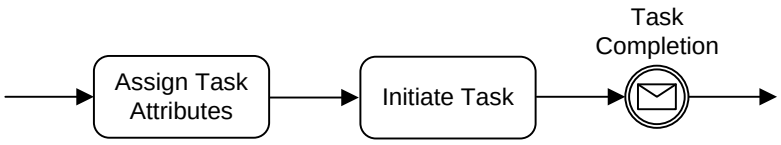


Fig. 16: Process fragment for integration of human tasks

IDS Scheer is another BPM vendor offering functionality for support of process fragments in process design. ARIS Express [10] is a modeling tool offered by this vendor. This tool allows ad hoc definition, storage, and reuse of process fragments in the process language Event-driven Process Chains (EPCs). Process fragments in this tool are understood as a possibly unconnected combination of objects. This definition differs from the concept we discussed in Section 2 as it imposes less restrictions on the model. ARIS Express provides a palette that is integrated in the tool. Within the modeling space a selection of objects can be defined as a fragment, this subsequently appears on this palette. From this palette fragments can be dragged and dropped on the modeling space for reuse.

8. Conclusions

In this article we presented an enhancement of the concept of process fragments along with concrete examples. We showed different application scenarios in which this concept can be applied for the development of process-based and service-based applications and we discussed the potential impact on integration of processes across different businesses using multiple process fragment libraries. We argue that process fragments enable an easier and faster development of process-based applications. We also claim that process fragment counterparts, i.e. fragments designed for interaction with a process or service from the partner's point of view, ease and speed up application integration.

Regarding our first claim, process-based applications have to be developed using either sub-processes, or using atomic language constructs like "task" or "service invocation". In order to design a particular behavior, each activity has to be placed in correct order, using the right attributes etc. When using process fragments, the behavior that should be designed can be reused with less effort, once it is stored in a fragment library. The fragment approach makes the development faster, assuming particular behavior is required frequently in process design. The fragment approach makes the development easier when the fragment implements complex logic, as that logic does not have to be re-designed each time when needed. Furthermore, efficient process design is ensured when the fragments that are reused have an efficient design.

Regarding the second claim, we can compare the situation of application integration when not using process fragment counterparts with the situation when they are used. At first, the interfaces of the offering (process or service) need to be analyzed and textual documentation needs to be consulted in order to derive a protocol for integration. In principle, each time this is done a process fragment counterpart is derived. If another partner would like to integrate with the offering, this procedure has to be performed again. Otherwise, if the business partners and service consumers are provided with a precise description on how the service can be used, this time-consuming procedure can be left out. An efficient design of fragment counterparts furthermore reduces integration problems and malfunctioning service choreographies, which is time-consuming for both, provider and consumer. At best, integration means integrating a given fragment into a process.

Process fragments are important in many different fields. These include Grid computing, manufacturing engineering, and scientific processes. There, process fragments can be identified, designed, extracted and reused. Fragments from particular application domains may also be useful in other domains, or bring up new ideas which are helpful in many fields where processes are used. Therefore, we advocate building up public libraries to share this knowledge.

Acknowledgements

The work published in this article was partially funded by the COMPAS project (www.compas-ict.eu) under the EU 7th Framework Programme Information and Communication Technologies (contract no. FP7-215175), the S-Cube project (www.s-cube-network.eu) under the Network of Excellence (contract no. FP7-215483), the 4CaaSt project (www.4caast.eu) under EU 7th Framework Programme Information and Communication Technologies (contract no. 258862), and the DFG Cluster of Excellence Simulation Technology (www.simtech.uni-stuttgart.de, contract no. EXC310).

References

[1] ADAMS, M., ter HOFSTEDE, A., EDMOND, T., van der AALST, W., Worklets: A Service-oriented Implementation of Dynamic Flexibility in Workflows. *Proceedings of the 14th International Conference on Cooperative Information Systems (CoopIS)*, Springer, 2006.

[2] AVRILIONIS, D., CUNIN, P., FERNSTRÖM, C., OPSIS: a View Mechanism for Software Processes which Supports their Evolution and Reuse. *Proceedings of the 18th International Conference on Software Engineering (ICSE),* IEEE Computer Society, 1996.

[3] BARROS, A., DECKER, G., DUMAS, M., WEBER, F., Correlation Patterns in Service-Oriented Architectures. *Proceedings of the 10th International Conference on Fundamental Approaches to Software Engineering (FASE)*, Springer, 2007

[4] BERNSTEIN, P., DAYAL, U., An Overview of Repository Technology. *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, Morgan Kaufmann, 1994.

[5] DECKER, G., KOPP, O., BARROS, A., An Introduction to Service Choreographies. *Information Technology*, Oldenbourg Verlag, 50(1):122–127, 2008.

[6] EBERLE, H., LEYMANN, F., SCHLEICHER, D., SCHUMM, D., UNGER, T., Process Fragment Composition Operations. *Proceedings of the 5th Asia-Pacific Services Computing Conference (APSCC),* IEEE, 2010.

[7] EBERLE, H., UNGER, T., LEYMANN, F., Process Fragments. *Proceedings of the 17th International Conference on Cooperative Information Systems (CoopIS)*, Springer, 2009.

[8] ESHUIS, R., NORTA, A., A Framework for Service Outsourcing using Process Views. *Proceedings of the 14th IEEE International EDOC Conference (EDOC 2010),* IEEE Computer Society, 2010.

[9] HALLERBACH, A., BAUER, T., REICHERT, M., Managing Process Variants in the Process Lifecycle. *Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS'08),* 2008.

[10] IDS Scheer: *ARIS Express*, 2010. [online] on < http://www.ariscommunity.com/aris-express >

[11] KHALAF, R., From RosettaNet PIPs to BPEL Processes: A three level Approach for Business Protocols. *Data and Knowledge Engineering,* 61(1):23–38, 2007.

[12] KOPP, O., EBERLE, H., LEYMANN, F., UNGER, T., The Subprocess Spectrum. *Proceedings of the 3rd International Conference on Business Process and Services Computing (BPSC),* Gesellschaft für Informatik e.V. (GI), 2010.

[13] KOSCHMIDER, A., HABRYN, F., GOTTSCHALK, F., Real Support for Perspective-Compliant Business Process Design. *Proceedings of the 4th International Workshop on Business Process Design*, Springer, 2008.

[14] LEYMANN, F., ROLLER, D., *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, 2000, ISBN 0-13-021753-3

[15] LIU, A., LI, Q., HUANG, L., XIAO, M., A Declarative Approach to Enhancing the Reliability of BPEL Processes, *IEEE International Conference on Web Services (ICWS),* IEEE Computer Society*,* 2007.

[16] LOHMANN, N., MASSUTHE, P., WOLF, K., Operating Guidelines for Finite-State Services. P*roceedings of the 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007),* Springer, 2007.

[17] MA, Z., LU, W., LEYMANN, F., Query Structural Information of BPEL Processes. *Proceedings of the 4th International Conference on Internet and Web Applications and Services (ICIW 2009),* IEEE*,* 2009.

[18] MA, Z., WETZSTEIN, B., ANICIC, D., HEYMANS, S., LEYMANN, F., Semantic Business Process Repository. *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM),* 2007.

[19] Maybridge: *The Maybridge Ro3 Fragment Library*, product flyer, 2007. [online] on < http://www.maybridge.com/images/pdfs/ro3frag.pdf >

[20] NITZSCHE, J., van LESSEN, T., LEYMANN, F., WSDL 2.0 Message Exchange Patterns: Limitations and Opportunities. *Proceedings of the 3rd International Conference on Internet and Web Applications and Services (ICIW),* IEEE*,* 2008.

[21] Object Management Group (OMG): *Business Process Model and Notation (BPMN)*, Version 2.0, OMG Document Number formal/2011-01-03, January 2011.

[22] Oracle: *Oracle JDeveloper 11g*, 2009. [online] on < http://www.oracle.com/technology/products/jdev/index.html >

[23] Organization for the Advancement of Structured Information Standards (OASIS): *Business Process Execution Language 2.0 (BPEL)*. 2007.

[24] ROLLAND, C., PRAKASH, N., Reusable Process Chunks. *Proceedings of the 4th International Conference Database and Expert Systems Applications (DEXA'93),* Springer, 1993.

[25] RosettaNet: Cluster, Segments, and PIPs, 2010, [online] on < http://www.rosettanet.org/ >

[26] SADIQ, S., SADIQ, W., ORLOWSKA, M., Pockets of Flexibility in Workflow Specification. *Proceedings of the 20th International Conference on Conceptual Modeling*, Springer, 2001.

[27] SCHUMM, D., ANSTETT, T., LEYMANN, F., SCHLEICHER, D., STRAUCH, S., Essential Aspects of Compliance Management with Focus on Business Process Automation. *Proceedings of the 3rd International Conference on Business Process and Services Computing (BPSC),* Gesellschaft für Informatik e.V. (GI), 2010.

[28] SCHUMM, D., KARASTOYANOVA, D., LEYMANN, F., STRAUCH, S., Fragmento: Advanced Process Fragment Library. *Proceedings of the 19th International Conference on Information Systems Development (ISD'10),* Springer, 2010.

[29] SCHUMM, D., LEYMANN, F., MA, Z., SCHEIBLER, T., STRAUCH, *S.,* Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls. *Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI'10),* 2010.

[30] SCHUMM, D., LEYMANN, F., STREULE, A., Process Views to Support Compliance Management in Business Processes. *Proceedings of the 11th International Conference on Electronic Commerce and Web Technologies (EC-Web'10)*, Springer, 2010.

[31] SEIDITA V., COSSENTINO M., GAGLIO S., A Repository of Fragments for Agent Systems Design. *Proceedings of the Workshop on Objects and Agents (WOA'06)*, 2006.

[32] STAHL, T., VÖLTER, M., CZARNECKI, K., *Model-driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006, ISBN 0-47-002570-0

[33] van der AALST, W., ter HOFSTEDE, A., KIEPUSZEWSKI, B., BARROS, A., Workflow Patterns. *Distributed and Parallel Databases,* 14(1):5–51, Springer, 2003.

[34] van der AALST, W., DREILING, A., GOTTSCHALK, F., ROSEMANN, M., JANSEN-VULLERS, M.H., Configurable Process Models as a Basis for Reference Modeling. *Proceedings of the Workshop on Business Process Reference Models (BPRM),* Springer, 2005.

[35] VANHATALO, J., VÖLZER, H., LEYMANN, F., Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. *Proceedings of the 15th International Conference on Service-Oriented Computing (ICSOC 2007)*, Springer, 2007.

[36] World Wide Web Consortium (W3C): *Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts*, W3C Recommendation, June 2007. [online] on < http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626/ >

All links were last followed on December 17, 2010.