Process Fragments*

Hanna Eberle, Tobias Unger, and Frank Leymann

University of Stuttgart, Institute of Architecture of Application Systems
Universitätsstraße 38, 70569 Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

Abstract. The concepts presented in this paper are motivated by the assumption that process knowledge is distributed knowledge and not completely known just by one person. Driven by this assumption we deal in this paper with the following questions: How can partial process knowledge be represented? How can this partial knowledge be used to define something more complete? To use higher level artefacts as building blocks to new applications has a long tradition in software engineering to increase flexibility and reduce modeling costs. In this paper we take a first step in applying this concept to processes, by defining process building blocks and operations which compose process building blocks. The building blocks will be referred to as process fragments in the following. The process fragment composition may take place either at design or runtime of the process. The design time approach reduces design costs by reusing artefacts. However the runtime fragment composition approach realizes high flexibility due to the possibility in the dynamic selection of the fragments to be composed. The contribution of this work lies in a fragment definition that enables the fragment modeler to represent his 'local' and fragmentary knowledge in a formal way and which allows fragment models to be composed.

1 Introduction

Making business processes more flexible is a broad research area in BPM today. Flexibility in business processes means that the business process is able to react to varying situations and requirements. Flexibility in business processes can be realized in different ways. One way to realize flexibility is to model all alternatives possible as process. Often this approach is not possible, because the relevant knowledge becomes available e.g. not until the process is running. Flexibility is required especially in highly dynamic environments. Environments where not everything can be known at one time, e.g. design time, where one person has not a complete view on the process, and where things change very often. In these highly dynamic environments we consider process knowledge as something local and fragmentary. Where 'local' means that the knowledge is bound e.g. to a certain

* This work is partially funded by the ALLOW project. ALLOW (http://www.allow-project.eu/) is part of the EU 7th Framework Programme (contract no. FP7-213339).

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part I, LNCS 5870, pp. 398–405, 2009.

location, to a certain situation or context or time or person. The term fragment is derived from the Latin verb *frangere* which stands for the english word *to break*. Therefore a fragment is defined as a part broken off something whole, like a shard is a part of a broken vase. A shard becomes a fragment, if it can be identified and glued to the vase it belongs to. The whole thing has some properties which are not evident in a fragment, e.g. a shard cannot be used as a container for something to store, whereas a vase can used as one. Fragments of a vase can not be glued together arbitrarily. The fitting fragments must be found and glued together in a suitable way. Having this assumption in mind, that knowledge is something 'local' and fragmentary, which needs to be build together depending on time or location, to receive a more complete view on the current knowledge for a certain situation, we adopt this idea for process modeling and execution. In our approach presented in this paper we model local process knowledge as process fragments, where process fragments can be glued together to a complete process. Process fragments are reflecting the partial and intermittent knowledge one modeler has at a certain time about a specific situation. Process fragments to be executed are chosen depending on data like the current location, and modeling data, like business case or other annotations. The work done in this paper in summarized in short in the following. We implement flexibility as 'local' process fragments, which can be glued together depending on situation, location or other context information. In this paper we motivate and draw an overall picture of the development and execution of fragment-based processes in Sections 2 and 3. We develop a fragment model definition, which leaves the modeler enough freedom in modeling 'local' knowledge and provides enough information to be able to compose fragments using syntactical correct operations in Section 5. We close this paper by relating our work to other work done in this area in Section 6 and a conclusion and a outlook for further work in 7.

2 Motivation and Application Scenario

As scenario we regard the local knowledge bank employees and other involved people have about the loan approval process [6]. Local knowledge about this domain is available from following persons: The bank front desk officer, the approver, the assessor, and the applicant. In the following we regard the local knowledge of the involved roles, which usually know at least the activities they have to perform, but the activities are not necessarily performed by them. The applicant knows that an application form has to be filled in to apply for a loan and that a notification is sent out by the bank to inform the applicant about either the reject or the accept of the application. The bank's front desk officer knows that credit information must be collected using a certain form. Afterwards the application is routed depending loan amount. The assessment officer knows about two assessment activities which are sequenced one after another. Depending on the outcome of the first assessment step either the other assessment step is performed or routed another way the risk is too high. The loan approval officer knows that a approval activity has to be done. This approval

step is necessarily done by the loan approval officer, because only he has the right and the duty to carry the responsibility of his decision. The loan approval officer knows that the assessment activity is performed in two cases; either the amount of the loan exceeds 10000 or the the risk is too high and someone has to take responsibility. After his work is done the loan approval is either rejected or approved. The local knowledge that the bank employees apply depends not only on the business case to be executed but other differentiations to be made are necessary. E.g. an assessment officer has differing checks to perform depending on the economic situation. If the economic situation is stable fewer checks have to be performed than if the economic situation is an instable one. Therefore the local knowledge affiliated with the a business case might also be context depended. Other scenarios e.g. in real world scenarios where actors and executors move through space are conceivable, where local knowledge gets successively available depending on the executors location.

3 Fragment-Based Process Modeling and Execution

All people having fragmentary knowledge about the process model shall be able to model their knowledge as process fragment. Therefore lots of people have access to modeling process. The new designed process fragments get later annotated with local information, e.g. at what situation this fragment is used. Because there might be different behaviors a bank exhibits depending on the overall current economical situation. The process fragments get stored in a fragment repository. Having process fragments acquired the fragment definitions are ready to get composed by the fragment composer. The fragment composer implements a set of operations on how fragment definitions can be glued together. One operation is presented in detail in the remaining parts of this work. Two different approaches for composition times are possible. Either the 'local', situation-depended knowledge gets integrated to a complete process before the

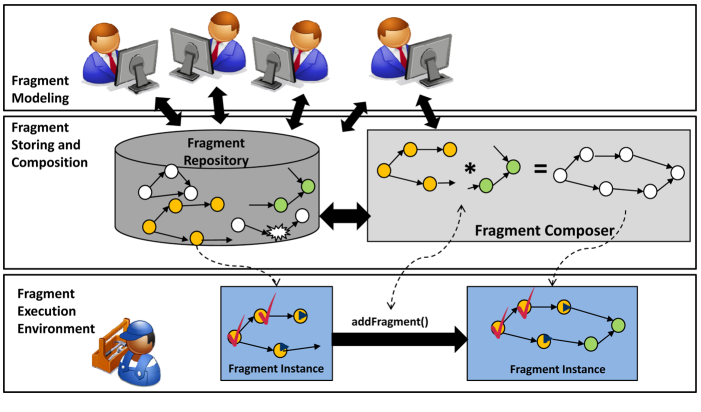

Fig. 1. Architectural Overview

process starts or it gets composed over a period of time, while known parts of the composed fragments can be executed in the meantime. The build-time approach can be used for either to build ad-hoc processes, which get executed only once, because it is strongly depending on a certain situation at composition time, or to as process which gets executed over and over again, as long as e.g. certain situation constraints hold. This build-time requires all fragments to be composed to be known before execution, which is a hard requirement for long running business processes, which run over years. For processes which have very high dependencies on current situations, e.g. like location, because they live in highly dynamic environments, the build-time approaches do not fit. Fragments and situation information becomes available not until runtime. This approach requires the running fragment instances to be migrated to the newly composed fragment definition. A overview of the architecture is shown in Figure 1.

4 Process Fragments

We consider a process fragment a fragmentary 'local' building block, representing incomplete 'local' knowledge, which can be composed with other fragments to build a complete and correct process model. Therefore a fragment model definition must serve following needs: 1. Fragment models must be able reflect the 'local' fragmentary process knowledge. 2. Fragment models must be composable to a complete process model. A fragment model definition must support and enable the fragment modeler to pour all knowledge he has into the formal form of a process based representation, a process fragment. Therefore we must provide modeling elements, which allow the modeler to specify, what is known, and to specify, what he knows he doesn't know. Therefore it must be possible to leave gaps in the definition, saying "'I know there happens something, but I cannot exactly tell what."'. All things that the modeler has concrete knowledge about can be modeled using established process modeling elements. Orderings of sets of activities can be specified by defining control connectors. Hyper-edges can be defined upon the set of activities and annotated with some alternative execution paths in case of faults appear within the contained activity set. Additionally to the established process modeling elements we provide modeling elements, which enable to model, where the exact business logic is not known. E.g. if it is known that activity A must be before B, but what exactly happens between A's completion and B's execution, is not known. Therefore we introduce a new modeling element called *region*. A region stands for business logic, which is not further defined. It can be connected with activities by defining normal control connectors (Figure 2). In the case that it is known, that three distinguishable proceedings are possible after activity C, but it's not known what activities are to be done afterwards, we leave the modeler the freedom to model this as activity C with three outgoing control connectors with no target activities defined. We leave the modeler even the possibility not to model control flow relation between activities at all. As a second step we need to define, what makes a fragment correct. We base our considerations herein on the assumption, that the composition of fragments should result in a correct process model. Therefore we derive correctness

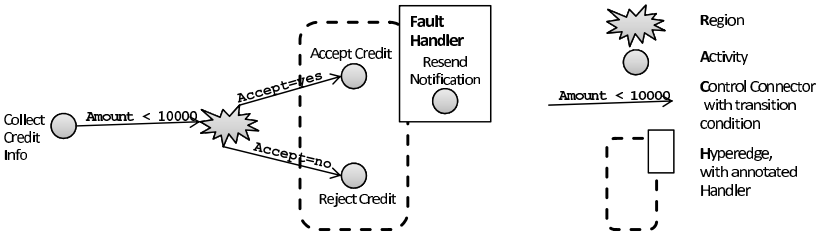

Fig. 2. Fragment Example

criteria directly from correctness criteria for process models. Fragment correctness criteria are defined in the following. A correct process model is required to have **no circles**. So does the fragment model. Because if a fragment building block contains a circle, the process completed of this fragment contains a circle. A correct process model is required to have no two control connectors to have the same target and source activities. The same holds for fragments. The hyperedges concerning transactional behavior of a correct process model are required to be properly nested. Again, each fragment must comply to this constraint. In order to restrict the numbers of dangling control connectors we define following constraint, where dangling means, that either the source or the target of a control connector is not defined. If there are dangling control connectors within a **connected fragment partition** with an equal transition condition one missing a target activity and the other one lacking in the source activity definition and if these two dangling control connectors can be glued together without defining a circle in the fragments control paths, than these two connectors must be glued together. We call this constraint *compactness constraint*, where compactness stands for the least fan out possible. A example of a non-compact fragment and a compact fragment is shown in Figure 3.

Fig. 3. Non-Compact Fragment vs. Compact Fragment

5 Fragment Composition Using Redundancy

Due to the 'local' knowledge modeling approach it is possible that some information is modeled redundantly. Two modeler might have partially overlapping knowledge i.e. some activities are contained in several fragments as well as some links connected to such activities. This redundancy can be used to define a first fragment composition operation by using the redundant information as gluing points between these two fragments. Based on the two fragment definitions we

have given a set of the redundant activities, mapping exactly one activity onto one activity. There are lots of conceivable criteria [e.g. [8]], where the activity equality definition can be based on. The definition of these criteria is out of scope of this work. Furthermore, we assume, that no fragment contains more than one activity of the same activity definition. Note, that the fragments to be composed must not be contradicting, what means that two fragments are contradicting with respect to a composition operation, if the composed fragment is not compliant to the correctness criteria defined above, e.g. if the composed fragment contains circles and not properly nested hyperedges. The composition operation consists of following steps also presented in Figure 4. Firstly fragments activity, region, hyperedge and control connector sets, get united, where two equal activities get united to one activity. Now, the result may contain two control connectors connecting the same activities. To comply to the fragment correctness constraints, we merge these control connectors as follows: If the two control connectors have the same transition condition, we simply delete one connector, if the two control connectors have differing transition conditions the control connectors get merged by joining the transition condition with a OR-operator. After this step, we carry on with further refinement steps to make it compliant to further fragment constraints. Step two we are trying to substitute parts of

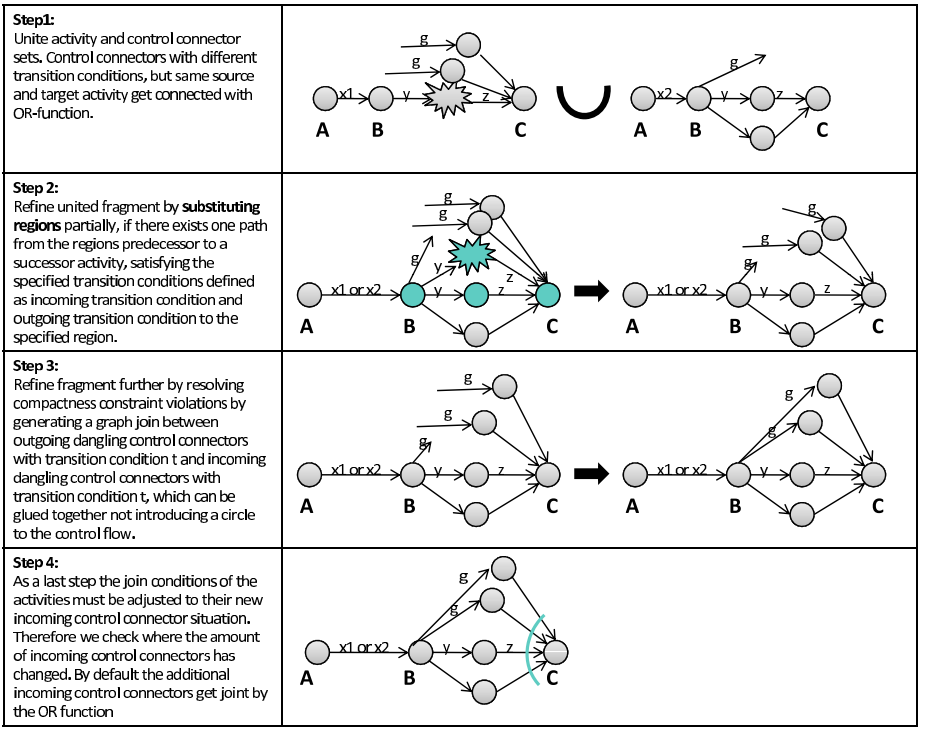

Fig. 4. Fragment Composition Operation

regions, with concrete parts. Afterwards we connect all dangling control connectors with the same transition condition to comply to the compactness constraint and not introducing circles by connecting the wrong control connectors. The last step deals with the adjustment of join conditions. Join conditions must be adjusted, where the amount of incoming control connectors has changed during the operation. The composition operation can be also applied to non-overlapping fragment knowledge. If this is the case the result fragment of the operation is a fragment with two unconnected parts. The composition operation has following algebraic properties. It is closed, commutative and has a neural element. It is not associative. The proof of these properties is out of scope of this paper.

6 Related Work

Composition operations are well known in the domain of formal languages, automaton theory [4], whereas the composition operation approach is new to the workflow domain. Since we use composition to realize flexibility in workflows we focus in the following on further flexibility approaches in the workflow domain. One possibility to deal with flexibility in workflows is to model all alternatives into the workflow model. The result of the modeling process is a very complex and very probably an unreadable workflow model. All information has to be available at design-time. A more dynamic approach in dealing with flexibility is to leave some gaps in the workflow specification, by e.g. defining some abstract activities with no fix activity implementation. The gaps are to be stuffed at runtime, when additional information gets available. There are many possible realizations of this gap-flexibility approach. In [1] a concept is introduced, on how subprocesses can be found and bound at runtime as implementation to a abstract activity. The subprocesses are annotated with contextual information, which enables a context aware selection of suitable subprocesses at runtime according to the current context configuration. These subprocesses are called 'worklets'. Other approaches dealing with dynamic service selection can be found here [5]. There exist also aspect-oriented approaches in workflow management to constitute frlxibility. Activities get annotated with aspects. If the aspect is hit by process context information the normal execution of a workflow can be interrupted and inter-weaved with additional activities [2]. In [9] process fragments are inserted into a process model in order to realize flexibility both at design- and runtime. Process fragments are also discussed in [7] as special reuse approach. Process variants as described in [3] also used for process flexibility could be mapped onto process fragments as representation. If we compare these approaches with the fragment-based flexibility approach, it strikes that all approaches base on one main process definition, which can be slightly changed and adapted during runtime, which is not the case in our fragment-based approach.

7 Conclusion and Outlook

In this paper we presented a new flexibility approach for processes. We motivated our approach with a scenario where the process knowledge is local and

fragmentary and needs to be composed to receive a more complete view on the whole process. We presented a fragment definition including some correctness criteria to be able to represent 'local' knowledge. Based on fragment definition we defined a composition operation, which composes two fragments based on knowledge redundancies of both fragments. Lots of work towards a realization of our vision remains to be done, e.g. more composition operations, their formal specification and a formal fragment metamodel needs to be defined, building together a process fragment algebra. Tools supporting the fragment concept and an execution environment need to be built to compose fragments and execute fragments. The question on how fragments can be annotated to enable a automatic fragment selection for the next composition step will be part of the ongoing work.

References

1. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 291–308. Springer, Heidelberg (2006)
2. Courbis, C., Finkelstein, A.: Towards an Aspect Weaving BPEL Engine. In: Coady, Y., Lorenz, D.H. (eds.) Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software 2004, Lancaster, United Kingdom (March 2004)
3. Hallerbach, A., Bauer, T., Reichert, M.: Issues in Modeling Process Variants with Provop. In: BPD 2008, pp. 54–65 (2009)
4. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison Wesley, Reading (2000)
5. Karastoyanova, D., Leymann, F., Nitzsche, J., Wetzstein, B., Wutke, D.: Parameterized BPEL Processes: Concepts and Implementation. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 471–476. Springer, Heidelberg (2006)
6. Leymann, F., Roller, D.: Production Workflow - Concepts and Techniques. PTR Prentice Hall, Englewood Cliffs (2000)
7. Ma, Z., Leymann, F.: BPEL Fragments for Modularized Reuse in Modeling BPEL Processes. In: ICNS 2009, pp. 1–6. IEEE Computer Society, Los Alamitos (2009)
8. Martin, D., et al.: OWL-S: Semantic Markup for Web Services (2004), `http://www.w3.org/Submission/OWL-S/`
9. Reichert, M., Rinderle-Ma, S., Dadam, P.: Flexibility in Process-Aware Information Systems. In: Petri Nets 2009, vol. 2, pp. 115–135 (2009)