Change Propagation in Decentralized Composite Web Services

Walid Fdhila, Aymen Baouab, Karim Dahman,
Claude Godart, Olivier Perrin, and François Charoy
LORIA - INRIA - UMR 7503
BP 239, F-54506 Vandœuvre-lès-Nancy Cedex, France
{firstname.lastname}@loria.fr

***Abstract*—Every company wants to improve the way it does business, or produce things more efficiently, and make greater profit. Therefore, business processes have become subject to evolutionary changes, which in turn increase the need for an efficient change support. In this sense, many researches were conducted to deal with business process adaptation to changes. The latter may result in the restructuring of the whole or a part of the process. Most of the proposed approaches focus on adaptation to changes in centralized processes. In sharp contrast to these works, our operation of change adaptation that we present in this paper, concerns decentralized orchestrations. Indeed, many recent approaches were proposed to decompose a composite web service into small partitions. Since the activities, the control and data flows are distributed over these partitions, it becomes difficult to specify the changes directly. Moreover, changing a derived partition may affect the way it interacts with others. In order to overcome these deficiencies, we propose a design-time methodology to support changes in decentralized business processes. We mainly demonstrate how to propagate the changes made on a centralized specification of composite web service to its resulting decentralized sub-processes.**

***Keywords*-decentralized orchestrations, business process, change propagation, web service.**

I. Introduction

Recently, many approaches were proposed to partition business processes [1], [2], [3]. The partitioning transforms the centralized process into behaviorally equivalent distributed sub-processes. These partitions are executed independently at distributed locations and can be invoked remotely. They directly interact with each other using asynchronous messaging without any centralized control. The flexibility introduced by the derived decentralized processes on the other hand raises new requirements like adaptation to change. Indeed, in today's dynamic business world, the economic success of an enterprise increasingly depends on its ability to react to changes within its environment in a quick and flexible way [4]. Changes may range from simple modifications of the process to a complete restructuring of the business process to improve efficiency. In the context of the decentralized service orchestrations, applying these changes in a straightforward manner on the derived partitions is a complex maintenance task, since the control and data flows are decomposed over multiple partitions. In this sense, this paper presents a method for adaptation to change in the decentralized composite web services. In sharp contrast to previous works [5], [6], our change adaptation concerns decentralized orchestrations. Given a well-behaved structural update on a centralized orchestration, our approach *automates the change forward propagation* that consistently propagates the update to the derived decentralized partitions. This includes the identification of the partitions concerned by the modification, and the incorporation of the necessary changes in each of them. The main advantage of this method, is that only concerned partitions by the change are affected, and there is no need to recompute the whole decentralization or redeploy all the partitions.

This paper is structured as follows. Section II presents the formal definitions needed to provide a generic approach, while section III details our adaptation to change mechanism for the decentralized processes. In sections IV and V we discuss the related work, summarize the contribution and outline future directions.

II. Formal Model

In order to provide a generic approach for change adaptation in the decentralized business processes, we adopt a high level reasoning using an abstract notation. The composite web services are generally captured by means of an orchestration model: a process model in which each activity represents either an intermediate work step (e.g. a data transformation) or an interaction with one of the services participating in the composition (the *component services*). The process model specifies the control-flow and data-flow relations between activities, using a specialized language such as the Business Process Execution Language (WS-BPEL) or the Business Process Modeling Notation (BPMN).

Definition 1 (Process): Formally, a process $\mathcal{P}$ is a tuple ($\mathcal{O}$, $\mathcal{D}$, $\mathcal{E}c$, $\mathcal{E}d$, $\mathcal{S}$) where

- $\mathcal{O}$ is a set of objects which can be partitioned into disjoint sets of activities $\mathcal{A}$, events ($start$ and end) and control patterns $\mathcal{CTR}$ (AND,XOR,Sequence, etc),
- $\mathcal{D}$ is a set of data,
- $\mathcal{E}c$ is a set of control edges where, $\mathcal{E}c \subset \mathcal{O}\times\mathcal{O}$.
- $\mathcal{E}d$ is a a set of data edges where, $\mathcal{E}d \subset \mathcal{A}\times\mathcal{A}\times\mathcal{D}$,
- $\mathcal{S}$ is the set of services invoked by the process.

COLLABORATECOM 2011, October 15-18, Orlando, United States

DOI 10.4108/collaboratecom.2011.28

A process activity consists of a one-way or a bidirectional interaction with a service via the invocation of one of its operations. The set of activities that refer to the same service s is denoted $\mathcal{A}_s \mid s \in \mathcal{S}$. We define the $preset$ ($postset$) of an activity a_i, denoted $\bullet a_i$ ($a_i\bullet$), as the set of activities which may execute **just** before (after) a_i and related to it by a set of control dependencies. In this paper, we consider that the processes are structured [7].

The partitioning of a composite web service, leads to a set of interconnected partitions, each of which defines the relationship between the objects it includes. Each partition communicates with other partitions using the interaction patterns (i.e. send, receive..) [8].

Definition 2 (Partition): A sub-process or a partition is a tuple P_s = ($\mathcal{O}_s$, $\mathcal{D}_s$, $\mathcal{E}c_s$, $\mathcal{E}d_s$) where

- $\mathcal{O}_s$ is a set of objects of P_s. $\mathcal{O}_s \subset \mathcal{O} \cup \mathcal{A}_{dummy(i)}$ where $\mathcal{A}_{dummy(s)}$ is a set of dummy activities. Dummy activity is an activity with zero execution time (used for synchronization).
- $\mathcal{D}_s \subset \mathcal{D} \cup \mathcal{S}ync$, where $\mathcal{S}ync$ is a set of control data necessary for synchronization with other partitions.
- $\mathcal{E}c_s$ is the set of control edges, $\mathcal{E}c \subset \mathcal{O}_s \times \mathcal{O}_s$,
- $\mathcal{E}d_s$ is the set of data edges, $\mathcal{E}d \subset \mathcal{A}_s \times \mathcal{A} \times \mathcal{D}_s$. (control edges between partitions are transformed to data edges since they are routed in messages).
- $s \in \mathcal{S}$ is the set of services invoked by the partition.

Next, we define the transitive *postset* (resp., transitive *preset*) of an activity a_i denoted $T\_a_i\bullet$ ($\bullet T\_a_i$), as the set of activities in the same partition as a_i, which may execute just after (before) it, and linked to it by a set of control dependencies.

III. Change propagation

In this section, we present our methodology for decentralized business processes adaptation to change. We remind that our approach concerns only already partitioned processes. This means that we do not seek to provide a change support for a centralized process. Instead, we demonstrate how to propagate the changes made on a centralized specification of a composite web service to its resulting decentralized sub-processes. The approach is structured as follows. First the designer specifies the changes using the centralized process specification, then we compute the new configurations of the decentralized fragments enclosing the changes. Finally, we propagate the changes to the concerned partitions. In this way, only the fragments which are concerned by the changes would be affected. Moreover, there is no need to re-partition the centralized process and re-deploy all the derived partitions.

A. Change operations

In general, process models can be decomposed into SESE fragments [9]. A SESE fragment is a non-empty subgraph in the process model with a single entry and a single exit edge. For every change in the process model, there is at least one enclosing fragment. Here, we consider only the smallest fragment that encloses the changes. This can be achieved using the process structure tree (PST) [9]. In the following, we consider that the fragments enclosing the changes are already identified (the identification issue is out of scope of this paper). Formally, a fragment has the same definition as a process (c.f. definition 1), except it has no start and end events, instead it has one entry and one exit edges. The changes that can be made on a process model can be resumed using three formal operations as follows:

- ***Insert(fragment, entry, exit)***: this operation is used to insert a new fragment into the process. This fragment should be inserted between the $entry$ and $exit$ edges in the centralized process model.
- ***Delete(entry,exit)***: this operation is used for the deletion of the fragment between the $entry$ and $exit$ edges in the centralized process model.
- ***Update(fragment,entry,exit)***: this operation updates the existing fragment between $entry$ and $exit$ edges in the centralized process model, and replace it by $fragment$. This operation can also be replaced by the two consecutive operations $delete(entry, exit)$ and $insert(fragment, entry, exit)$.

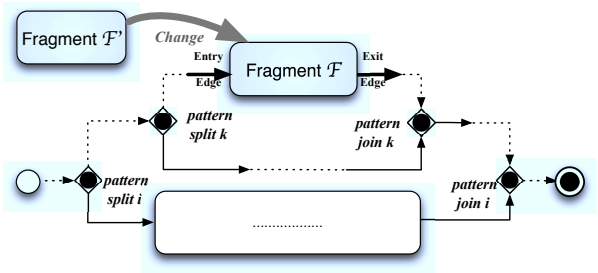

Figure 1. generic process example

B. Change adaptation

This section describes the different steps to propagate the changes made on the centralized process model to the derived decentralized partitions. To have a better understanding, we consider a general example of a centralized process model $\mathcal{P}$, as depicted in Figure 1. The process model is structured through split and join patterns, and enclosed with $start$ and end events. The partitioning of this process, results in $|S|$ interconnected partitions P_I, P_J, P_K, etc, each of which executed by a separate orchestrator (c.f. Figure 2).

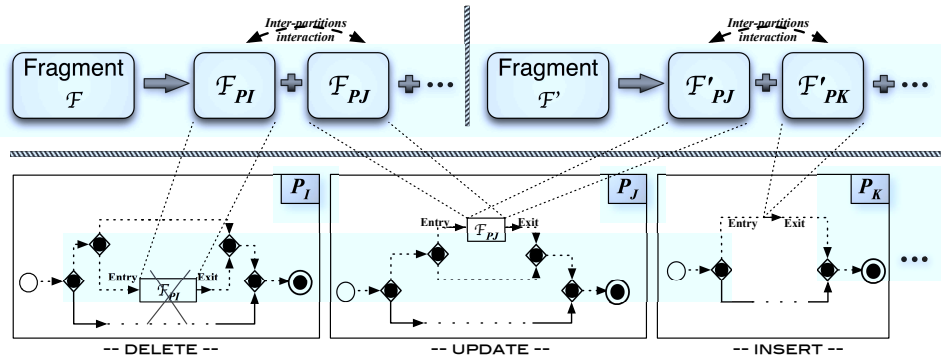

Figure 2. generic example for change management

Now we assume that the user want to replace the fragment $\mathcal{F}$ in the centralized process by the fragment $\mathcal{F}\prime$. For this purpose, we first identify the partitions affected by this change using activities identifiers. Then, we identify the blocs of activities to change inside the affected partitions. Indeed, a simple change in a partition may result in other modifications (including interactions with other partitions or control patterns). For each identified fragment, we notice the entry and the exit edges. Two use cases are possible: the fragment is updated and replaced by another fragment, or deleted. Using the new fragment $\mathcal{F}\prime$, the next step is to determine what to insert in each affected partition. For this purpose, we have to partition $\mathcal{F}\prime$.The steps toward change propagation are as follows:

1- Change specification: the designer specifies the changes to do using the operations: Insert, Delete and Update. If the operation is a delete, then he has to indicate the concerned fragment (i.e. in figure 1 $Delete(Entry_{edge}, Exit_{edge})$). If the operation is an Insert, then he has to specify the fragment to add and in which place in the process model (i.e. in figure 1 $Insert(\mathcal{F}\prime, Entry_{edge}, Exit_{edge})$). Otherwise, he has to specify both the fragment to update and the new fragment to insert (i.e. in Figure 1 $Update(\mathcal{F}\prime, \mathcal{F}.entry, \mathcal{F}.exit)$).

2- Partitions identification: Using the fragments $\mathcal{F}$ and $\mathcal{F}\prime$, we identify all the partitions that would be affected by the change. Indeed, during partitioning, each activity is assigned to a partition upon to a certain criteria. If the activity responds to the criteria of the partition then it would be assigned to it (i.e. activities having the same role, or invoking the same service). So, using the criteria assigned to each activity we can determine the partition it would belong to. By this way, each partition having a criteria of one of $\mathcal{F}$ or $\mathcal{F}\prime$ activities would be affected by the change.

3- Fragments partitioning: this step consists in decentralizing separately the fragments $\mathcal{F}\prime$ and $\mathcal{F}$ into interconnected sub-fragments, using partitioning techniques for structured processes. In figure 2, we take into consideration only the sub-fragments $\mathcal{F}_{PJ}\prime$, $\mathcal{F}_{PK}\prime$, $\mathcal{F}_{PI}$ and $\mathcal{F}_{PJ}$, since they cover the three possible scenarios: insert, delete or update a sub-fragment into a partition.

4- Change translation: After $\mathcal{F}$ and $\mathcal{F}\prime$ partitioning, change operation for the process model is decomposed into one or more change operations. Each operation represents the change to make on the corresponding partition. The generic formula for operation transformation is $operation(x, y, ..) \Rightarrow operation_1(x_1, y_1, ..) \wedge ... \wedge operation_2(x_2, y_2, ...)$, where $operation_i$ is the change to apply to partition P_i. For instance, the generated change operations on partitions P_I, P_J and P_K are as follows:

$Update_P(\mathcal{F}\prime, \mathcal{F}.entry, \mathcal{F}.exit) \Rightarrow Update_{PJ}(\mathcal{F}\prime_{PJ}, \mathcal{F}_{PJ}.entry, \mathcal{F}_{PJ}.exit) \wedge Delete_{PI}(\mathcal{F}_{PI}.entry, \mathcal{F}_{PI}.exit) \wedge Insert_{PK}(\mathcal{F}\prime_{PK}, entry, exit)$.

5- Partitions adaptation to change: This step consists in applying the changes to the corresponding partitions. For this purpose, we first, have to determine exactly where to insert the sub-fragments $\mathcal{F}_{PJ}\prime$, $\mathcal{F}_{PK}\prime$. The first scenario related to the update of $\mathcal{F}_{PJ}\prime$ in P_J is simple, since we already know the entry and exit edges of $\mathcal{F}_{PJ}$. So, we have just to look for these edges in the partition and replace all the fragment between them by the fragment $\mathcal{F}_{PJ}\prime$. The deletion of the latter, implies the deletion of all the interactions with other activities in the same partition or other partitions. The partitions which interact with any activity concerned by the change are also concerned by the change, since we have to update its corresponding interaction edges. Formally, the update of $\mathcal{F}_{PJ}$ in a P_J by $\mathcal{F}_{PJ}\prime$ corresponds to the deletion of all objects $o \in \mathcal{O}_{\mathcal{F}_{PJ}}$, edges $e \in \mathcal{E}c_{\mathcal{F}_{PJ}} \cup \mathcal{E}d_{\mathcal{F}_{PJ}}$, and data, and their substitution by the objects, edges, and data of $\mathcal{F}\prime_{PJ}$.

The Delete operation is similar to the update, except that we do not insert a new sub-fragment. We simply look for the entry and exit of $\mathcal{F}_{PI}$ in the partition P_I. Then we delete the sub-fragment between them. If the entry edge of the sub-fragment to delete, is linked to a (choice or parallel) split

control patterns (outside the sub-fragment), and the exit edge is linked to its corresponding join element, then we look if the other branches linking these two elements include only dummy activities or not. If yes then we delete these two patterns. We iterate this operation on each nested constructs linking the sub-fragment $\mathcal{F}_{PI}$ to its transitive $preset\ \bullet T_{\mathcal{F}_{PI}}$ and $postset\ T_{\mathcal{F}_{PI}}\bullet$ (we extend the definition of transitive postset (transitive preset) to that between a fragment and its subsequent (previous) activities). Otherwise, we replace the sub-fragment to delete, by a dummy activity.

Now, to insert sub-fragment $\mathcal{F}_{K'}$ in the partition P_K, we have to identify the $entry$ and $exit$ edges. For this purpose, we first compute the transitive preset and postset of $\mathcal{F}_{K'}$ in P_K ($\bullet\mathcal{F}_{K'}$, $\mathcal{F}_{K'}\bullet$). Then, we identify all the control patterns that link them in the centralized process model. Next, we identify each split pattern ctr in this control path linking it to its $\mathcal{F}_{K'}\bullet$, such as $\overline{ctr}$ is in the path linking it to its $\bullet\mathcal{F}'_K$ ($\overline{ctr}$ is the correspondent join element of ctr). For each *choice* ctr found, we look if it already exists in the partition. If yes, we just add a new branch linking ctr to $\overline{ctr}$ in which we put $\mathcal{F}'_K$. If no, we add it and its corresponding $\overline{ctr}$, then we put the $\mathcal{F}'_K$ between them (in parallel with a dummy activity). In some cases, the update or the insertion of a fragment may result in the creation of a new partition or the deletion of an existing partition.

Due to lack of space, we do not present in this paper the formal algorithm which resumes all the adaptation to change steps.

IV. RELATED WORK

Several issues related to change management have been addressed in business process management and workflow literature. For instance, the ADEPT proposal enables controlled changes at the process type as well as the process instance level [10]. In [5], authors present important issues related to process changes and discuss organizational structures. In [6], the authors motivate the need for the controlled change of organizational models and present different adaptations models to be supported by respective components. In [11], the authors describe the Epsilon merging language used to specify how models are merged. In [12], the authors propose a metamodel for the specification and detection of syntactical and semantical conflicts. All the mentioned approaches, address change adaptation in a centralized process. They also deal with how to dynamically adapt running instances to changes. This, may be complementary to our work.

In the decentralized setting, [13] presents a formal model for a distributed workflow change management (DWFCM) that uses a rule-topic ontology and a service ontology to support the needed run-time flexibilty. This work is different from our proposal, since they do not seek to propagate a pre-defined changes on a centralized process to that on the derived partitions. Their work is more focused on run-time adaptation using the migration rules. In [14] the authors present a unidirectional model incremental transformation approach. The aim of the work is the definition and the realization of an automatic synchronizer for managing and re-establishing the structural consistency of heterogeneous source and target models.

V. CONCLUSION

In this paper, we have presented an approach to adapt decentralized orchestrations to changes specified on the corresponding centralized process. The proposed approach is based on three change patterns $Insert$, $Update$ and $Delete$. The method consists in partitioning the fragment to change into sub-fragments, which in turn, are integrated into the corresponding partitions. To the best of our knowledge, this is the first work that takes on changes adaptation in decentralized composite web services.

REFERENCES

[1] W. Fdhila, U. Yildiz, and C. Godart, "A flexible approach for automatic process decentralization using dependency tables," in *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*. Los Angeles, CA, USA: IEEE Computer Society, 2009, pp. 847–855.

[2] R. Khalaf and F. Leymann, "E role-based decomposition of business processes using bpel," in *ICWS*, 2006, pp. 770–780.

[3] W. Fdhila, M. Dumas, and C. Godart, "Optimized decentralization of composite web services," in *CollaborateCom 2010, 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 11-14 2010, pp. 1 –10.

[4] M. Hammer and S. A. Stanton, *The reengineering revolution: A handbook*. New York: HarperBusiness, 1995.

[5] M. Pesic, M. H. Schonenberg, N. Sidorova, and W. M. P. van der Aalst, "Constraint-based workflow models: Change made easy," in *OTM Conferences (1)*, 2007, pp. 77–94.

[6] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change patterns and change support features - enhancing flexibility in process-aware information systems," *Data Knowl. Eng.*, vol. 66, no. 3, pp. 438–466, 2008.

[7] B. Kiepuszewski, A. H. M. ter Hofstede, and C. Bussler, "On structured workflow modelling," in *CAiSE*, 2000, pp. 431–445.

[8] A. P. Barros, M. Dumas, and A. H. M. ter Hofstede, "Service interaction patterns," in *Business Process Management*, 2005, pp. 302–318.

[9] J. Vanhatalo, H. Völzer, and F. Leymann, "Faster and more focused control-flow analysis for business process models through sese decomposition," in *ICSOC*, 2007, pp. 43–55.

[10] M. Reichert and P. Dadam, "Adept$_{\text{flex}}$-supporting dynamic changes of workflows without losing control," *J. Intell. Inf. Syst.*, vol. 10, no. 2, pp. 93–129, 1998.

[11] D. S. Kolovos, R. F. Paige, and F. Polack, "Merging models with the epsilon merging language (eml)," in *MoDELS*, 2006, pp. 215–229.

[12] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe, "Algebraic approaches to graph transformation - part i: Basic concepts and double pushout approach," in *Handbook of Graph Grammars*, 1997, pp. 163–246.

[13] V. Atluri and S. A. Chun, "Handling dynamic changes in decentralized workflow execution environments," in *DEXA*, 2003, pp. 813–825.

[14] K. Dahman, F. Charoy, and C. Godart, "Towards consistency management for a business-driven development of soa," in *The 15th IEEE International Enterprise Distributed Object Computing Conference*, Helsinki, Finland, 2011.