HAL
open science

Adaptation Patterns for Service-Based Inter-Organizational Workflows

Saida Boukhedouma, Mourad Chabane Oussalah, Zaia Alimazighi, Dalila Tamzalit

To cite this version:

Saida Boukhedouma, Mourad Chabane Oussalah, Zaia Alimazighi, Dalila Tamzalit. Adaptation Patterns for Service-Based Inter-Organizational Workflows. 7th IEEE Conference on Research Challenges in Information Science, May 2013, Paris, France. pp.567-576. hal-01063854

HAL Id: hal-01063854

https://hal.science/hal-01063854

Submitted on 24 Sep 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Adaptation Patterns for Service Based Inter-Organizational Workflows

Saida Boukhedouma[1,2], Mourad Oussalah[2]
[1] USTHB University – Department of Computer science
Algiers, Algeria
{sboukhedouma, zalimazighi}@usthb.dz

Zaia Alimazighi[1], Dalila Tamzalit[2]
[2] University of Nantes
Nantes, France
{mourad.oussalah, dalila.tamzalit}@univ-nantes.fr

***Abstract*— The SOA (Service Oriented Architecture) paradigm provides important advantages like interoperability, reusability and flexibility required in the area of business applications. In our research works, we focus on the use of SOA to implement inter-organizational workflows (IOWF). Our goal is to obtain IOWF models flexible enough in order to ease their adaptation at build-time and at runtime, because services are loosely coupled components, easily invoked and interoperable. This paper focuses on specific and well common IOWF-architectures defined in the literature; it deals with adaptation of IOWF process models obeying to these architectures. First, we define the concept of Service-Based Cooperation Pattern (SBCP) that supports service-based IOWF models meeting one of the specific architectures considered. Then, we state a set of recurrent operations of adaptation (attached to process and interaction aspects) that can be applied on service-based IOWF models, and we illustrate their implementation for IOWF models specified with BPEL.**

Keywords—IOWF, SOA, Service-Based Cooperation pattern, Orchestration Function, Adaptation Pattern.

I. INTRODUCTION

The concept of B2B (Business to Business) has been promoted with the use of business oriented technologies such as workflow [1] and web services [2] supported by service oriented architectures (SOA) [3]. Since the year 2000, many works deal with the combination of WF and web services to build collaborative business applications suitable to *ad-hoc* cooperation or *structured* cooperation. Ad-hoc cooperation means that the schema of the business process is defined on the fly at runtime and process instances don't necessarily follow the same process model. In structured cooperation, the steps of the business process and interactions in the system are well defined resulting in an IOWF model clearly defined; so all process instances should follow the IOWF model implemented.

In our research works, we are interested in structured cooperation supported by the concept of inter-organizational workflow (IOWF). In [4] [5], generic architectures of IOWF have been defined to support structured cooperation. These architectures are the *capacity sharing*, the *chained execution*, the *subcontracting*, the *case transfer*, the *extended case transfer* and the *loosely coupled WF*; we consider them as basis of cooperation models between businesses because they cover a wide range of existing business processes since they express the different ways in which businesses can cooperate together. However in their initial form, these architectures were subject to criticisms because of their rigidity and the difficulty to adapt business processes to support changes [6].

Due to internal and external events and new market constraints, businesses should continually or occasionally adapt their business processes. So, the final objective of our research works is to propose mechanisms providing *flexibility* of IOWF process models suitable to structured cooperation. We define flexibility around three main axes which are *adaptability*, *evolutivity* and *reusability*.

But before we get to deal with flexibility, we should have process models which are flexible enough to support *adaptation, evolution and reuse*. For that, we use a SOA-based approach to define Service-Based Cooperation Patterns (SBCP) corresponding to the basic architectures defined in [4] [5].

The main issue of the current work is to deal with the first aspect of flexibility which is the adaptability of IOWF process models. Then, we describe our framework of adaptation composed by a set of adaptation patterns that we have implemented for IOWF models specified with BPEL. We focus on process (functional and behavioral) and interactional perspectives.

However, in order to ease the comprehension of the paper and to make it self-containing, we introduce in a generic manner, the concept of SBCP. So, we propose a generic meta-model for SBCP in order to exhibit the main concepts for IOWF definition using SOA-based approach. We state that the basic IOWF-architectures considered can be implemented through *global orchestration* of services in case of centralized or hierarchized control or *distributed local orchestrations* of services in case of decentralized control, according to constraints relative to each architecture. The main questions that we had to answer are: how to structure workflows implied in cooperation into services in order to meet a specific IOWF-architecture? What is the appropriate type of control? How to define interactions between services provided by different partners?

The rest of the paper is structured as follows: Section II explains the motivations of our research. Section III presents some related works attached on one hand, to IOWF approaches and on the other hand, to patterns-based approaches. Section IV synthesizes the necessary background to understand the paper such IOWF process definition concepts and aspects of

flexibility of IOWF models. Section V lays the basis of our approach for WF interconnection using services; here, we explain the concepts of SBCP and orchestration function. Section VI describes the basic adaptation patterns proposed. Section VII gives some implementation details and section VIII summarizes the paper and talks about future works.

II. MOTIVATIONS

The need of flexibility in IOWF processes comes from business constraints like new market demands, strategic changes of organizations, the need of additional resources and competencies and from technical constraints due to technological evolutions. These constraints and others force businesses to review their processes in order to make the necessary adjustments using the adaptation mechanisms offered.

Globally in our research works, we set two main objectives: the first one is to propose a set of generic cooperation patterns supporting flexible inter-organizational process models. These models correspond to fairly common IOWF-architectures defined in the literature [4], [5] covering a large number of existing processes. The cooperation patterns that we propose are based on SOA paradigm and called Service-Based Cooperation Patterns (SBCP); a service which is the central concept of SOA correspond to a loosely coupled and platform independent component. The second objective of our research works is to implement frameworks of adaptation, evolution and composition (for reuse) patterns that can be applied on the IOWF obeying to the SBCP proposed.

The idea of using services to build collaborative business applications is not new. The motivations behind this come from three main points: (1) the relevance of service orientation, (2) the benefits of service orientation for the information system and (3) the benefits of service orientation for the cooperation. For the first point, the concept of service (particularly web services) provides credible answers to constraints and problems attached to the information system like the luck of flexibility, the reluctance to openness and those attached to the cooperation like the need to preserve the autonomy and the confidentiality [7]. For the second point, the service-based approach provides a certain degree of flexibility to the information system by easing the participation in new business opportunities and meeting new market demands. For the third point, the cooperation between business partners is realized by service composition [7]. Then, businesses provide their services with a certain degree of abstraction by publishing them through their interfaces; this allows preservation of autonomy and confidentiality.

Also, for adaptation, we use a pattern-based approach in order to enumerate the adaptations that can occur repeatedly in IOWF processes. This allows modular and reusable implementation of the proposed patterns starting with elementary patterns and going to more complex ones by reuse of the first ones.

III. RELATED WORKS

With the emergence of SOA and web services standards, many research works deal with orchestration and choreography of web services [8], [9], especially based on BPEL4WS [10] in order to build business processes by service composition.

Other research works such as [11], [12] show the interest of combining BPM (business process management), workflow and SOA for the re-use of services to construct dynamic business processes. This had a great impact in promoting B2B relationships since several approaches and platforms have been developed to support the B2B cooperation using WF and SOA. In *structured* cooperation for example, we can cite some approaches like CoopFlow [6], CrossFlow [13], CrossWork [14], Pyros [15] and e-Flow [16].

Also, flexibility is an important property to be satisfied by business processes and their systems allowing them to support changes. Even if some approaches like CoopFlow, Pyros and e-Flow provide *internal adaptation* of workflows without compromising the coherence of the global process, a large number of the proposed solutions are not flexible enough because they are closely coupled with the platforms. So for any changes, they impose to re-adapt the interfaces and to newly build the structure of interaction. Moreover, WF flexibility is perceived at two complementary levels: (1) at the *system level*, the flexibility defines the ability of a WFMS (WF management system) to face unexpected and erroneous situations [17], [18]. (2) at the *level of process models* that defines the ability of a process model to be adaptable, evolvable and reusable; many research works have been proposed describing different techniques such as adaptation patterns [19], [20], [21], rule-based adaptation patterns [22], [23] and constraint-based modeling [24] to support flexibility of process models. For example, in [21], the authors identify the most important process change patterns and change features for PAIS (process aware information systems). In [25], a framework was described using adaptation patterns and aspect–programming in order to support process adaptation for BPEL engines.

The concept of pattern was initially used in software engineering as the abstraction from a concrete form which keeps recurring in specific non-arbitrary context. In the workflow area, this concept has been usually used for business process modeling [26], business process improvement or changes [21], [25] or exception handling [27].

In this paper, we describe our framework of adaptation composed by a set of adaptation patterns that can be applied on IOWF process models specified with BPEL. Thanks to its modularity, our framework of adaptation is easily maintainable and extensible. The process models considered obey to specific IOWF-architectures which are in turn implemented according to a set of service-based cooperation patterns (SBCP). A SBCP is a concept that we define using three main dimensions: service, orchestration function and interaction.

IV. Basic Definitions and Concepts

A. IOWF Definition

An IOWF can be defined as a manager of activities involving two or more workflows *autonomous*, possibly *heterogeneous* and *interoperable* in order to achieve a common business goal [4].

B. IOWF Architectures

In [4][5], generic architectures of IOWF have been defined in order to support structured cooperation which must obey, depending on the needs of partners, to a schema clearly defined. These architectures are the *capacity sharing*, the *chained execution*, the *subcontracting*, the *case transfer*, the *extended case transfer* and the *loosely coupled WF*. These architectures have been characterized according to two main dimensions: the *partitioning of the process* and the *control of execution*. Regarding to the first dimension, two types of partitioning are distinguished: *process schema partitioning* and *instance partitioning*. Process schema partitioning means that the IOWF process model is implemented as fragments at the partner's sites. Instance partitioning means that the execution of a process instance is distributed among the different sites in a disjoint manner (at each moment, an instance is located at one site).

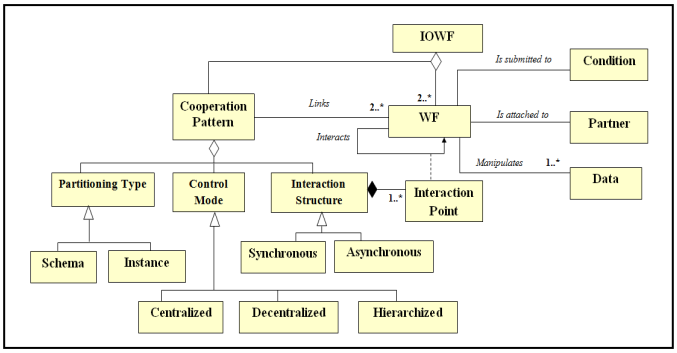
Since IOWF are distributed systems, the control of instance execution can be *centralized*, *decentralized* or *hierarchized*. The control is *centralized* if the execution of process instances is delegated to one system that also manages all interactions between the systems of partners; this is suitable for the *capacity sharing*. The control is *decentralized* if the execution of instances is distributed among the systems of all partners and each system manages itself its interactions with the other systems, this is appropriate for the *chained execution*, the *loosely coupled* and for the *(extended)case transfer* architectures. We say that a control is *hierarchized* if each system manages its own WF and there is one principal system that controls interactions with one or more secondary systems, this is suitable for the *subcontracting* architecture.

In the following, we exhibit the main concepts of IOWF process definition and we introduce the concept of cooperation pattern that we define using three main dimensions: *partitioning*, *control* and *interaction.*

C. IOWF Meta-model

Fig. 1 bellow shows a meta-model of IOWF process definition, we can see that an IOWF process model is defined by a set of *WFs* (fragments of the global IOWF) and a *cooperation pattern*. Each WF is attached to a *partner*, manipulates *data* and is submitted to *condition* of invocation. A given cooperation pattern is attached to a specific architecture of IOWF; it links two or more workflows and is defined around three main dimensions: the partitioning of the process, the control of execution and the structure of interaction. This last is defined by a set of interaction points between WF fragments. Intuitively a cooperation pattern defines the manner in which WF fragments are distributed among the partner's sites, how the execution of instances is managed and how WF fragments interact together.

Fig. 1. Meta-model of IOWF Process Definition



D. Flexibility of IOWF Models

As already evoked in the introduction, the environment of businesses and the business processes describing their behavior are naturally dynamic, because they are continually submitted to new market constraints and unexpected events. Indeed, a business process is perpetually subject to changes calling into question its structure and its validity. So, a business process should be flexible enough in order to support these changes.

Through the concepts exhibited on the meta-model (see Fig. 1.), we can see that an IOWF model covers four main axes: *process* (concepts of IOWF, WF, condition, cooperation pattern, partitioning type and control mode), *organization* (concept of partner), *data* and *interaction* (concepts of interaction structure and interaction point*)*. Consequently, we can affirm that the constraints of flexibility in IOWF models are not limited to one axis, but cover the four axes.

Also, we perceive the flexibility of process models through three main perspectives: adaptability, evolutivity and reusability that we define as follows:

The *adaptability* of an IOWF process model defines its capacity to easily support changes while maintaining the coherence of the process after changes, the overall functionality and the cooperation (the set of partners). Hence, an IOWF model is *adaptable* if one or more of the entities (WF, condition, data, interaction points) composing it can be modified without affecting the global functionality of the process and the cooperation.

The *evolutivity (*called *evolutive adaptability)* of an IOWF process model is its capacity to accept *expansion* of its global *functionality* and/or expansion of *cooperation* inducing additional business partners and so additional WF fragments where maintaining the coherence of the process, we say that the IOWF model is *evolvable.*

The *reusability* of a model defines its capacity to be easily integrated with another model in order to build more and more complex models. Then, an IOWF model is *reusable* if it can be manipulated as a separate entity (*IOWF*) and to be integrated to other models in order to build more complex IOWF processes which cover more functionalities and services.

Let's notice that in our work, we focus on flexibility reflected at process and interaction axes (although it involves and also draws on other levels – data and organization).

In order to make the paper self-containing, we introduce the following section to explain the basis of our approach of WF interconnection using the SOA concepts.

V. Basis of our Approach

In our previous works [28] [29] [30], we have considered each basic IOWF-architecture and we have defined a corresponding SBCP in order to deal with IOWF models flexible enough to ease their adaptation. The main idea of our approach is to encapsulate each WF fragment into a single (composite) service or a set of services depending on the IOWF-architecture to meet.

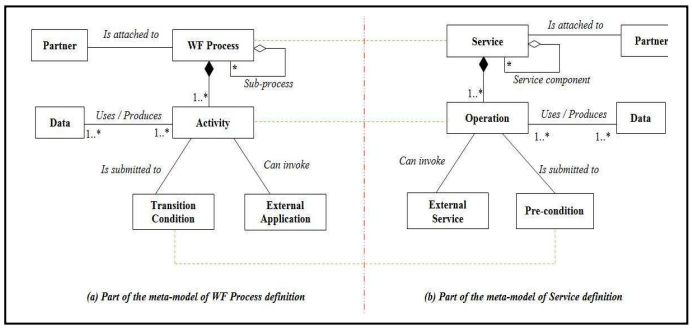
A. *Encapsulation of a WF Process into Services*

The encapsulation of a WF process (or a sub-process) into a service is possible due to conceptual and technical similarities between the concept of WF and the concept of service. Fig. 2 exhibits these conceptual similarities.

- *Conceptual Aspects*

A WF process is attached to a business partner, as a business service. A service is eventually composed by other services (components), in the same manner a WF process is eventually composed by sub-processes having the same structure as the global WF. At the lower level of decomposition, a WF process is hierachized into activities; an activity uses/produces data, it is submitted to a transition condition and can invoke external applications. Also, a service is hierarchized into operations (activities); each operation uses/produces data, it is submitted to a pre-condition (analog to transition condition) and can invoke external services (applications).

Fig. 2. Correspondence of Concepts – WF vs Service



In addition, a WF process covers a global business functionality that can be decomposed into sub-functionalities performed by sub-processes. Service in turn, has a global business functionality that can be decomposed into sub-functionalities performed by the service components. Therefore, we can say that a WF process is conceptually similar to a business oriented service.

- *Technical Aspects*

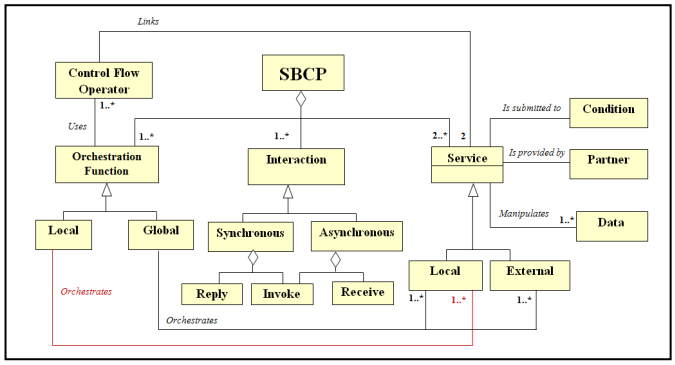
Technically, a service has an interface and a description allowing its invocation in accordance with syntactic, semantic and QoS constraints. Similarly, a WF has a description and an interface (set of API) for its invocation from another WF through the interface 4 of the reference model proposed by the WFMC coalition. Thus a WF process (or sub-process) can be considered as a business service performing a well defined functionality and that should be invoked through an interface, under some constraints. Hence the idea of encapsulating a WF process in a service which is a loosely coupled, interoperable and platform independent component.

B. *Service Based Cooperation Pattern (SBCP)*

With our vision, interactions between WF fragments turn to invocation of services provided by several partners. For that, our approach focuses on three main questions: (1) How to *structure* the WF process into services? (2) How to *control* the execution of instances? (3) How to *define interactions* between services provided by different partners? These three questions exhibit three main dimensions that we use to define the concept of SBCP. Here, we define a SBCP in a generic manner (covering all the IOWF-architectures considered) in order to exhibit the main concepts for service-based IOWF definition.

A SBCP is defined by three main dimensions: the distribution of services on the partner's sites, the control of execution and the set of interactions like shown on Fig. 3.

Fig. 3. Meta-model of SBCP Definition



Regarding to the first dimension which is the *distribution of services*, we consider that each service encapsulates part or all of the WF process and is implemented at the partner's site that provides it. This dimension corresponds to the dimension *Process partitioning* which is defined for the initial IOWF-architectures. From the perspective of a given partner, a service can be implemented locally (local service) or provided by an external partner (external service).

The second dimension which is the *control of execution* (centralized, decentralized or hierarchized) is expressed through the concept of orchestration function that abstracts the structure of the process in terms of control flow and interactions between services composing the IOWF process. Hence, in case of centralized control, there is one global orchestration function implemented at the site of one partner that controls the execution of the whole IOWF. By contrast, in case of decentralized control, there is a set of local

orchestration functions. Each orchestration function is implemented at one partner site and allows the control of the fragment implemented at the same partner site. In case of hierarchized control, there is one global orchestration function that controls the invocation of internal and external services and a set of local orchestration functions that control the execution of secondary workflows implied in the "*subcontracting*" cooperation. The concept of orchestration function is defined and illustrated in section C bellow.

The third dimension defines the interactions between services of several partners implied in the IOWF process. This dimension is expressed via interactional activities (invoke/receive for asynchronous communication and invoke/reply for synchronous communication).

C. Orchestration Function and Control Flow

Like shown on the meta-model of Fig. 3, the concept of *orchestration function* describes the control flow between services composing the IOWF using basic control flow operators.

TABLE I. BASIC CONTROL FLOW OPERATORS

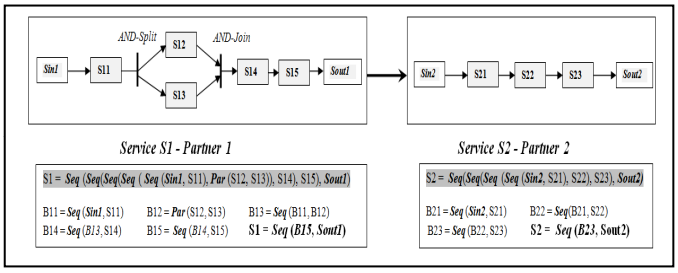
Operator	Schema	Description	Orchestration function
Seq	S1 → S2	*Sequential execution* S1 followed by S2	*Seq* (S1,S2)
Par	AND-Split: S1 → S2, S3	*Simultaneous execution* of S1 and S2	*Par* (S2,S3)
Alt	OR-Split: S1 → S2, S3	*Inclusive choice* between S1 and S2	*Alt* (S2,S3)
Exl	XOR-Split: S1 → S2, S3	Describes an *exclusive choice* of S1 and S2	*Exl* (S2,S3)
	S1, S2 → AND-Join → S3	*Synchronous merge* of S1 and S2 after parallism Expressed using *Par* and *Seq* operators	*Seq* (*Par* (S1,S2), S3)
	S1, S2 → OR-Join → S3	*Simple merge* of S1 and S2 after inclusive choice Expressed using *Alt* and *Seq* operators	*Seq* (*Alt* (S1,S2), S3)
	S1, S2 → XOR-Join → S3	*Simple merge* of S1 and S2 after exclusive choice Expressed using *Exl* and *Seq* operators	*Seq* (*Exl* (S1,S2),S3)

In Table I, we introduce these basic operators and we express them using a general notation independently from any language or platform.

Remark. To describe multi-choice – respectively multi-parallel - (more than two edges), we can decompose on several simple choices – respectively several simple parallel blocks. For example, *Alt* (S1, S2, S3) is expressed as *Alt* (*Alt* (S1, S2), S3) or *Alt* (S1, *Alt* (S2, S3)).

Fig. 4 bellow illustrates the concept of orchestration function using our notation; we give an example of IOWF obeying to the "*chained execution*" *pattern*. The process schema describes an IOWF implying two partners, partner 1 and partner 2 implementing their WFs as services *S1* and *S2* respectively. Partner 1 provides his WF composed by *internal* services *S11, S12, S13, S14, S15* and partner 2 provides his WF composed by internal services *S21, S22* and *S23*. For more readability and less complexity of the orchestration function, we can structure the process fragments into blocks *Bij* of sequential, parallel or alternative services. In a hierarchical manner, a block can be expressed using other blocks. The orchestration function can be represented by a binary tree with two types of nodes: operators and services.

Fig. 4. Example of orchestration functions



VI. ADAPTATION PATTERNS

According to the meta-model of Fig. 3, adaptations of an IOWF process model obeying to a SBCP turn to modifications of the entities composing it that means *services*, *orchestration functions* and/or *interactions*. Then, we classify our adaptation patterns into three main categories: *Service* adaptation patterns, *Control Flow* adaptation patterns and *Interaction* adaptation patterns.

A. Service Adaptation Patterns

These patterns concern the modifications that can be applied on the services composing the IOWF process; these modifications are typically adding, removing, replacing, merging of two services (sequential, parallel or alternative) and decomposing a service into a block of two services expressing sequential, parallel or alternative execution. An adaptation of a service usually induces modification on the *orchestration function* using it or a modification of closely attached attributes like *condition* or *data* (see Fig. 3).

- *Adding, Removing and Substituting Services*

Adding a service is done in order to insert an additional step in the process. The reverse operation of adding is the *removing* of services. For *adding* or *removing* of services, it is to distinguish adding or removing of a service on *one edge* composed by sequential services or in a block composed by *two edges* expressing parallel or alternative execution. Table II describes the basic patterns of *adding* services illustrated by generic process schemas and the corresponding *orchestration functions*. We can see that there are elementary patterns named AP1.1, AP1.2, respectively for adding a new service before or after a service in the process, and there are more

elaborated patterns like AP1.3, AP1.4 and AP1.5 which are implemented using elementary patterns AP1.1 or AP1.2, depending on the location of the service to add. Adding a service in an alternative or a parallel block requires the generation of the appropriate condition using a wizard which is provided to the user of our application.

TABLE II. DESCRIPTION OF "SERVICE ADDING" PATTERNS

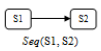
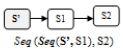
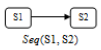
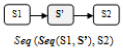
AP1: "Adding Service" Patterns				
Pattern Reference	**Pattern Description**	**Before adaptation**	**After adaptation**	**Patterns used**
AP1.1	Add into *sequence* before a service	 *Seq*(S1, S2)	 *Seq* (*Seq*(**S'**, S1), S2)	**None** (Elementary pattern)
AP1.2	Add into *sequence* after a service	 *Seq*(S1, S2)	 *Seq* (*Seq*(S1, **S'**), S2)	**None** (Elementary pattern)
AP1.3	Add on one edge of *inclusive choice*	*Seq* (*Seq*(S1, ***Alt***(S2, S3)), S4)	*Seq* (*Seq*(S1, *Alt* (**Seq**(S2,**S'**), S3)), S4)	AP1.1 AP1.2
AP1.4	Add on one edge of *exclusive choice*	*Seq* (*Seq*(S1, *Exl* (S2, S3)), S4)	*Seq* (*Seq*(S1, *Exl* (**Seq**(S2,**S'**), S3)), S4)	AP1.1 AP1.2
AP1.5	Add on one edge of *parallel execution*	*Seq* (*Seq*(S1, ***Par*** (S2, S3)), S4)	*Seq* (*Seq*(S1, *Par*(**Seq**(S2,**S'**), S3)), S4)	AP1.1 AP1.2

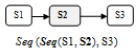
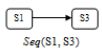
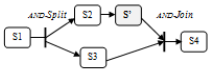
Table III shows typical operations of removing of services (service *S2* for example). Let's notice that two configurations are possible when removing a service *S* from a block with two edges: (1) service *S* is in sequence with other services, (2) service *S* is alone on the edge; this results on two different scenarios for adaptation. These two configurations are represented only for inclusive choice, but they are also considered for exclusive choice and parallel execution. We can see on Table III that AP2.1 is an elementary pattern and AP2.2, AP2.3, AP2.4, AP2.5, … are implemented using AP2.1.

Another basic operation of adaptation concerns the substitution (replacing) of services. This is typically a removing of the service to replace followed by an adding of the new service. Then, the pattern AP3 (called "Service Substitution" Pattern) is implemented using patterns AP1.x and PA2.x for respectively adding and removing, depending on the location in the process schema (in sequence, parallel or alternative) of the service to be replaced. Updating the data flow dependencies between services is done after each operation of adding/removing of services.

- *Fusion and Decomposition of Services*

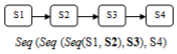
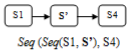
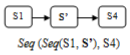
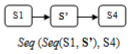
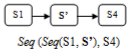
The operation of *fusion* can concern two services linked by a sequence, an inclusive choice, an exclusive choice or a parallel execution, in order to simplify the process model and to abstract several services into one. Table IV bellow describes these basic operations and the corresponding orchestration functions modified after each operation for merging *S2, S3* in a single service *S'*. We can state that since services to merge are in the same block, they become easier to

TABLE III. DESCRIPTION OF "SERVICE REMOVING" PATTERNS

AP2: "Service Removing" Patterns				
Pattern Reference	**Pattern Description**	**Before adaptation**	**After adaptation**	**Patterns used**
AP2.1	Remove from *sequence*	 *Seq* (*Seq*(S1, **S2**), S3)	 *Seq*(S1, S3)	**None** (Elementary pattern)
AP2.2	Remove from one edge with several services of *inclusive choice*	*Seq* (*Seq*(S1, *Alt* (**Seq**(**S2**,S3), S4)), S5)	*Seq* (*Seq*(S1, ***Alt***(S3, S4)), S5)	AP2.1
AP2.3	Remove from one edge with single service of *inclusive choice*	*Seq* (*Seq*(S1, *Alt*(**S2**,S3)), S4)	*Seq* (*Seq*(S1, S3), S4)	AP2.1
AP2.4	Add on one edge of *parallel execution*	 *Seq* (*Seq*(S1, *Par*(**Seq**(S2,**S'**), S3)), S4)	*Seq* (*Seq*(S1, ***Par*** (S2, S3)), S4)	AP2.1

remove and to replace, because the block (*Alt (S2,S3), Par (S2,S3)* or *Exl (S2, S3)*) is considered as a single *composite service* to be replaced. More elaborated operations of fusion concern configurations such as services to merge are not in the same block. For example in a model described by the orchestration function *Seq(Seq(**S1**, Par(**S2**,S3)), S4)*, the operation of merging *S1* and *S2* cannot be done directly since we must know if we maintain the parallelism or we don't maintain it; this information should be provided as additional parameter. In both cases, this must be decomposed into elementary operations of removing and adding of single services or blocks.

TABLE IV. DESCRIPTION OF FUSION PATTERNS

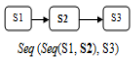
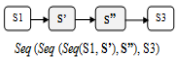
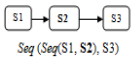
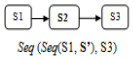
AP4: Fusion Patterns				
Pattern Reference	**Pattern Description**	**Before adaptation**	**After adaptation**	**Patterns used**
AP4.1	Fusion of *sequence*	 *Seq* (*Seq* (*Seq*(S1, **S2**), **S3**), S4)	 *Seq* (*Seq*(S1, **S'**), S4)	AP1.1 AP1.2 AP2.1
AP4.2	Fusion of *inclusive choice*	*Seq* (*Seq*(S1, *Alt* (**S2**, **S3**)), S4)	 *Seq* (*Seq*(S1, **S'**), S4)	AP1.1 AP1.2 AP2.3
AP4.3	Fusion of *exclusive choice*	*Seq* (*Seq*(S1, *Exl* (**S2**, **S3**)), S4)	 *Seq* (*Seq*(S1, **S'**), S4)	AP1.1 AP1.2 AP2.6
AP4.4	Fusion of *parallel execution*	*Seq* (*Seq*(S1, *Par* (**S2**, **S3**)), S4)	 *Seq* (*Seq*(S1, **S'**), S4)	AP1.1 AP1.2 AP2.5

Then, the fusion patterns are implemented using the adding and the removing patterns AP2.5 and AP2.6 which are not represented on Table III, correspond to removing a service from one edge with a single service of parallel execution and of exclusive choice respectively.

The reverse operation of fusion is the *decomposition* of a service to obtain a block of two services that can be

sequential, parallel or alternative block. The decomposition of services can be done to improve the parallelism in the process (parallel decomposition) or to add condition (alternative decomposition) due to new constraints or to have more control on process execution (sequential decomposition).We can see on Table V that the decomposition of a service consists to *remove* a single service (*S2* for example) and to *add a block* composed by two services (*S' and S"*) linked by a sequence, an alternative or a parallel operator. This explains the use of adding patterns AP1.x and removing Patterns AP2.x.

TABLE V. DESCRIPTION OF DECOMPOSITION PATTERNS

AP5: Decomposition Patterns				
Pattern Reference	**Pattern Description**	**Before adaptation**	**After adaptation**	**Patterns used**
AP5.1	Decomposition into *sequence*	 *Seq (Seq*(S1, **S2**), S3)	 *Seq (Seq (Seq*(S1, S'), S"), S3)	AP1.1 AP1.2 AP2.1
AP5.2	Decomposition into *inclusive choice*	 *Seq (Seq*(S1, **S2**), S3)	*Seq (Seq*(S1, *Alt* (S', S")), S3)	AP2.1 AP1.3
AP5.3	Decomposition into *exclusive choice*	*Seq (Seq*(S1, **S2**), S3)	*Seq (Seq*(S1, *Exl*(S', S")), S3)	AP2.1 AP1.4
AP5.4	Decomposition into *parallel execution*	 *Seq (Seq*(S1, S'), S3)	*Seq (Seq*(S1, *Par*(S', S")), S3)	AP2.1 AP1.5

B. *Control Flow Adaptation Patterns*

This category of patterns concerns modification of the control flow between services composing the IOWF process, without affecting the services themselves. This is typically a replacing of an operator of control flow by another; we can replace for example a sequence operator (***seq)*** by parallel operator (***par)*** (parallelization of services) to improve the execution time of process instances, or vice versa (sequentialization of services) if an execution of a service becomes dependant from another service, or alternation of services if an execution of a service depends from a given condition.

Even if there is no modification on services implied in the IOWF, the implementation of the control flow patterns uses other patterns of adding and removing services (see Table VI) because we have to update input and output data of services and also the conditions of invocation.

C. *Interaction Adaptation Patterns*

This category of patterns concerns modification of the interactions between services composing the IOWF process and provided by different partners. Specifically, updating the structure of interaction is done by adding, removing or updating *interactional points* (see table VII).

TABLE VI. DESCRIPTION OF "CONTROL FLOW" ADAPTATION PATTERNS

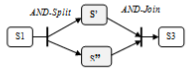
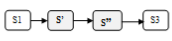
AP6: "Control Flow" Adaptation Patterns				
Pattern Reference	**Pattern Description**	**Before adaptation**	**After adaptation**	**Patterns used**
AP6.1	Sequentialization of services	 *Seq (Seq*(S1, *Par*(S', S")), S3)	*Seq (Seq (Seq*(S1, S'), S"), S3)	AP1.1 AP1.2 AP2.3
AP6.2	Parallelization of services	*Seq (Seq (Seq*(S1, S'), S"), S3)	*Seq (Seq*(S1, *Par* (S', S")), S3)	AP2.1 AP1.5
AP6.3	Inclusive Alternation of services	 *Seq (Seq (Seq*(S1, S'), S"), S3)	*Seq (Seq*(S1, *Alt*(S', S")), S3)	AP2.1 AP1.3
AP6.4	Exclusive Alternation of services	*Seq (Seq (Seq*(S1, S'), S"), S3)	*Seq (Seq*(S1, *Exl*(S', S")), S3)	AP2.1 AP1.4

TABLE VII. DESCRIPTION OF "INTERACTION" ADAPTATION PATTERNS

AP7: "Interaction" Adaptation Patterns			
Pattern Reference	**Pattern Description**	**Scenarios**	**Patterns used**
AP7.1	Add Interaction Point	- Add an external service - Substitute a local service by an external one	AP1.x AP3
AP7.2	Remove Interaction Point	- Remove an external service - Substitute an external service by a local one	AP2.x AP3
AP7.3	Update Interaction Point	- Update input/output data exchanged - Update the interaction mode (synchronous/asynchronous)	AP3

On Table VII, we describe simple scenarios of adapting interaction points. Then, for example, adding an interaction point can be realized by adding an external service (provided by an external partner) or by substituting a local service by an external one in case of a new subcontracting for example. This can be realized using AP1.x patterns (depending on the structure of the process) for adding services or the AP3 pattern for substituting services. The update of interaction point can concern the modification of the data flow or the modification of the interaction mode which can be done by substituting external services containing "receive" and "reply" activities for respectively asynchronous and synchronous interactions.

VII. SOME IMPLEMENTATION DETAILS

A. *Implementation Tools*

We have implemented a framework containing the set of adaptation patterns previously described (and others patterns). For the development of our application, we have considered process models specified with BPEL and interpreted by the WF engine OPEN ESB 2.2, we also used a plug-in SOA Netbeans. We have developed our framework using the Java language and the IDE Netbeans, the application server used is GlassFish server version 2. To implement the adaptation patterns, we have used the API jdom2 that eases the modification on the code BPEL specifying the WF processes

since it is based on the XML language. For example, we simply use the class *Element* implemented in the API jdom to create a new XML tag.

Our framework of adaptation is as modular as possible since we implement a separate class for each adaptation pattern. Then, we create a class for adding a service *after* another service in a sequential branch, another class for adding a service *before* another service in a sequential branch, another class for adding a service in an alternative block, etc. This eases the maintenance of the application and the reuse of existing patterns to implement other ones; for example the operations of substitution, fusion and decomposition are implemented using elementary operations of adding and removing of services (see Tables IV, V and VI).

B. *Illustration of the AP1.2 Pattern*

Fig. 5 bellow shows the interface related to implementation of AP1.1 or AP1.2 patterns (add a service in sequence). The designer of the WF process has to introduce some parameters like the name of the service (for example "NewService") to add, the inputs and outputs, the location (before or after what service). Fig. 6 shows the code java corresponding to the implementation of the pattern AP1.2 "add in sequence after…"

Fig. 5. Interface corresponding to **"Adding a service"**

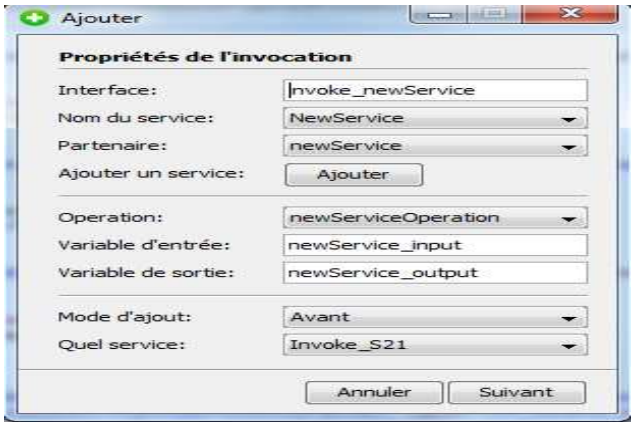


Fig. 6. Part of the java code corresponding to **"Adding a service"**

```
 * and open the template in the editor.
 */
package OperationPack;

import org.jdom2.Element;

/**
 * @author user
 */
public class InsererAvant implements Inserer{
    private void insererAvant(Element eltcourant, Element eltnouveau) {
        if ((eltcourant != null) && (eltnouveau != null)) {
            Element e = (Element) eltcourant.getParent().clone();
            eltnouveau.setNamespace(eltcourant.getNamespace());
            if (e.getName().equals("sequence")) {

eltcourant.getParent().addContent((eltcourant.getParent().indexOf(eltcourant)),
eltnouveau); }
             else {
                Element n = new Element("sequence");
                String s = "";
                s = "Sequence_" + eltcourant.getAttributeValue("name") + "_" +
eltnouveau.getAttributeValue("name");
                n.setAttribute("name", s);
                n.setNamespace(eltcourant.getNamespace());
                n.addContent(eltnouveau);
                n.addContent(eltcourant.clone());
eltcourant.getParent().addContent((eltcourant.getParent().indexOf(eltcourant)), n);
                Supprimer.remove(eltcourant);
            }}
        }
```

Use of the API jdom2

Fig. 7 bellow shows the specification BPEL of the process adapted after adding "NewService" after Service S21.

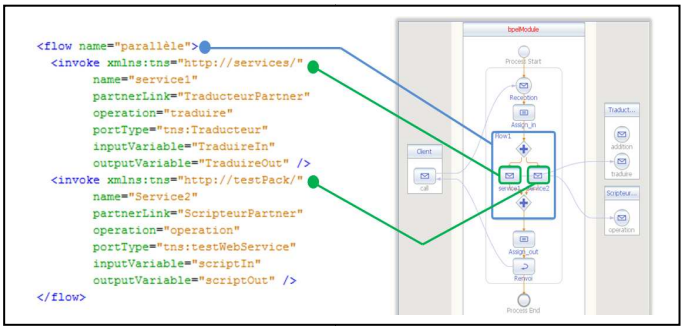
Fig. 7. The specification BPEL adapted after adding "NewService"

```
<variable xmlns:tns="http://WebServices/" name="newService_output" messageType="tns:newServiceOperationResponse" />
riables>
quence>
<receive name="Receive1" createInstance="yes" partnerLink="PartnerLink1" operation="Operation1" xmlns:tns="http://j2ee.netbeans.org/wsdl/Processus/Partner1"
<assign name="Assign1">
    <copy>
        <from variable="Operation1In" part="Input1" />
        <to>$S11_Input.parameters/input$11</to>
    </copy>
</assign>
<invoke name="Invoke11" partnerLink="S11" operation="operationS11" xmlns:tns="http://WebServices/" portType="tns:S11" inputVariable="S11_Input" outputVarial
<assign name="Assign2">
    <copy>
        <from>$S11_Output.parameters/return</from>
        <to>$newService_input.parameters/input</to>
    </copy>
</assign>
<invoke xmlns:tns="http://WebServices/" name="invoke_newService" partnerLink="newService" operation="newServiceOperation" portType="tns:NewService" inputVa
<assign name="Assign_newService" >
    <copy>
        <from>$newService_output.parameters/return</from>
        <to variable="S21_Input" part="Input2"/>
    </copy>
</assign>
<invoke name="Invoke_S21" partnerLink="PartnerLink2" operation="S21" xmlns:ns="http://j2ee.netbeans.org/wsdl/Processus/Partner2" portType="ns:PortType2" in
<assign name="Assign3">
    <copy>
        <from variable="S21_Output" part="Output2" />
        <to>$S13_Input.parameters/input$13</to>
    </copy>
```

C. *Illustration of the AP4.4 Pattern*

Here we illustrate the implementation of the AP4.4 pattern corresponding to a fusion of two parallel services (see Table IV). Fig. 8 shows a schema of a BPEL process containing two services in a parallel block.

Fig. 8. Schema and specification BPEL of a process containing a parallel block



On Fig. 9, we show the interface of fusion and part of the code java corresponding to the implementation of the pattern AP4.4 "Parallel Fusion of services". Fig. 10 shows the schema of the BPEL process after merging services Service1 and Service2 into one service "NewService".

D. *Update Variales and Conditions*

In order to maintain the coherence of the process after adaptation, our application provides an interface allowing the update of the data flow (that means data flow dependencies) in the process. It is to select a service and all input/output variables are displayed to the designer who selects the appropriate input/output variables.

Also, when the adaptation concerns alternative blocks, we have to generate the correct conditions of choice, then our

application provides a simple graphical wizard allowing the generation of simple or composite conditions.

Fig. 9. Interface and part of the java code corresponding the implementation of the AP4.4 pattern

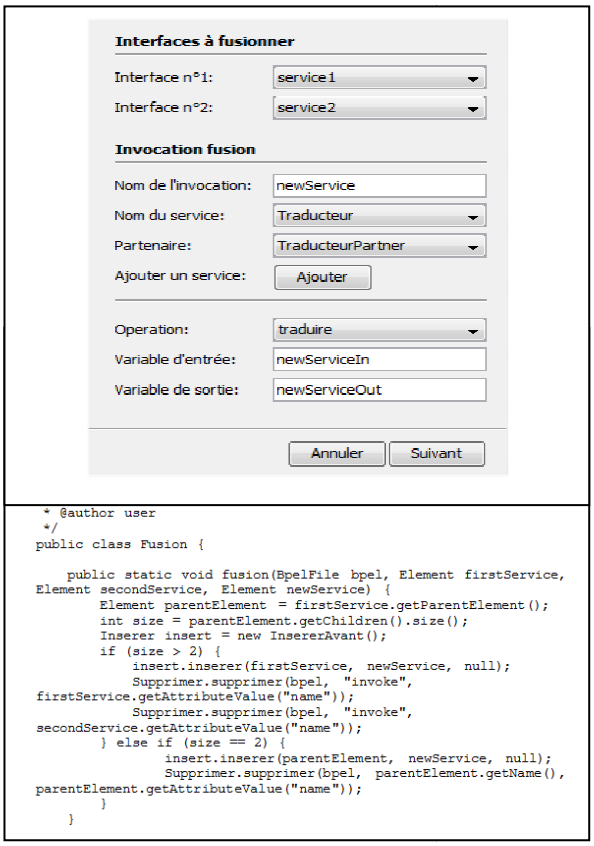
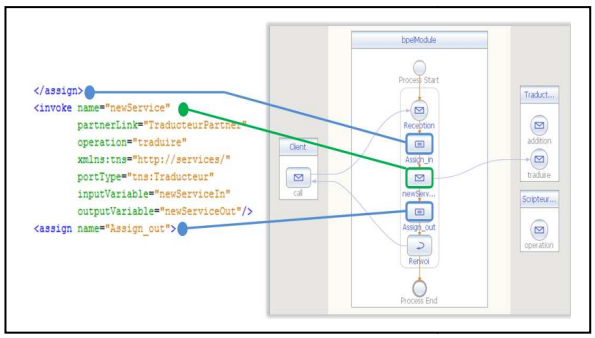


Fig. 10. Schema and specification BPEL of a process after the the fusion of "Service1" and "Service2" into "NewService".



E. *Dynamic Adaptation*

The scenarios described in the previous section are suitable to static adaptation that means the adaptation at build-time that requires the intervention of the designer to set the operation of adaptation.

Let's notice that the adaptation patterns implemented can also be applied at runtime combined with a technique of dynamic adaptation; this requires an automatic setting of the needed parameters (service name/ input/ output/ condition, etc.) to the operations of adaptation. At the current stage of our work, we have only simulated dynamic adaptation by suspending process instances, then applying adaptation and finally resuming the execution of the instances suspended. To perform this simulation, we have used the API BPELMonitor containing the methods that provide the manipulation of the process instances in execution; these methods are:

- *getBPELInstances* : returns the query of all instances in course of the specified process in parameters.
- *suspendInstance* : suspends the instance specified in parameters.
- *terminateAllInstance* : resumes all the instances suspended for a specific process specified in parameters.

To simulate dynamic adaptation, the user should specify the composite application in which the BPEL process is running in order to get the process-id. Then, using the process-id, a specific process instance is stopped to do the required adaptation. After that, the user should deploy the composite application again to take into account the adaptation made. Finally, the suspended instance is resumed in order to observe that the instance runs conformably to the adapted model.

VIII. Conclusion and Future Works

This paper deal with adaptability of IOWF models in the context of structured cooperation. In our research works, we consider process models obeying to generic IOWF-architectures defined in [4] [5]. Our contribution consists in two main issues: first, to make the paper self-containing, we introduced the concept of Service-Based Cooperation Pattern (SBCP) in order obtain IOWF process models flexible enough to ease their adaptation. A SBCP is defined around three main dimensions: the structuring of the IOWF process into services, the control of execution which is expressed through global or local orchestration functions and the structure of interaction with external services provided by other partners involved in the cooperation. The second (which is the main) issue of this paper is to describe our framework of adaptation patterns that can be applied on IOWF models obeying to a given SBCP. We focused on functional, behavioral and interactional aspects of the process. So, we have defined three categories of adaptation patterns: "Service" adaptation patterns, "Control Flow" adaptation patterns and "Interaction" adaptation patterns. We have illustrated some implementations of our patterns on BPEL process models.

The proposed patterns are applied on process models at the design time. As already evoked, we have only simulated the processing of dynamic adaptation (at runtime) and we intend to implement a tool to support it, using the patterns already implemented by setting automatically all parameters needed to apply a specific adaptation according to a specific situation detected by the WF engine (like a failure, an unavailability for a service or a service-interface change according to a given situation). This requires the definition of a dynamic adaptation technique such as a rule-based technique.

Another issue that we are currently developing is the definition and the implementation of a set of evolution patterns. We define operations of evolution (called evolutive adaptation) basis on two perspectives: the expansion of the

IOWF *functionality* and the expansion of the *cooperation*. The expansion of the IOWF functionality is performed using adaptation patterns already described that means "Service adding" patterns and "Service Substituting" patterns; the only difference is that the services newly injected provide additional functionalities. Also, we state that the operations of evolution, particularly the expansion of the cooperation, depend on the IOWF-architecture, we have implemented some evolution patterns suitable to the *chained execution* and the *subcontracting* architectures.

Furthermore, with the proposed approach, we can deal with reusability (well supported by SOA) of IOWF process models which is another aspect of flexibility. Reusability allows the combination of several IOWF obeying to the same or different IOWF-architectures, in order to build more complex business processes based on existing ones. For this issue, we are working to define a set of composition patterns for BPEL process models.

ACKNOWLEDGMENT

We would like to thank our students Djamel-Eddine Khelladi and Younes Asma for their participation in the implementation of the framework.

REFERENCES

[1] W.V.D. Aalst, "Workflow Management: Models, Methods and Systems". The MIT Press. Cambridge, Massachusetts, London, England. 2002.

[2] G. Alonso, F. Casati & H. Kuno, " Web services: concepts, architectures and applications", Heidelberg, Germany, Springer Verlag, 2004.

[3] Mike P. Papazoglou, Willem-Jan van den Heuvel, "Service Oriented Architectures : approaches, technologies and research issues", the VLDB Journal, vol.16, pp 389-415, 2007.

[4] W.V.D. Aalst, "Process oriented architectures for electronic commerce and interogranizational workflow", Journal of Information systems, volume 24 issue 9, 1999.

[5] W.V.D Aalst , "Loosely Coupled Interorganizational Workflows : modeling and analyzing workflows crossing organizational boundaries", Journal of Information and Management Vol37, Pp: 67-75 Issue 2, March 2000.

[6] I. Chebbi, "CoopFlow : an approach for ascendant cooperation of workflows in virtual enterprises" . Phd Thesis, National Institute of Telecom, France, 2007.

[7] P. Grefen, , N. Mehandjiev, G. Kouvas, , G. Weichhart, and R. Eshuis, "Dynamic business network process management in instant virtual enterprises", Computers in Industry, **60**(2), pp. 86–103. 2009

[8] C. Peltz, , "Web Services Orchestration and Choreography", *IEEE Compute*r, Vol. 36, No. 10:46-52, 2003.

[9] T. Amirereza, "Web Service Composition Based Interorganizational Workflows", Sudwestdeutscher Verlag fur Hochschulschriften edition, 2009, ISBN 9783838106700.

[10] D. Jordan, J. Evdemon, "Web services business process execution language V.2.0", *W3C*. 2006.

[11] F. Leymann, , D. Roller, and M.-T. Schmidt, "Web Services and Business Process Management" *IBM Systems, Journa*l, Vol. 41, No. 2, 2002.

[12] S. Gorton, C. Montangero, S. Reiff-Marganiec, L.Semini, "StPowla: SOA, Policies and Workflows", ICSOC 2007 workshops, LNCS 4907, pp. 351-362, 2009.

[13] P. Grefen, K. Aberer, Y. Hoffer, and H. Ludwig "CrossFlow : Cross-organizational workflow management for service outsourcing in dynamic virtual enterprises". *IEEE Data Engineering Bulletin*, 24(1) :52–57, 2001.

[14] N. Mehandjiev, I. Stalker, K. Fessl, and G. Weichhart., "Interoperability contributions of CrossWork". *In invited short paper to Proceedings of INTEROP-ESA'05 Conference*, Geneva, February 2005. Springer-Verlag.

[15] K. Belhajjame, G. Vargas-Solar, and C. Collet, "Pyros - an environment for building and orchestrating open services". In *Proceedings of the 2005 IEEE International Conference on Services Computing, pages 155–164, Washington, DC, USA, 2005*.

[16] F. Casati and M. Shan, "Dynamic and adaptive composition of e-services". *Information Systems*, 26(3):143–163, 2001.

[17] S.W. Sadiq, M.E. Orlowska, "On capturing Exceptions in workflow process models". In proceedings of ER'2001.

[18] J. Meng, , S.Y.W Su, H. Lam, A. Helal, , J. Xian, X. Liu, and S. Yang, "DynaFlow: a dynamic inter-organisational workflow management system", *Int. J. Business Process Integration and Management*, Vol. 1, No. 2, pp.101–115. 2006

[19] Q. He, Y. Yan, H. Jin, "Adaptation of web service composition based on WF patterns" *In proceedings of Service Oriented Computing*, ICSOC, 2008.

[20] M. Döhring, B. Zimmermann, L. Karg, "Flexible Workows at design- and Runtime using BPMN2 Adaptation Patterns". In proceedings of BIS'2011- Springer, 2011.

[21] B. Weber, M. Reichert, S. Rinderle-Ma**,** "Change patterns and change support features- Enhancing flexibility in process-aware information systems". Journal of Data & Knowledge Engineering volume 66, pp 438-466, 2008.

[22] R. Muller, U. Greiner, E. Rahm, " AGENT-WORK: a workflow system supporting rule-based workflow adaptation". In journal of Data and Knowledge Engineering , Data and Knowledge Engineering 51 (2) 223-256, 2004

[23] M. Döhring, B. ZimmermaSnn, E. Godehardt, "Extended workflow flexibility using rule-based adapatation patterns with eventing semantics". In proc. of INFORMATIK'10, pp. 216,226- 2010.

[24] M. Pesic, MH. Schonenberg, N. Sidorova, W. Van der Aalst. "Constraint-based workflow models: Change made easy". In Proceedings of the OTM Conference CoopIS'2007. In vol 4803 of Lecture Notes in Computer Science, pp 77–94.Springer-Verlag, Berlin, 2007.

[25] S. Tragatschnig, U. Zdun, " Runtime Process Adaptation for BPEL Process Execution Engines", 15th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011.

[26] W.V.D. Aalst, W.M.P, ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. DAPD 14(1), pp. 5-51, 2003.

[27] N. Russell, W.V.D Aalst, W.M.P, A.H.M ter Hofstede, "Exception handling patterns in process-aware information systems". In: CAiSE'06 (Luxembourg), pp. 288-302, 2006.

[28] S. Boukhedouma, Z. Alimazighi, M. Oussalah, D. Tamzalit, "SOA based approach for interconnecting workflows according to the subcontracting architecture". In proceedings of MCCSIS- IADIS International Conference, CT'2011. Italy. Pp 3-12. ISBN:978-972-8939-40-3, 2011.

[29] S. Boukhedouma, Z. Alimazighi, M. Oussalah, D. Tamzalit, "Adaptability of service based workflow models : the chained execution architecture". In proceedings of BIS'2012, Lithuania, 21-23 may. W. Abramowicz et al. (Eds.): BIS 2012, LNBIP 117, Springer-Verlag Berlin Heidelberg 2012.

[30] S. Boukhedouma, Z. Alimazighi, M. Oussalah, D. Tamzalit, "Interconnecting workflows using services: an approach for case transfer with centralized control". In proceedings of ICISTM'2012, S. Dua et al. (Eds.): CCIS 285, pp.396–401, Springer-Verlag Berlin Heidelberg, 2012.