

Flexible Loosely Coupled Inter-Organizational Workflows using SOA

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

▶ **To cite this version:**

Saida Boukhedouma, Mourad Chabane Oussalah, Zaia Alimazighi, Dalila Tamzalit. Flexible Loosely Coupled Inter-Organizational Workflows using SOA. 2013 ACS Conference on Computer Systems and Applications (AICCSA'2013), May 2013, Fes, Morocco. pp.1-8. hal-01063849

HAL Id: hal-01063849
https://hal.science/hal-01063849

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.

Flexible Loosely Coupled Inter-Organizational Workflows Using SOA

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*— Service Oriented Architecture (SOA) is a paradigm that provides important properties for the development of business applications like flexibility and loose coupling. In our research work, we focus on the use of SOA to implement specific architectures of inter-organizational workflows (IOWF). The current paper deals with the "Loosely Coupled Workflow" specifying an IOWF-architecture that connects two or more workflows -attached to a set of business partners- communicating in an asynchronous manner according to a public communication protocol conjointly defined by all partners. The first issue of this work is to define a service based cooperation pattern called LC-IOWF pattern suitable to the architecture considered in order to obtain IOWF models flexible enough to ease their adaptation. The proposed LC-IOWF pattern is based on three main dimensions: services, control of execution and interactions. Then, we define three categories of adaptation patterns corresponding to the three dimensions exhibited. Particularly, the third category of these patterns called "Interaction adaptation patterns" concerns adaptations affecting the communication protocol and constitutes a specific type of adaptation compared with other IOWF-architectures. For implementation, we consider IOWF models specified with BPEL.**

Keywords — ***LC-IOWF, Service, Cooperation pattern, Adaptation pattern, Asynchronous communication.***

I. INTRODUCTION

In the business area, the B2B cooperation was initially supported by concepts and tools of *inter-organizational workflow* (IOWF) [1] that implies a set of business partners providing common services to customers. With the emergence of *service oriented architectures* (SOA) [2] and web services standards [3], many research works have been directed towards the combination of workflow and SOA for the development of collaborative business applications.

In our research work, we are interested in structured cooperation supported by the concept of IOWF. In [4], [5], generic architectures of IOWF have been defined: the *capacity sharing*, the *chained execution*, the *subcontracting*, the *case transfer*, the *extended case transfer* and the *loosely coupled WF*. We consider these architectures as basis of our research work 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 support changes [6].

Also, business processes evolve in a dynamic and unstable environment where *flexibility* is an important property that must be satisfied by process models and the systems that implement them. Consequently, we set two objectives of our research works: the first one is to define flexible IOWF models easily adaptable based on existing and fairly common IOWF-architectures and the second one is to provide mechanisms to support changes on the novel models. For that, using a SOA-based approach, we propose *service based cooperation patterns* suitable to the basic architectures defined in [4], [5]. We state that an IOWF process 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 IOWF-architecture [7], [8].

This paper focuses on the *loosely coupled* IOWF-architecture defining a model of cooperation that connects two or more WFs (attached to several partners) interacting together in an asynchronous manner according to a public communication protocol, in order to reach a common business goal.

The first issue of this paper is to define a cooperation pattern based on services called *LC-IOWF pattern;* this last is defined through three main dimensions: services, control and interactions. So, we obtain service-based IOWF models that remain flexible enough to support changes. We define the flexibility of process models according to three aspects: adaptability, evolutivity and reusability. However, at this stage of our work, we focus on the first aspect which is the adaptability of process models, this constitutes the second issue of the paper; we describe the set of adaptation patterns classified in three categories conformably to the three dimensions defining the LC-IOWF pattern. Let's notice that we have implemented a framework of adaptation containing the set of adaptation patterns for IOWF processes specified with BPEL.

In the following, Section II presents some related works and explains the motivation of our work. Section III synthesizes the necessary background to understand the paper. Section IV describes the *cooperation pattern* suitable to the *loosely coupled* architecture and illustrates the concept of *orchestration function.* Section V describes the three categories of adaptation patterns. Section VI gives some implementation

details. Section VII concludes the paper and talks about future works.

II. Related Works and Motivations

The idea of using services to build collaborative business applications is not new. The motivations behind this come from three main points: the relevance of service orientation, the benefits of service orientation for the information system and the benefits of service orientation for the cooperation. For the first point, the concept of service 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.

With the emergence of SOA and web services standards, many research works deal with orchestration and choreography of web services [9], [10], especially based on BPEL4WS.

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 propriety 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 and obeying to the LC-IOWF pattern.

Conceptually, a pattern-based approach allows the enumeration of all recurrent and structurally well defined situations that can occur repeatedly to adapt IOWF processes. From the implementation perspective, the pattern-based approach 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. 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.

B. The Loosely Coupled Architecture

The loosely coupled IOWF (LC-IOWF) is defined by a set of WFs which are distributed among the partner's sites and that interact together using a public protocol based on message exchanges. The communication mechanism used for interaction is *asynchronous.* WF processes operate essentially independently, but have to interact at certain points to exchange data and to ensure the correct execution of the overall business process. The loosely coupled architecture is based on a *process schema partitioning* (disjoint WF fragments are distributed among the partner's sites) and obeys to *decentralized* control of process instances because each partner manages the execution of the WF fragment that he implements and controls the interactions with other WF fragments.

Fig. 1 shows a generic meta-model of LC-IOWF process definition using the UML notation. We can see that a LC-IOWF process model is defined by a set of *WFs* and a *cooperation pattern*. The cooperation pattern links two or more WF through a set of *messages* attached to the *interaction points* in the IOWF. Each WF is attached to a *partner*, manipulates *data* and is submitted to *condition* of invocation. A cooperation pattern is then defined around three dimensions: the *partitioning of the process,* the *control of execution* and the set of messages expressing the *structure of interaction*.

Fig. 1. Meta-model of LC-IOWF Process Definition

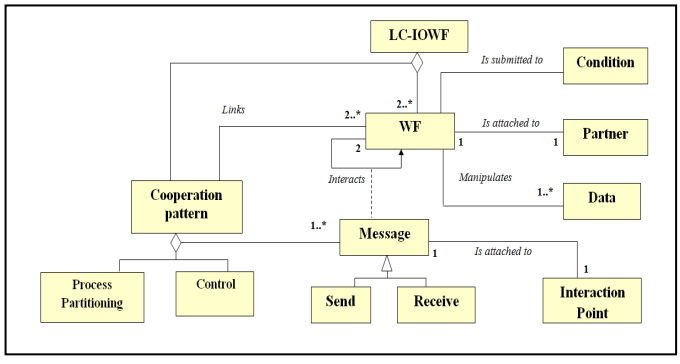
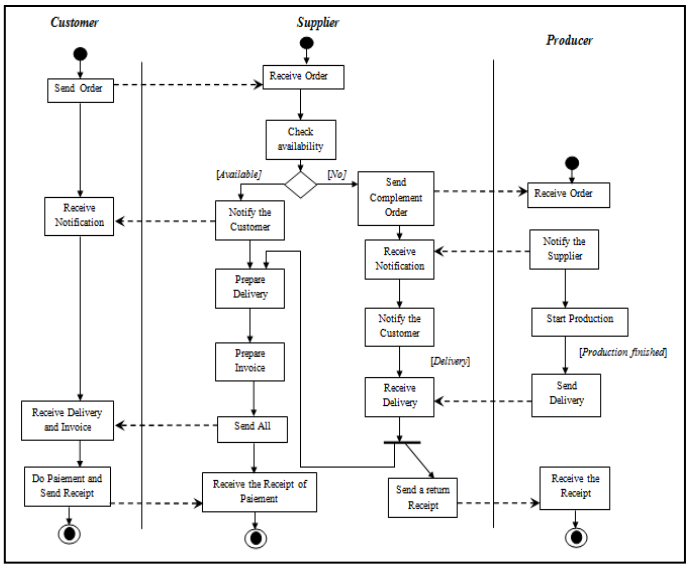


Fig. 2 bellow shows UML activity diagram describing an example of a LC-IOWF process. The process implies three partners: a customer, a supplier and a producer. It consists of managing customer's orders for a given type of products. The customer sends its order to a supplier who checks the availability of products to satisfy the customer's order. If the quantity of products is sufficient, then the customer is notified by a message "Preparing order" else he is notified by a message "Waiting for production" after the supplier has sent the order of production to the producer. When the supplier order is received, the producer starts production and notifies the supplier with a message "Start production". When the production is finished, the producer sends delivery to the supplier who sends them in turn to the customer with a corresponding invoice and sends simultaneously a return receipt of delivery to the producer. The customer does the payment and sends the payment receipt to the supplier. The set of messages exchanged between the three partners are schematized with dotted arrows.

Fig. 2. Example of LC-IOWF process *"Managing Customer's Orders"*



C. Flexibility of IOWF Models

Through the concepts exhibited on the meta-model of Fig. 1, we can see that an IOWF model covers four main axes: *process* (concepts of IOWF, WF, condition and cooperation pattern), *organization* (concept of partner), *data* and *interaction* (concepts of message 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) and in the current paper, we are interested by the first aspect of flexibility which is the *adaptability* of IOWF models. In the next section, we describe the service-based *LC-IOWF pattern* suitable to the loosely coupled architecture in order to obtain IOWF models easily adaptable.

IV. The LC-IOWF Pattern Based on Services

Globally, 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. Then, in order to define a service-based cooperation pattern suitable to a specific IOWF-architecture, the question is to decide which parts of the WF process should be encapsulated within services in order to abstract them and to invoke them from outside. Specifically, *it is to encapsulate a WF process, a sub-process or an activity in a service.*

A. Structuring of the WF Process Into Services

The structuring of the IOWF process into services is done by taking as reference, the *interaction points* in the process model. As shown, on the meta-model of Fig.1, an interaction point is attached to a message and then to an interaction activity (send or receive) in the process. Then, we propose first to isolate the *interaction activities* in the WF process; after that, we structure the WF process of each partner into a set of sub-processes to be encapsulated each of which into a service, by applying the rules set out bellow.

Rule R1: isolate each interaction activity "invoke" or "receive" in the process.

For the cutting of the process into sub-processes, we define the rules R2 and R3.

Rule R2: in a sequential branch (see Fig. 3)

A sub-process in a WF process is delimited: by (i) two interaction activities or (ii) by the start-point of the process and the first interaction activity or (iii) by the last interaction activity and the end-point of the process.

Rule R3: in an alternative (or parallel) bloc (see Fig. 4) Two possibilities are envisaged:

(1) If the bloc doesn't contain any interaction activity, it is considered as a single activity.

(2) If the bloc contains at least one interaction activity:
- Insert fictive interaction points at the OP-Split and the corresponding OP-Join in the process and cut the process at these two points.
- Apply the rule ***R1*** on each edge containing interaction activities.

Rule R4: Encapsulate each sub-process within an internal service.

Fig. 3. Transformation of a sequential process schema

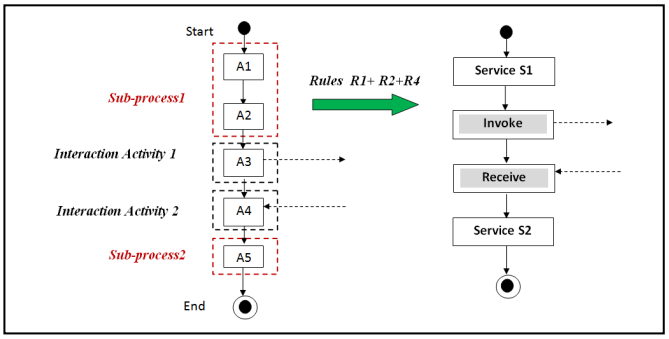
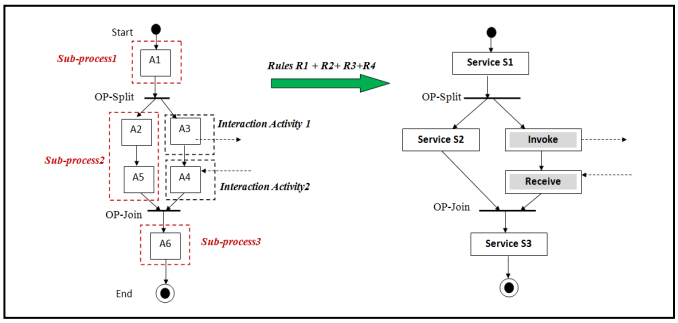
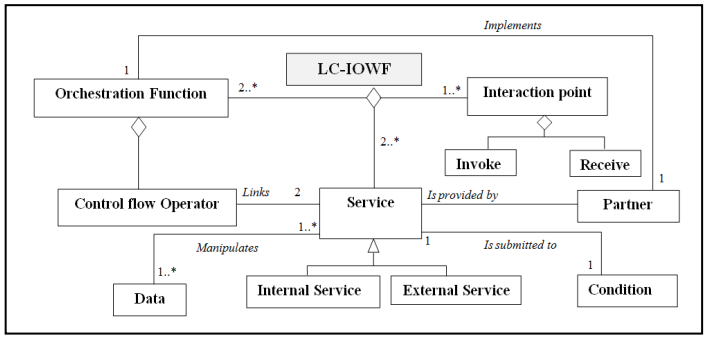


Fig. 4. Transformation of a schema containing parallel or alternative blocs



In addition to the cutting of the WF process into services, we should decide about the appropriate mode of control of execution at runtime and the structure of interaction between services. This leads us to 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 on which is based the LC-IOWF pattern as shown on Fig. 5.

Fig. 5. Meta-model of the LC-IOWF Pattern



Regarding to the first dimension which is the *distribution of services*, we consider that each service encapsulates part 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-architecture. 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* 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 decentralized control, there is a set of local orchestration functions, each of which implemented at one partner site and allows the control of the fragment implemented locally. The concept of orchestration function is defined and illustrated in section B bellow.

The third dimension defines the interactions between services of several partners involved in the IOWF process. This dimension is expressed via interactional points using interactional activities (in BPEL, this is realized by activities *invoke/receive* for asynchronous communication)

B. Orchestration Function and Control Flow

Like shown on the meta-model of Fig. 5, the concept of *orchestration function* describes the control flow between services composing the IOWF using basic control flow operators. In Table I, we introduce these basic operators and we express them using a general notation independently from any language or platform.

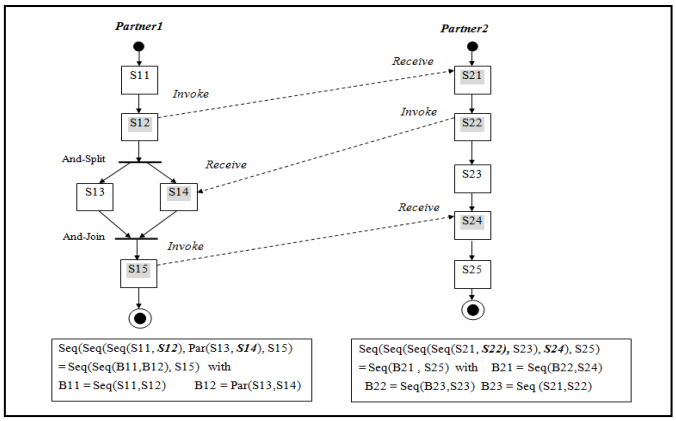
Remark. To describe multi-choice – respectively multi-parallel - (more than two edges), we can decompose on several simple choices – respectively several simple parallel blocs. For example, *Alt* (S1, S2, S3) is expressed as *Alt* (*Alt* (S1, S2), S3) or *Alt* (S1, *Alt* (S2, S3)).

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* (S1,S2)
Alt	*OR-Split*: S1 → S2, S3	*Inclusive choice* between S1 and S2	*Alt* (S1,S2)
Exl	*XOR-Split*: S1 → S2, S3	Describes an *exclusive choice* of S1 and S2	*Exl* (S1,S2)
	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)

Fig. 6 bellow illustrates the concept of orchestration function; we give an example of a process obeying to the LC-IOWF *pattern*. The process schema describes an IOWF implying two partners, partner 1 and partner 2 implementing their WFs as orchestration of services. Partner 1 provides his WF composed by *internal* services *S11and S13 and* interactional activities *S12, S14* and *S15* and partner 2 provides his WF composed by internal services *S23 and S25* and interactional activities *S21, S22* and *S24*.

Fig. 6. Illustration of orchestration functions on a schema obeying to a LC-IOWF pattern



Interaction activities correspond to an "*invoke*" activity from one partner and a "*receive*" activity by the other partner; For example, activity S12 of partner 1 corresponds to an activity *invoke service S21* of partner 2 *that receives* invocation data needed at partner 2 to perform the rest of the process. For more readability and less complexity of the orchestration function, we can structure the process fragments into blocs *Bij* of sequential, parallel or alternative services. In hierarchical manner, a bloc can be expressed using other blocs.

In the next section, we focus on the issue of adaptability of IOWF models. So, we describe a set of adaptation patterns covering the three dimensions on which the LC-IOWF pattern is defined.

V. ADAPTATION PATTERNS

According to the meta-model of Fig. 5, adaptations of an IOWF process model 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 bloc 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). Let's notice that these patterns are applied locally by each partner in order to apply a modification on internal services. The modifications affecting the structure of interaction are more complex and are explained in section C.

- *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 bloc 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.

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 bloc 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.

TABLE II. DESCRIPTION OF "SERVICE ADDING" PATTERNS

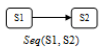
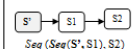
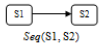
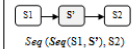
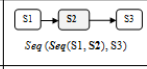
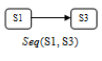
AP1: "Adding Service" Patterns				
Pattern Reference	**Pattern Description**	**Before adaptation**	**After adaptation**	**Patterns used**
AP1.1	Add into *sequence* before a service	 *Seq*(S1, S2)	 *Seq* (*Seq*(**S'**, S1), S2)	**None** (Elementary pattern)
AP1.2	Add into *sequence* after a service	 *Seq*(S1, 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. DESCRIPTION OF "SERVICE REMOVING" PATTERNS

AP2: "Service Removing" Patterns				
Pattern Reference	**Pattern Description**	**Before adaptation**	**After adaptation**	**Patterns used**
AP2.1	Remove from *sequence*			**None** (Elementary pattern)
AP2.2	Remove from one edge with several services of *inclusive choice*			AP2.1
AP2.3	Remove from one edge with single service of *inclusive choice*			AP2.1
AP2.4	Add on one edge of *parallel execution*			AP2.1

For the removing patterns, we can see 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 PA1.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.

- *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 bloc, they become easier to remove and to replace, because the bloc (*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 bloc. 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 blocs.

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 bloc of two services that can be sequential, parallel or alternative bloc. 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 bloc* 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 IV. DESCRIPTION OF FUSION PATTERNS

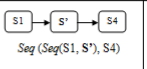
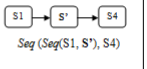
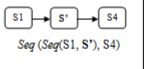
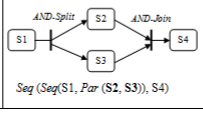
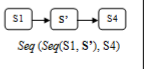
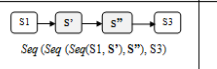
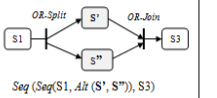
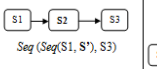
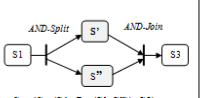
AP4: Fusion Patterns				
Pattern Reference	**Pattern Description**	**Before adaptation**	**After adaptation**	**Patterns used**
AP4.1	Fusion of *sequence*			AP1.1 AP1.2 AP2.1
AP4.2	Fusion of *inclusive choice*			AP1.1 AP1.2 AP2.3
AP4.3	Fusion of *exclusive choice*			AP1.1 AP1.2 AP2.6
AP4.4	Fusion of *parallel execution*			AP1.1 AP1.2 AP2.5

TABLE V. DESCRIPTION OF DECOMPOSITION PATTERNS

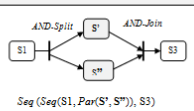
AP5: Decomposition Patterns				
Pattern Reference	**Pattern Description**	**Before adaptation**	**After adaptation**	**Patterns used**
AP5.1	Decomposition into *sequence*			AP1.1 AP1.2 AP2.1
AP5.2	Decomposition into *inclusive choice*			AP2.1 AP1.3
AP5.3	Decomposition into *exclusive choice*			AP2.1 AP1.4
AP5.4	Decomposition into *parallel execution*			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.

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

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 structure of interaction. Specifically, this is done by adding, removing or updating *interactional points* (see table VII).

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 "invoke" activity in the process requester - Add a corresponding "receive" activity in the process requested. - Re-structure the process using rules R1, R2, R3 and R4	AP1.x AP3 AP4.x AP5.x
AP7.2	Remove Interaction Point	- Remove an "invoke" activity from the process requester - Remove the corresponding "receive" activity from the process requested. - If the interaction is two-way Remove an "invoke/request" activities corresponding to the response of the request removed. - Re-structure the process using rules R1, R2, R3 and R4	AP2.x AP3 AP4.x AP5.x
AP7.3	Update Interaction Point	- Update input/output data exchanged - Update the interaction mode (one- way/ two-way)	AP7.1 AP7.2

On Table VII, we describe generic scenarios of adapting interaction points. Then, for example, adding an interaction point can be realized by adding an "invoke" activity at the process requester and a "receive" activity at the process requested. If the interaction is two-way (asynchronous request/response), this should be followed by adding of "invoke/receive" activities for response in the reverse direction. After adding the necessary interaction activities, the process should be re-structured according to the rules R1, R2, R3, R4 specified in section IV.A. The pattern AP7.1 is implemented using AP1.x, AP3, AP4.x and/or AP5.x depending on the structure of the process. For removing an interaction point, it is to remove an "invoke" activity from the process requester and a corresponding "receive" activity from the process requested; then if there is a two-way interaction, we should remove a corresponding "invoke/receive" activities in the reverse direction. As for the AP7.1 pattern, the structure of the process should be updated. The update of interaction point can concern the modification of the data flow exchanged or the modification of the interaction mode one-way/two-way; then the AP7.3 pattern uses AP7.1 or AP7.2.

VI. SOME IMPLEMENTATION DETAILS

We have implemented a framework containing a 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 of 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 bloc, etc. This eases the reuse of existing classes 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).

For each operation of adaptation, a wizard interface is provided allowing the setting of all parameters necessary to perform the adaptation.

Also, in order to maintain the coherence of the process after adaptation, our application provides an interface allowing the update of the data flow 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.

Furthermore, when the adaptation concerns alternative blocs, we have to generate the correct conditions, then our application provides a simple graphical wizard allowing the generation of simple or composite conditions.

After each operation of adaptation, we run the process in order to check that the adaptation has been successfully done.

Regarding to the LC-IOWF pattern, we have implemented an IOWF-process using the BPEL designer (manually) by specifying two BPEL-Processes and message exchanges between them, we used correlation sets in order to maintain the coherence of the communication protocol. We are

currently working on the implementation of the LC-IOWF pattern (a priori in a semi-automatic way) and we are still working on the implementation of "Interaction" adaptation patterns (AP7.x) which remain more complex from the other categories of patterns because they necessitate the update of the correlation sets.

VII. Conclusion and Future Works

This paper deal with adaptability of IOWF models obeying to the *loosely coupled* architecture defined in [4], [5]. Our contribution consists in two main issues; first, we defined a cooperation pattern called *LC-IOWF pattern* based on *services* in order to deal with flexible IOWF models thanks to SOA advantages. In order to maintain a *decentralized* control, each partner implements his orchestration function and the communication protocol is implemented through *interactional* activities "invoke" and "receive". For the second issue, we state the main adaptation patterns classified in three main categories basis on three dimensions: services, control and interaction. Specifically, the "interaction" adaptation patterns concern the update of the communication protocol and requires more processing in order to keep the communication protocol coherent and consistent.

Currently, we are working on the implementation of the LC-IOWF pattern (a priori in a semi-automatic way) and we complete the implementation of the "interaction" adaptation patterns. As future works, we intend to define and implement some operations of evolution (called evolutive adaptation) that we distinguish from other adaptations basis on two perspectives the expansion of the global *functionality* of the IOWF process and the expansion of the *cooperation*. Furthermore, with the proposed approach, we can deal with reusability (well supported by SOA) of IOWF process models which is another aspect of flexibility allowing 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.

References

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

[2] 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.

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

[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] 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.

[8] 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.

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

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

[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.