

Service Based Cooperation Patterns to Support Flexible Inter-Organizational Workflows

Saida Boukhedouma

University of Science and Technologies Houari Boumediene, Algiers, Algeria
E-mail: sboukhedouma@usthb.dz

Mourad Oussalah

University of Nantes, France
E-mail: mourad.oussalah@univ-nantes.fr

Zaia Alimazighi

University of Science and Technologies Houari Boumediene, Algiers, Algeria
E-mail: zalimazighi@usthb.dz

Dalila Tamzalit

University of Nantes, France
E-mail: dalila.tamzalit@univ-nantes.fr

Abstract— Service Oriented Architecture (SOA) is a paradigm that provides important advantages like interoperability, reusability and flexibility, particularly beneficial for B2B applications. In the current paper, we consider specific architectures of inter-organizational workflows (IOWF) fairly widespread in the B2B area and implementing different cooperation schemas. Our aim is to propose new generic IOWF-architectures by using the SOA paradigm in order to obtain IOWF models flexible enough to ease their adaptation, evolution and reuse. For that, we introduce the concept of Service-Based Cooperation Pattern (SBCP) that supports the definition of IOWF models based on services. A SBCP is defined by three main dimensions: the distribution of services, the control of execution and the structure of interaction between services. Also, we define a concept of composite cooperation pattern based on the combination of elementary patterns. We illustrate our approach by a general description of our cooperation framework called “S-IOWF” that supports the implementation of IOWF models obeying to the described SBCP. Three main points characterize our approach: (i) the use of a pattern-based approach; (ii) the definition of composite patterns by reusing elementary ones and (iii) the support of several cooperation schemas with different types of control.

Index Terms — IOWF, SOA, Service Based Cooperation Pattern (SBCP), Flexibility, Composite Pattern

I. Introduction

Since the year 2000, many works deal with the combination of business oriented technologies such as workflow [1] and web services [2] supported by SOA [3], to build collaborative and distributed business applications which are suitable for *ad-hoc* cooperation [4] or *structured* cooperation [5][6]. 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. Ad-hoc cooperation is appropriate for occasional and non durable B2B relationships. However, in many situations, business partners need to agree together in order to build structured and durable cooperation to reach a common business goal according to a “winner-winner” policy. In structured cooperation, the steps of the business process and interactions in the system are well defined resulting in an IOWF model clearly defined and followed by all process instances.

In our research work, we are interested in structured cooperation supported by the concept of inter-organizational workflow (IOWF). In [7], [8], generic architectures of IOWF have been defined to support this kind of 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 express different cooperation schemas. However in their initial form, these architectures were subject to criticisms because of their rigidity and the difficulty to adapt to changes [9].

Furthermore, because the environment of businesses is naturally dynamic and unstable, business processes are continually or occasionally subject to changes. Then, the final objective of our research is to deal with flexibility of IOWF models by providing mechanisms that support their adaptation, evolution and reuse. However, before we get to deal with flexibility, we define new IOWF-architectures that support process models flexible enough in order to ease their adaptation, evolution and reuse. So, the current paper focuses on the description of these new IOWF-architectures using the SOA paradigm.

The use of SOA approach for WF interconnection is not new and is motivated by the fact that services are loosely coupled components, easily invoked, business oriented and platform independent and SOA paradigm supports integration, reuse and composition of services. Then, our contribution in this paper is to define and to implement *Service-Based Cooperation Patterns* (SBCP) corresponding to the basic architectures defined in [7] [8]. We state that the basic 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, respecting the constraints of each IOWF-architecture.

Three main points characterize our contribution: (i) by considering several IOWF-architectures, we ensure that we cover a wide range of existing business processes (ii) By using a pattern-based approach, we ease the maintainability and the extensibility of the cooperation framework and (iii) by reusing existing IOWF models, we can build more complex ones obeying to *composite* cooperation patterns.

The rest of the paper is structured as follows: Section 2 presents some related works and explains the motivations of our research. Section 3 synthesizes the necessary background to understand the paper. Section 4 lays the basis of our approach for WF interconnection using services; here, we introduce the concept of SBCP. Section 5 describes the set of SBCP proposed. Section 6 gives some implementation details of our cooperation framework. Section 7 talks about generalized and composite cooperation patterns. Section 8 provides a comparison of some WF cooperation approaches proposed in the literature. Finally, Section 9 concludes the paper and talks about other works.

II. Related Works and Motivations

With the emergence of SOA and web services standards, many research works deal with orchestration and choreography of web services [10], [11], especially based on BPEL4WS [12]. Other research works such as [13], [14] show the interest of combining BPM, WF and SOA for reusing services to build 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. In *structured* cooperation, we can cite some approaches like CoopFlow [9], CrossFlow [15], CrossWork [16], Pyros [17], e-Flow [18] and DISCOBOLE [19]. A comparison of approaches is provided in Section 8 of this paper.

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. More recently, a certain number of approaches for flexible WF cooperation have been proposed [20], [21], [22]. In [20], the author describes a methodological framework for service-based dynamic cooperation using aspect-programming and context adaptation. The author of [21] describes a framework for dynamic composition of services with asynchronous communication and mechanisms of adaptation for service-based business processes. The author of [22] uses web services and model driven engineering for the construction of extensible business oriented applications.

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 [23], [24], [25]. (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 [26], [27], [28], rule-based adaptation patterns [29], [30] and constraint-based modeling [31] to support flexibility of process models. For example, in [28], the authors identify the most important process change patterns and change features for PAIS (process aware information systems). In [32], 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 context. In the WF area, this concept has been usually used for business process modeling [33], business process improvement or changes [28], [32] or exception handling [34]. More recently, the concept of pattern is used in model transformation; for example in [35], the author proposes transformation patterns to move from choreographies to orchestration of services. Also, workflow patterns are used for verification of service composition like in [36], [37].

This paper deals with WF cooperation and uses a pattern-based approach to define generic IOWF-architectures using the SOA paradigm, by introducing the concept of Service-Based Cooperation Pattern

(SBCP). The idea of using services to build collaborative business applications is not new; the motivations behind this come from three main points: the first point is the relevance of service orientation for the information system since the concept of service (mainly web services) provides credible answers to constraints and problems such as the lack of flexibility and the reluctance to openness. The second point is the benefits of service orientation for the information system because a 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. The third point is the benefits of service orientation for cooperation that is realized by service composition; then businesses provide their services with a certain degree of abstraction allowing them the preservation of *autonomy* and *confidentiality* which are, in addition to *flexibility*, important properties to be satisfied in WF cooperation.

Regarding the choice of the basic IOWF-architectures, we have considered those proposed in [7][8] because they define different cooperation schemas with different types of execution control and then cover a wide range of existing business processes. Consequently, our approach of WF cooperation (and adaptation) can be applied to a large number of existing IOWF processes.

Also, for conceptual aspects of our solution, we adopt a pattern-based approach to define the different schemas of WF cooperation allowing the enumeration of structurally well defined process schemas for WF interconnection. From the implementation perspective, the pattern-based approach allows modular and reusable implementation of the proposed patterns to build more complex ones called *composite* cooperation patterns.

III. Basic Definitions and Concepts

In this section, we introduce the necessary definitions and concepts to ease the understanding of the paper.

3.1 IOWF Definition and Architectures

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 [38].

In [7][8], generic architectures of IOWF have been defined in order to support structured cooperation which must obey, depending on the partners needs, 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” characterized by two main dimensions: the *partitioning of the process* and the *control of execution*.

Regarding 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, in a disjoint manner, among the partner’s sites.

Since IOWF are distributed systems, the control of instance execution can be *centralized*, *decentralized*, *hierarchized* or *mixed*. 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 like in 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 “Chained execution”, “Loosely coupled” and “(extended) Case transfer” architectures. The control can be a *mixture* of centralized and decentralized ones if each system manages the part of WF implemented locally but the management of interactions is delegated to one system; this can be applied to “(extended) Case transfer”. 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, like in the “Subcontracting”. More details of these architectures are given in Section 5 of the paper.

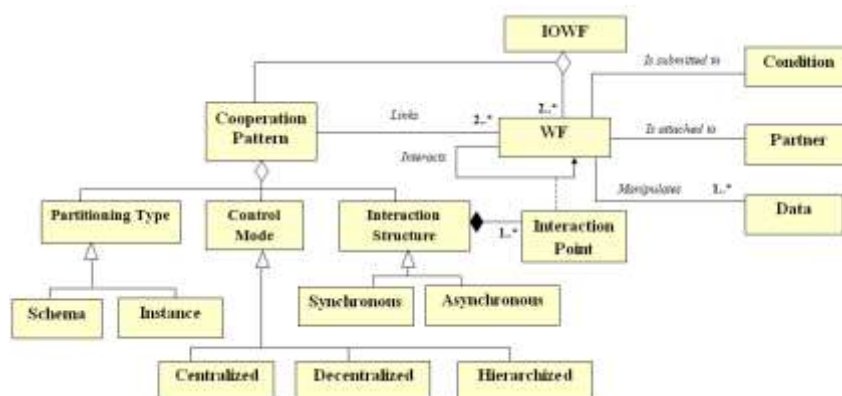


Fig. 1: Meta-model of IOWF process definition

3.2 IOWF Meta-Model

Fig. 1 below shows a meta-model that exhibits the main concepts 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 a *condition* of invocation. A given cooperation pattern is attached to a specific IOWF-architecture; 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 dimension is defined by a set of interaction points between WF fragments and is as important as the two first ones because the structure of interaction differs from a given architecture to another, so we consider it as a third characteristic of an IOWF-architecture that should be taken into account. 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.

3.3 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, 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.

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.

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

In the following section, we explain the basis of our approach mainly the generic schemas of structuring a WF process into services and the concept of SBCP.

IV. Basis of Our Approach

The main idea of our approach is to encapsulate each WF fragment into a single service or a set of services while preserving the interaction points in the basic IOWF-architecture so as interactions between WF fragments turn into invocations of services. The main question is: how to structure an IOWF process into services?

4.1 Structuring of an IOWF into Services

In order to structure an IOWF schema into services, we consider *interaction points* between the workflows involved in cooperation as *markers* allowing the cutting of a process schema into sub-processes to be encapsulated into services.

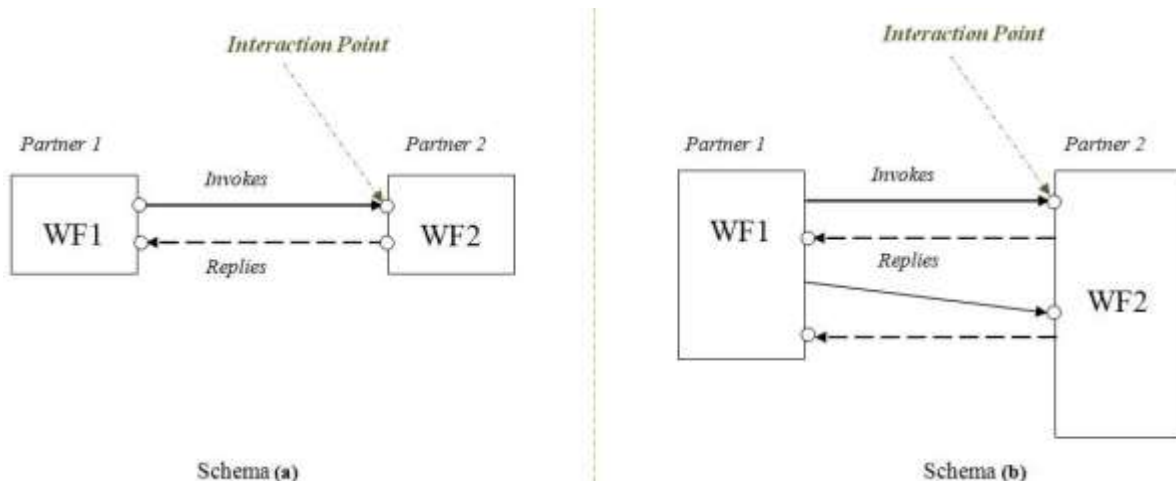


Fig. 2: Interaction Schemas of IOWF

According to the interaction points: we can envisage two configurations: (i) the interaction points frame the whole WF invoked; (ii) the interaction points are located at several points of the WF invoked. Fig. 2 shows two generic schemas of interaction in IOWF implying two partners, partner 1 and partner 2 which implement WF1 and WF2, respectively. In the schema (a) on the left, the interaction points frame entirely WF2; this corresponds to the “Chained execution” and the “Subcontracting”. In the schema (b) the interaction

points frame partially WF2; this is suitable for “Capacity sharing”, “(extended) Case transfer” and “Loosely coupled” architectures. The dashed arrows indicate an optional reply. Depending on the type of IOWF-architecture, the question is to decide which parts of the WF process should be encapsulated within services in order to invoke them from outside. Specifically, it is to encapsulate a WF process or a subprocess into a service.

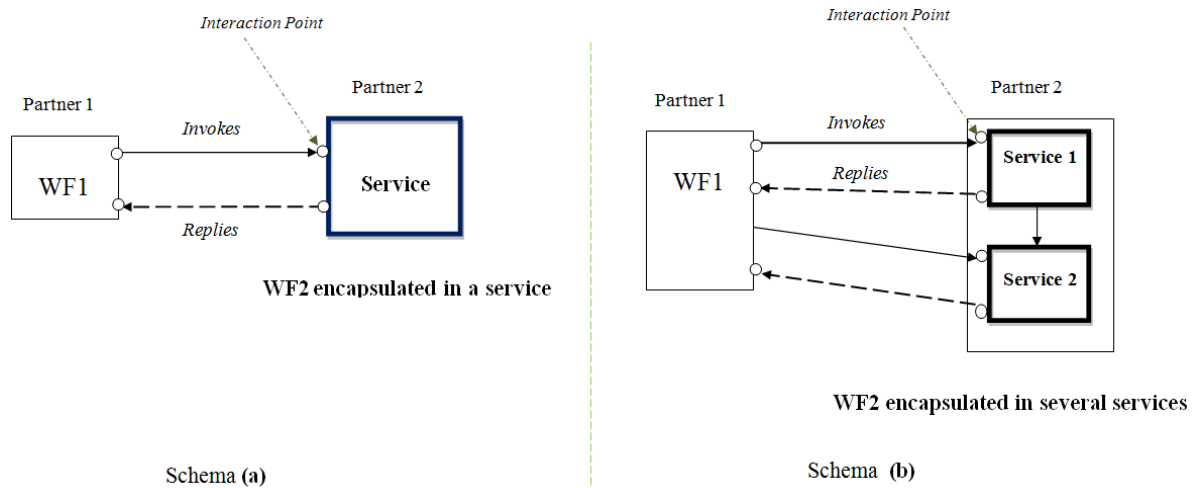


Fig. 3: Generic schemas of encapsulation into services

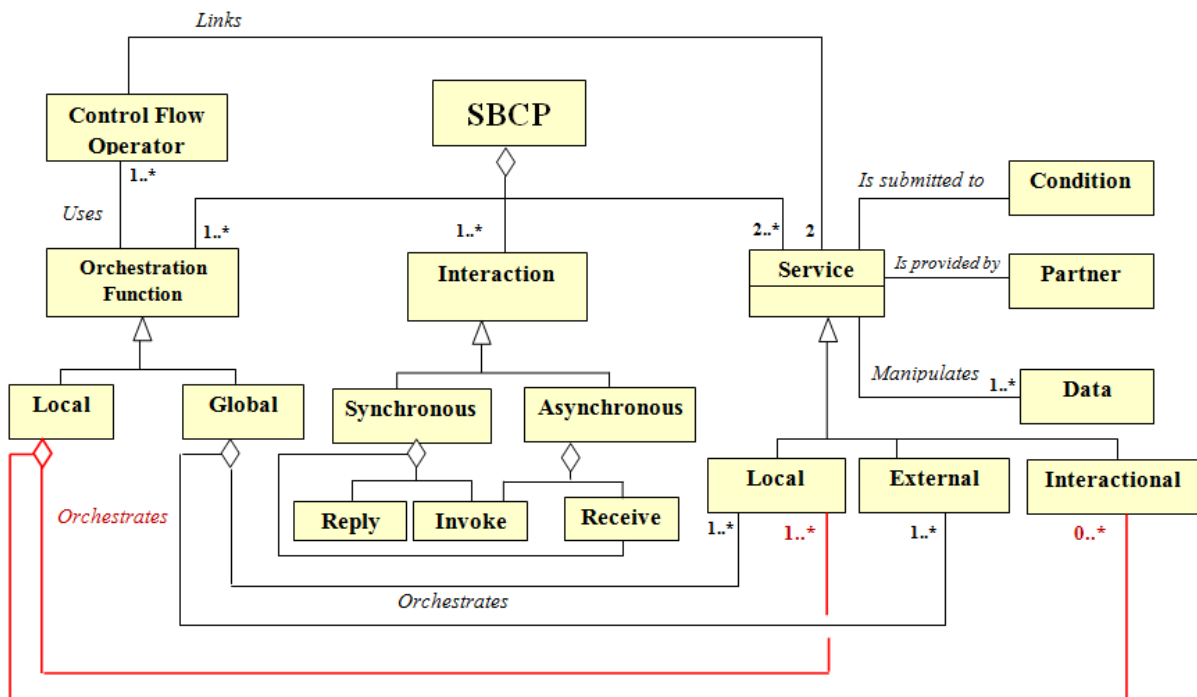


Fig. 4: Meta-model of a SBCP Definition

Starting with the generic schemas of Fig. 2, the parts of WF that should be encapsulated in services are those that require external invocation as schematized in Fig. 3. The schema (a) shows the transformation of the schema (a) of Fig. 2, where the invoked WF (WF2) is

entirely encapsulated into a single service. The schema (b) corresponds to the transformation of the schema (b) of Fig. 2 where WF2 is invoked at various interaction points and therefore requires its cutting into several services. Let’s notice that on Fig. 3, services are not

necessarily atomic; each service can be composed by several services but seems to be atomic from outside. Furthermore, depending on the IOWF-architecture, the operations of invocation are interpreted differently. Indeed, for “capacity sharing”, it is to invoke services from a global process, for a “chained execution”, invocation consists to forward the instance partially performed by a partner to another one in order to complete its execution; for a “subcontracting”, the invocation consists to delegate part (one activity or more) of a principal WF to a secondary WF. For a “(extended) case transfer”, the cooperation is to transfer process instances from one partner’ site to another to complete their execution and for a “loosely coupled WF”, the cooperation consists of asynchronous data exchanges.

4.2 Service Based Cooperation Pattern (SBCP)

In our approach, we define a new concept called SBCP based on SOA where we replace the concept of WF by the concept of *service*. A SBCP allows the characterization of a specific IOWF-architecture using SOA. Then, our approach for WF interconnection focuses on three main questions: (i) How to *structure* the WF process into services? (ii) How to *control* the execution of instances? (iii) 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 (see Fig. 4). Here, we define a SBCP in a generic manner for all IOWF-architectures; in Section 5, we exhibit the specificities of each cooperation pattern.

Regarding 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 site that provides it. This dimension corresponds to the dimension *Process partitioning* defined for the initial IOWF-architectures. From the perspective of a given partner, a service can be implemented locally or provided by an external partner; it can be an interactional service if it ensures interaction among services of different partners.

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 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. By contrast, in case of decentralized control, there is a set of local orchestration functions implemented at the partner’s sites in order to control the execution of the fragments implemented locally. 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 WFs implied in the cooperation.

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/receive/reply for synchronous communication).

4.3 Orchestration Function and Control Flow

Like shown on the meta-model of Fig. 4, the concept of *orchestration function* describes the control flow between services composing the IOWF using basic control flow operators. On Fig. 5, we introduce these basic operators and we express them using a general notation independently from any language or platform.

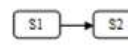
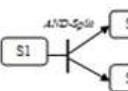

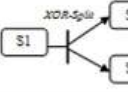
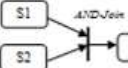
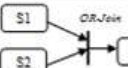
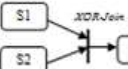
Operator	Schema	Description	Orchestration function
Seq		Sequential execution S1 followed by S2	Seq (S1,S2)
Par		Simultaneous execution of S1 and S2	Par (S2,S3)
Alt		Inclusive choice between S1 and S2	Alt (S2,S3)
Exl		Describes an exclusive choice of S1 and S2	Exl (S2,S3)
		Synchronous merge of S1 and S2 after parallelism Expressed using Par and Seq operators	Seq (Par (S1,S2), S3)
		Simple merge of S1 and S2 after inclusive choice Expressed using Alt and Seq operators	Seq (Alt (S1,S2), S3)
		Simple merge of S1 and S2 after exclusive choice Expressed using Exl and Seq operators	Seq (Exl (S1,S2), S3)

Fig. 5: Basic Control Flow Operators

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)).

Because of specific constraints of each IOWF-architecture considered, we define for each one a corresponding SBCP by refining the generic meta-model of Fig.4 in order to consider specific characteristics, according to the three dimensions identified.

V. The Proposed Cooperation Patterns

In this section, we specify the six SBCP that we propose to meet the basic IOWF-architectures considered. For each SBCP, we give some descriptive details, a generic schema, a meta-model and a set of specification rules.

5.1 The "Capacity Sharing" Pattern - SBCP1

SBCP1 meets the "Capacity sharing" architecture where the partners share the execution of a global WF model. This pattern is implemented as a set of services orchestrated using a global orchestration function implemented at one location inducing a centralized control of execution.

The *orchestrator* of services plays the role of the central WFMS (see Fig. 6); it decides the order of invocation of services. Each partner is responsible of performing the set of services attached to him. SBCP1 is described through the meta-model of Fig. 6. The specification rules set in the description (at the bottom of Fig. 6) express the set of actions to perform in order to obtain an IOWF obeying to SBCP1. An example of an orchestration function for this pattern can be $Seq(Seq(Seq(S1, S2), Par(S3, S4), S5))$ that is interpreted as the invocation of service S1, followed by S2, followed by simultaneous invocations of S3 and S4 and finally synchronized to invoke S5. The interaction pattern for SBCP1 obeys to a synchronous mode between the orchestrator and the set of services provided. In BPEL, the synchronous interaction pattern is realized using an *invoke* activity from the BPEL process and a *receive* activity from the service to accept the input data of the request and a *reply* activity from the service in order to return results and to enable the progress of the client process.

5.2 The "Chained Execution" Pattern - SBCP2

In the "Chained execution" architecture, each partner implements its own WF process. Workflows implied in cooperation are executed in *sequence*. The results of execution of WFi are input data of $WFi+1$. To obtain SBCP2 suitable to the "Chained execution" architecture, we propose to *entirely* encapsulate the WF of each partner within a service that means service Si encapsulates WFi provided by partner i . Process instances are executed according to the *sequence* of services implemented (see Fig. 7). Thus, the first service ($S1$) in the sequence is triggered by an external event (the occurrence of a new instance); for the other services, each of which is triggered by the service that precedes it in the sequence. In a general way, a service $Si+1$ is invoked by service Si that precedes it once Si terminates its execution. We can say that this architecture is implemented as *choreography* of services with *decentralized control*. Also, a reply to the service invoker (for notification) can be facultative.

SBCP2 pattern is described through the meta-model shown on Fig. 7.

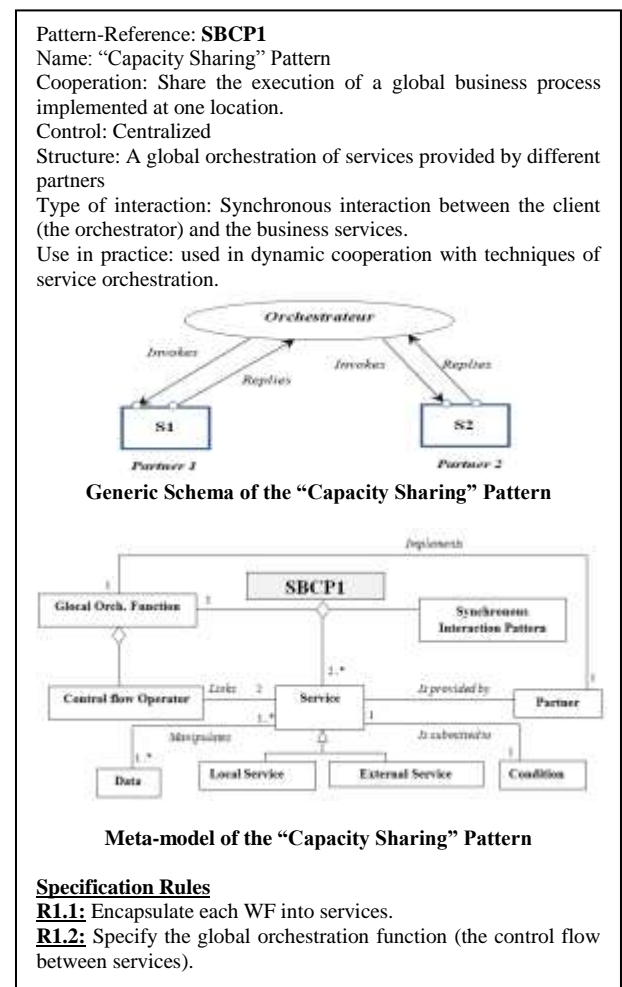


Fig. 6: Description of the "Capacity Sharing" Pattern - SBCP1

At internal level, services Si can be implemented as composite services since they respectively encapsulate the WF of each partner; it means that each internal activity of WFi is implemented as a *local service* Sij . Then, we propose to implement a local orchestration function at each partner where maintaining a *decentralized control* of execution in the IOWF. The local orchestrator of partner i receives *input data* from another orchestrator, *invokes its local service* (Si) with this input data and then invokes service $Si+1$ of the next partner by sending *results* (output) of its local service; this scenario is implemented at each partner implied in the IOWF. For this architecture, the interaction between services obeys to a "one-way" interaction pattern (considered as an asynchronous interaction in a single direction) if no reply is necessary or a "synchronous" interaction pattern if we consider a reply for notification. In a one-a-way interaction, the client sends a message to the service and does not wait for a response. In BPEL, this interaction pattern is implemented using an *invoke* activity from the client (WFi) and a *receive* activity at the service ($WFi+1$) that becomes in turn a client when

it invokes the next service (WFi+2). Fig. 8 illustrates the concept of orchestration function for an IOWF model obeying to SBPC2.

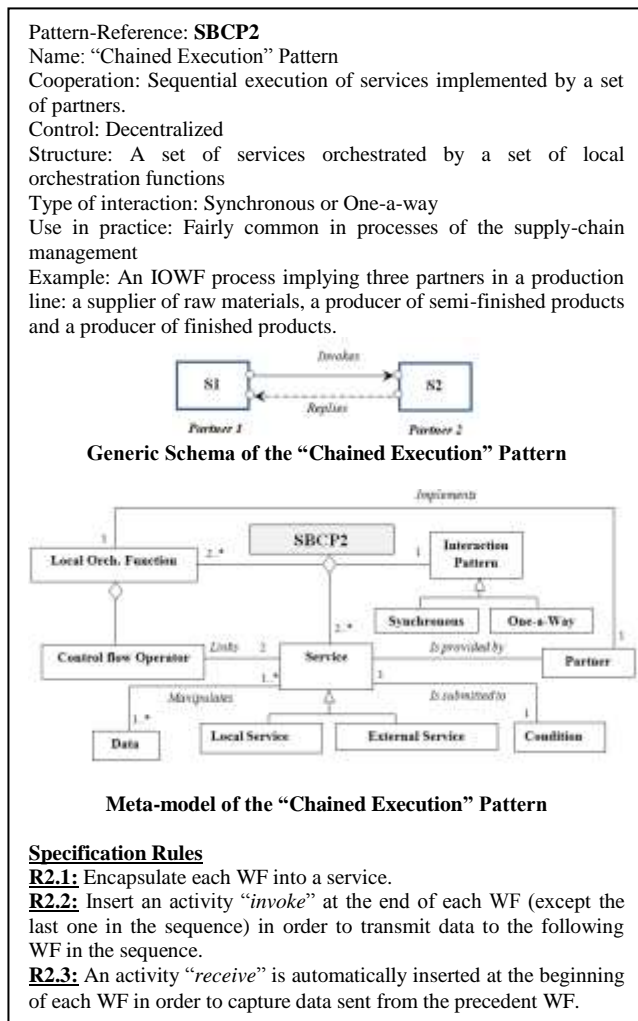


Fig. 7: Description of the "Chained Execution" Pattern – SBPC2

The process schema implies 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* and

partner 2 provides his WF composed by *internal* services *S21* and *S22*. For more readability and less complexity of the orchestration function, we can structure the WF fragments into blocks *Bij* of sequential, parallel or alternative services. In a hierarchical way, a block is expressed using other blocks. *Sout1* corresponds to an activity "invoke" of external service *S2* and *Sin2* corresponds to an activity "receive".

5.3 The "Subcontracting" Pattern – SBPC3

In the "Subcontracting" architecture, there is *one main workflow* attached to the main partner which subcontracts some activities not implemented locally to *one or more secondary workflows* implemented by other partners involved in the cooperation.

In order to obtain an IOWF obeying to SBPC3, we propose to entirely encapsulate *each secondary WF* involved in cooperation within a service. On Fig. 9 for example, partner 1 hosts the main WF and partner 2 provides his secondary WF as a global *service S2* which can be composite but from the perspective of the main partner, it is abstracted to a single entity; thus, Partner 1 invokes the service of partner 2 for subcontracting. To obtain an IOWF entirely based on services, the whole WF can be implemented as an *orchestration of local services* encapsulating activities of the main WF and *external services* provided by secondary partners. In the subcontracting architecture, the interaction between services is synchronous and the control of execution is *hierarchized* because the main WF manages the control of the whole process and controls invocation of external services. SBPC3 is described by the meta-model of Fig. 9.

To illustrate the concept of global orchestration function for SBPC3, we give a simple example of IOWF obeying to the "Subcontracting" pattern (see Fig. 10). The process schema describes an IOWF implying two partners, partner 1 and partner 2. Partner1 provides the main WF composed by *internal* services *S11*, *S12*, *S13*, *S14* and an invocation of *S2* which is the *external* service provided by partner 2.

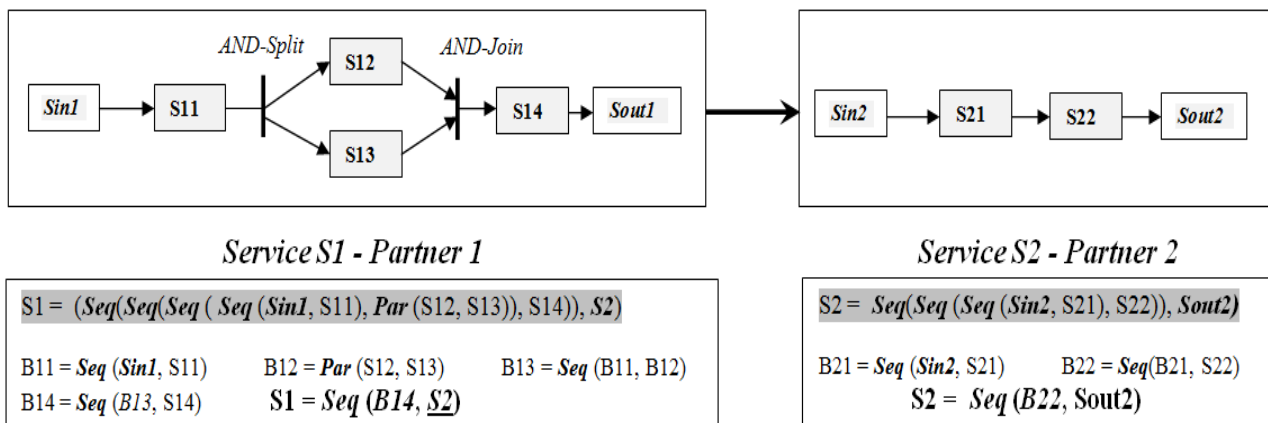


Fig. 8: Illustration of orchestration functions in SBPC2

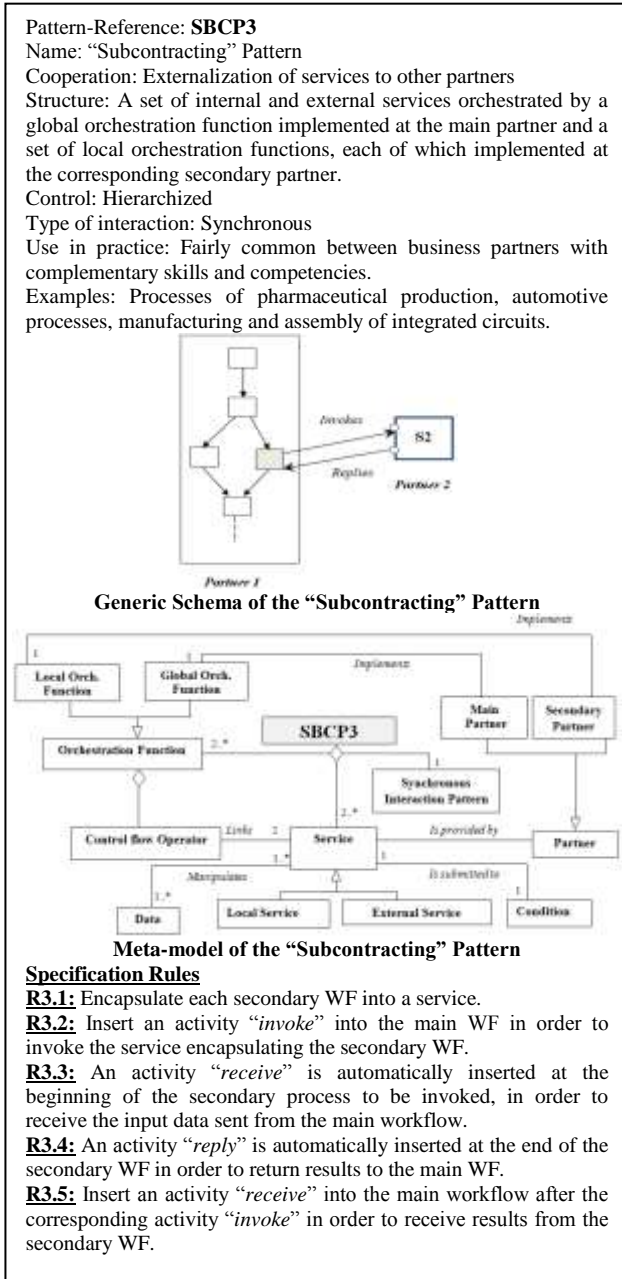


Fig. 9: Description of the "Subcontracting" pattern – SBCP3

5.4 The "(Extended) Case Transfer" Pattern - SBCP4 (SBCP5)

The "Case transfer" (respectively, the "Extended case transfer") architecture defines a form of cooperation fairly widespread in B2B, especially between partners engaged in the same profession and aiming to satisfy promptly many potential customers. In the "Case transfer" architecture, business partners share the same WF model implemented at each partner and hosted by a local WFMS. Their cooperation consists of transferring process instances (cases) from one location (partner) to another in order to achieve their execution. For example, one can envisage an IOWF involving a set of partners in a process of production; a customer's order may arrive at partner *x* but it is not completely performed by the

WF of this partner; the order may be transferred to other partners involved in the IOWF process. The transfer can occur for example, for load balancing among partners or because of the lack of skills at partner *x* to perform part of the process.

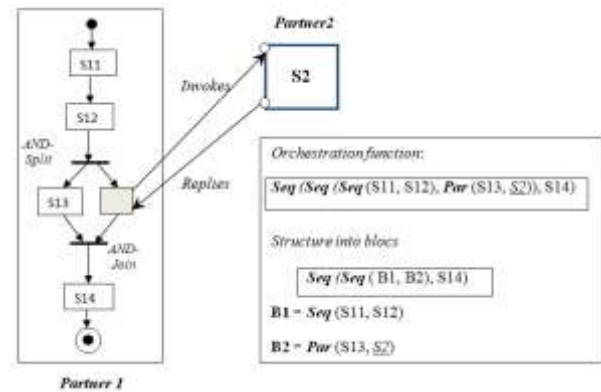


Fig. 10: Illustration of orchestration function in SBCP3

For the *extended* case transfer, the difference is that some activities can be implemented *differently* from one partner to another, while respecting the overall structure of the process and the global functionality covered. This pattern is provided for partners who want to preserve their expertise for some activities in the process that remain invisible from the other partners; this guarantees a certain degree of autonomy and confidentiality. Before describing the patterns SBCP4 (resp. SBCP5) suitable to the "Case-transfer" (resp. the extended case transfer) architecture, we should introduce some basic definitions mainly the notions of *transfer point* and *transfer policy* and explain how to structure the process into services according to transfer points in the IOWF model.

5.4.1 Transfer Point and Transfer Policy

A *Transfer point* is a state of the process where a *case transfer* can eventually occur; it can be each state of the process that guarantees coherent execution of instances when a transfer is done.

In fact, a transfer point should verify the following conditions: (i) it must be before the beginning or after the end of an activity. (ii) It should not interrupt the execution of an activity. (iii) It should not be between a routing operator *Split* and the corresponding operator *Join* that means: whether a parallel or an alternative branch is started in the process, the transfer of a process instance may take place only after synchronization (*Join*).

A *Transfer policy* is conjointly defined by all partners at build time. It defines the set of *transfer points* and expresses a set of *rules* governing the transfer of process instances from one location to

another. A *transfer rule* is associated to a *transfer point* and can be defined by a pair (*condition*, *action*) that means: if the condition is verified, an action of transfer is performed otherwise the instance continues its execution at its current location. An action specifies the location to where the instance will be transferred. Thus depending on the transfer policy, this location can be deterministic or not.

In order to structure an IOWF process obeying to the “Case transfer” architecture into services, our approach is to split each WF into sub-processes at the transfer points and to encapsulate each sub-process into a service (see Fig.11). A sub-process is part of a global WF process that can be composed by a single activity, a

single block of activities delimited by a *Split* operator and the corresponding *Join* operator or a sequence of several activities and/or blocks. A service in this case does not encapsulate the overall WF process but only a *sub-process*. A service can be run locally (if the transfer is not necessary) or relied on the other partner (if the transfer is necessary). At each moment, any process instance is at one location, hence the use of the “XOR” operator in the process model. A case transfer may be done in both directions from partner 1 to partner 2 or vice versa. The transfer points and the direction of transfers are fixed in the *transfer policy*. More details and examples of this approach are described in our previous works [39], [40].

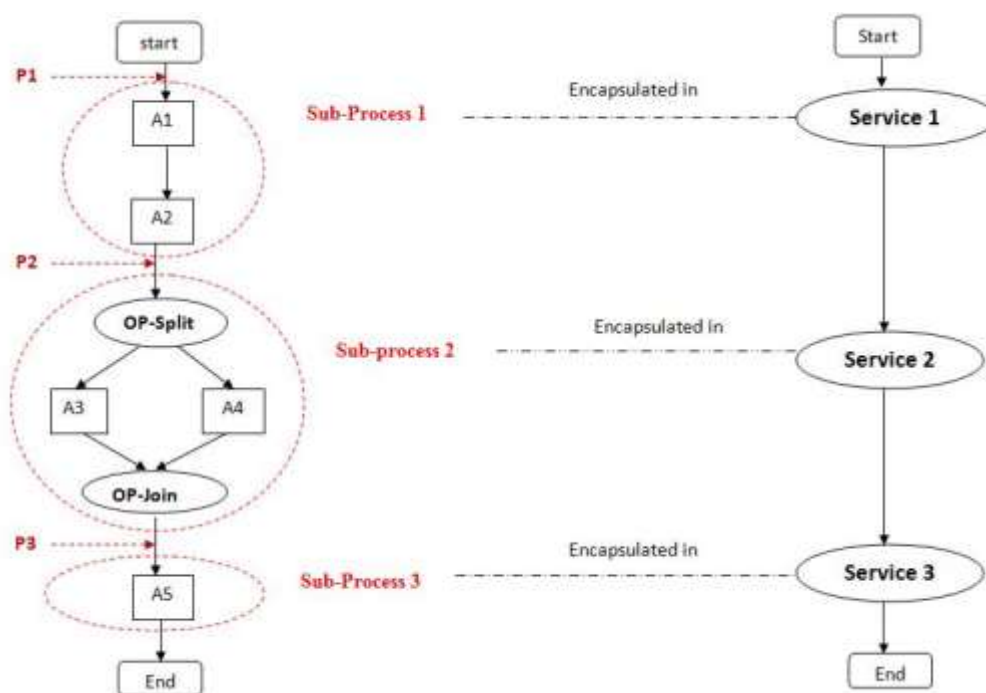


Fig. 11: Illustration of Transfer points and structuring of a WF process into services

An orchestration function for this architecture uses *Seq* and *Exl* operators because the process model turns into a sequence of a certain number of exclusive choices, depending on the number of transfer points in the process. According to a generic schema of Fig. 12, the expression of the orchestration function is *Seq (...Seq (Seq (S11, Exl (S21, S22), ..., Exl (Sn1, Sn2))*.

5.4.2 Managing transfers

For each partner, the control of execution of process instances is done locally by the local engine. Regarding the transfer of cases, we can envisage two modes of control: *decentralized* or *centralized* control [39], [40]. In the first mode, workflows implemented at each partner interact directly between them for transfer of instances; this mode is typically appropriate in case of a simple transfer policy (deterministic rules) and is

realized by injecting exclusive choices in the IOWF model at the transfer points, in order to decide for transfer or not according to transfer conditions. In the second mode, an additional component (a coordinator) is needed in order to manage all transfers to be done between the systems of the partners implied in the IOWF process. So, workflows don't interact directly with each other but they must do this through the *coordinator*. This second mode is appropriate in case of complex transfer policies (non deterministic rules), this can usually occur for load balancing in the system.

5.5 The “Loosely coupled WF ” Pattern – SBP6

The “Loosely coupled” 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 *asynchronous* message exchanges. WF processes

operate essentially independently, but have to interact at given points to exchange data and to ensure a coherent execution of the overall process. An interaction point is attached to a message and then to an interaction activity (invoke or receive) in the process. Fig.13 and Fig.14 bellow schematize the transformation of generic WF schemas into services, using the rules set in the bottom of Fig.15.

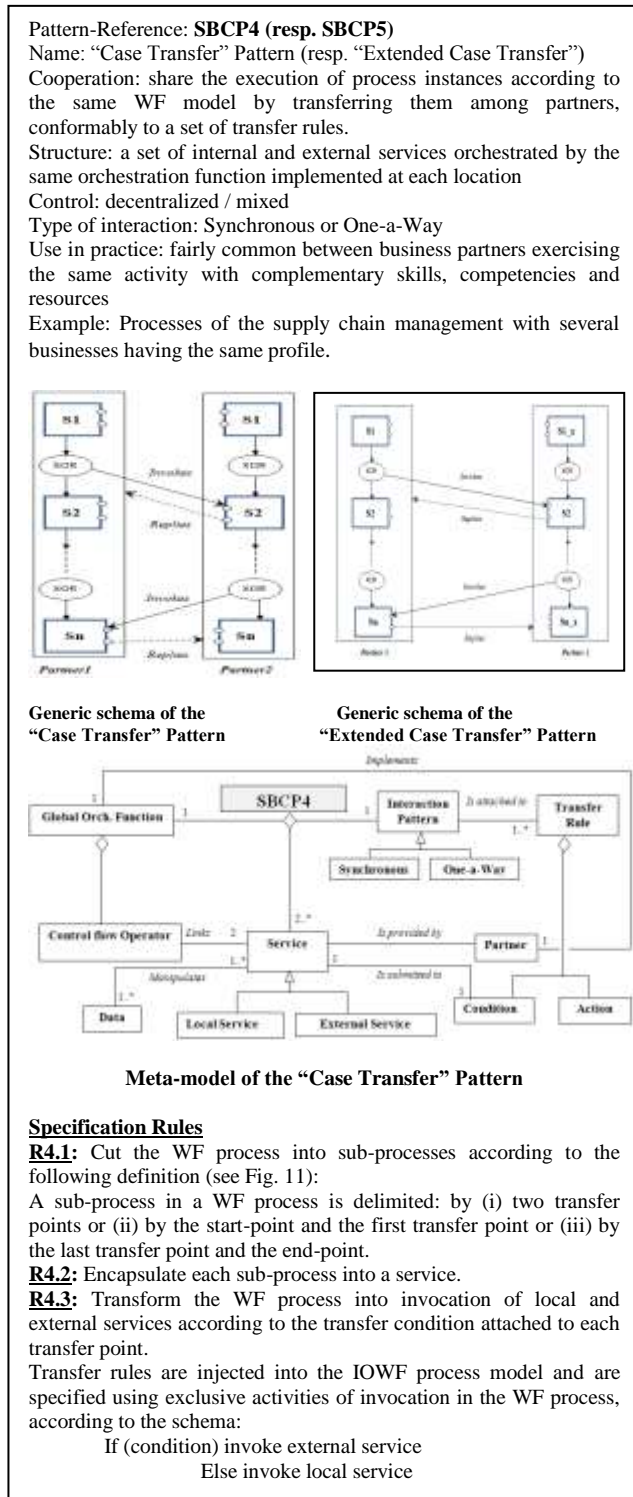


Fig. 12: Description of the "Case Transfer" Pattern- SBCP4 (SBCP5)

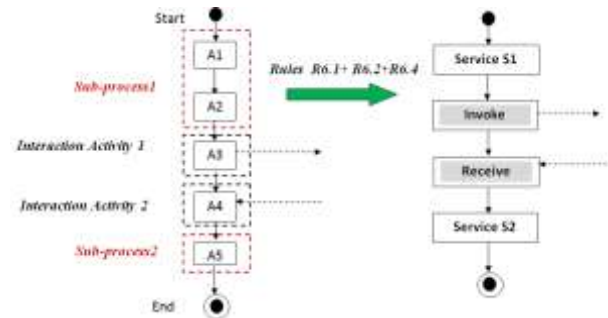


Fig. 13: Transformation of a schema containing sequential blocs

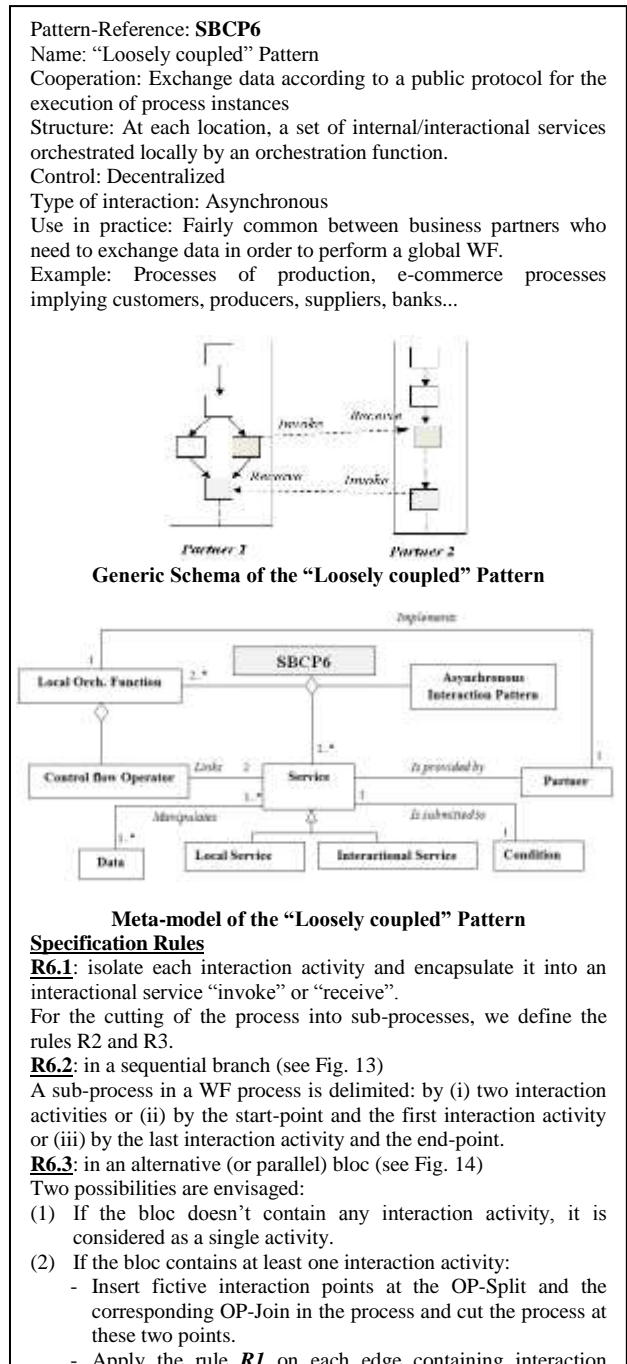


Fig. 15: Description of the "Loosely Coupled" pattern – SBCP5

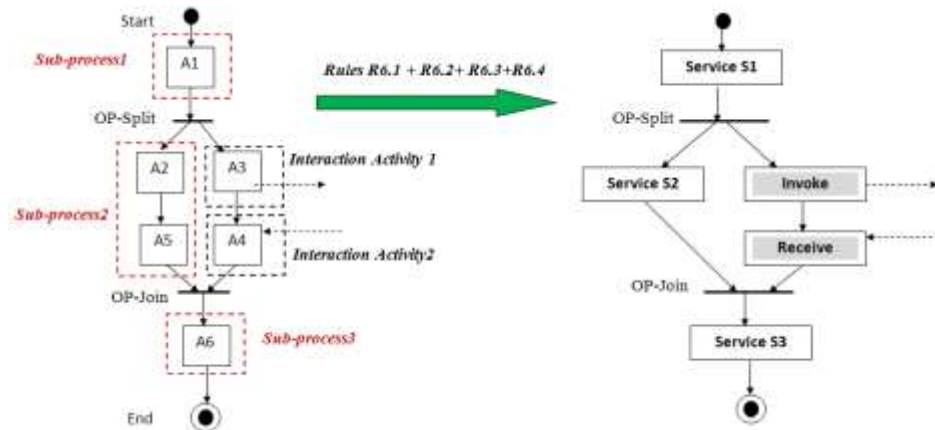


Fig. 14: Transformation of a schema containing parallel or alternative blocks

To obtain an IOWF model obeying to SBCP6, we propose first to isolate the *interaction* activities in the WF process of each partner in order to encapsulate them into *interactional* services. After that, we structure the WF process of each partner into a set of sub-processes to be encapsulated in local services.

The cutting of a WF process into interactional activities and sub-processes is done conformably to the rules set out in the description of the “Loosely coupled” pattern (see Fig.15) and schematized in Fig.13 and Fig.14.

In order to show the feasibility of our approach and to do our tests, we have implemented the proposed cooperation patterns in a framework of cooperation called “S-IOFLOW”. In the next section, we show the general architecture of our framework, its environment of development and the main functionalities that it provides. The process models are stored in repositories of distinct machines which play the roles of client or server depending on the different architectures considered.

VI. The Framework “S-IOFLOW”

“S-IOFLOW” is our cooperation framework that provides a set of wizards for the WF designers in order to build IOWF models obeying to a given SBCP among those considered in our work. Each wizard presents a set of steps to be followed by WF designers in order to realize a specific architecture starting with a set of WF fragments (based on web services) implemented at partner’s sites.

For the development of our framework, 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 cooperation patterns (interconnection of WFs), we have used the API jdom2 that eases the modification of the code BPEL specifying

the WF processes. For the development of the web services to do our tests, we have used the EJB (Enterprise Java Beans). Our framework of cooperation is as modular as possible since we implement separate classes for each cooperation pattern. Furthermore for design, we adopt the MVC (Model-View Controller) pattern that allows the separation between data and their processing. Fig. 16 describes the functional architecture of our framework according to the MVC pattern. Each wizard of the framework displays a set of interfaces to the user; when a user event occurs, the selected view calls the appropriate controller to do the composition by affecting the selected models (i.e BPEL files), then the models notify the concerned views for changes. This allows synchronization between the models and the views that display them. Also, each partner stores in his local servers the BPEL files specifying his business processes and the web services that he provides to the other partners. The cooperation framework is deployed on a common infrastructure where a copy of each BPEL file *selected* for cooperation is created. All changes are done via the appropriate wizard, on the created copies; once the designer *validates* the composition, these changes are reflected on the original BPEL files at the partner’s sites. Also, to check the execution of the composite process obtained, we use test applications. Before validation, a step of updating data flow in the composite process is done in a semi-automatic way via interfaces provided by the wizards. In Table 1 below, we give some implementation details of the cooperation wizards implemented. Since the architecture of deployment is a client/server, we specify for each cooperation pattern the clients and the servers.

The main classes of our framework are BpelFile and ListBpelFile classes which inherit from the class “observable” and all views of the models (detail, graphical, code) inherit from the interface “observer” which is notified by the class “observable” for all changes done on the models. The controller contains a set of classes implementing the set of cooperation patterns described in Section 5; these classes are named “CapacitySharing”, “ChainedExecution”, “Subcontracting”, “CaseTransfer” and

“LooselyCoupled” that inherit from an interface named “Composition”.

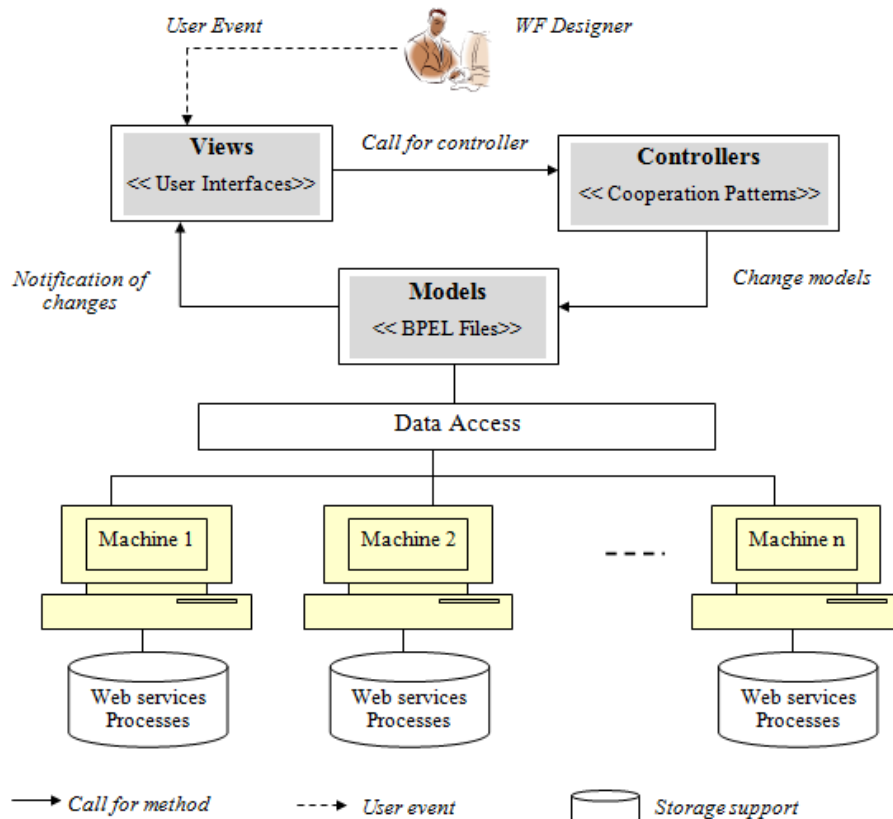


Fig. 16: Functional architecture of the framework according to the MVC pattern

VII. Generalized and Composite Patterns

For all patterns described in the previous sections, we have considered cooperation between two partners but it is possible, using our framework, to build IOWF models involving three or more partners, this is what we call *generalized* cooperation patterns. Typically, it is to build a cooperation between two partners and then to consider the resulting process with the third one to build another cooperation and so on until all processes implied in cooperation are taken into account. For example, for a “Chained execution” (SBCP2), it is to select the first process and the second one to build an IOWF implying the two processes and then to select the resulting process with the third one in the sequence. For a “(extended) Case transfer” (SBCP4, SBCP5), it is to duplicate the same process at each location and to select at each time two processes to define the set of transfer points and transfer rules between them. For a “Loosely coupling” (SBCP6), it is to select at each time, two processes that should interact with each other from the set of processes and define the interaction points between them. For the “Subcontracting” (SBCP3), it is to select the main process and the secondary processes one by one to define the cooperation; let’s notice that for this architecture, a secondary partner can also subcontract part of his process to another partner; this is what we call “multilevel subcontracting”.

Furthermore, our approach allows the construction of more complex IOWF models by reusing existing models that obey to one of the SBCP implemented. The more complex models are obtained by combining two or more SBCP. For example, one can build an IOWF process model P1 obeying to SBCP2 and should subcontract part of the process P1 to another partner providing a process P2 as a composite service. Then, by combining the two models, we obtain a process model P obeying to SBCP2 and SBCP3. The predominant pattern is the pattern that initiates the execution of the composite process and the secondary pattern is the second one. By combining the patterns in pairs and by considering the notions of predominant pattern and secondary pattern, we obtain a set of twenty composite cooperation patterns. Table 2 below describes examples of composite cooperation patterns; a composite pattern is referenced as “CmpSBCPij” where i is to the number of the predominant pattern (SBCPi) and j is the number of the secondary pattern (SBCPj); that means CmpSBCPij is obtained by the combination of SBCPi and SBCPj. Let’s notice that we have implemented some of these patterns such as CmpSBCP23, CmpSBCP32, CmpSBCP24 and CmpSBCP42.

Table 1: Description of the Wizards

The Wizard	Architecture of deployment	Steps	Implementation details
“Capacity Sharing” SBCP1	One Client site (the orchestrator) and a set of Server sites - Centralized control	Step 1: Select the architecture Step 2: Select partner links (Services) Step 3: Define the global process ----- Step 4: Assign parameters -----	Specify the control flow between services using an appropriate interface that provides the basic operators (sequence, if, flow) The input parameters of a service should be the output of a service preceding it.
“Chained Execution” SBCP2	All partner’s sites are Client/Server except the last one which is only a Server . - Decentralized control	Step 1: Select the architecture Step 2: Select the processes to chain ----- Step 3: Assign parameters ----- Step 4: Validate the composition	Specify the first process and the second process The input parameters of a second process should be the output of the first process.
“Subcontracting” SBCP3	The site of the main partner is a Client/Server and the sites of secondary partners are only Servers - Hierarchized control	Step 1: Select the architecture Step 2: Select the processes ----- Step 3: Do correspondence between invocation of an empty service (in the main process) and a secondary process Step 4: Assign parameters ----- Step 5: Validate the composition	Specify the main process with invocation of empty services and specify the secondary processes Substitute each invocation of an empty service by invocation of the appropriate secondary process The input parameters of a secondary process should be the output of a service its invocation in the main process.
“Case-Transfer” SBCP4 “Extended Case-Transfer” SBCP5	All partner’s sites are Client/Server - Decentralized or mixed control	Step 1: Select the architecture Step 2: Select partners and process Step 3: Add transfer points in the process and create sub- processes Step 4: Validate the composition	For each transfer point , we should define through an interface displayed by the wizard: - its location in the process (after/before any activity?), - the condition (using a simple editor provided by the wizard) and - the action (add an exclusive choice in the process model) - create sub-processes
“Loosely Coupled” SBCP6	All partner’s sites are Client/Server Decentralized control	Step 1: Select the architecture Step 2: Select processes Step 3: Define the interactional services in each process Step 4: Insert interactional services into the models Step 5: Assign parameters ----- Step 6: Validate the composition	Two types of interactional services are defined: ServiceSend with output data and ServiceReceive with input data The input parameters of a receiver process (ServiceReceive) should be the output of a sender process (ServiceSend).

Table 2: Examples of Composite Patterns

Pattern Reference	Pattern Name	Generic Schema	Predominant Pattern	Secondary Pattern	Control Mode	Description
CmpSBCP13 One of the services composing the global IOWF corresponds to another IOWF obeying to SBCP3	A Subcontracting in Capacity Sharing		SBCP1	SBCP3	Centralized/ Hierarchized	Invoke the main process of the inIOWF(2) from IOWF(1)
CmpSBCP31 The main WF subcontracts one of its activities to a group of partners implementing a global IOWF obeying to SBCP1	A Capacity Sharing in Subcontracting		SBCP3	SBCP1	Hierarchized/ Centralized	Invoke the global process obeying to SBCP1 considered as a secondary process from the main process

VIII. Comparison of Approaches

In Table 3 below, we present a comparison of our framework with some approaches proposed in the literature. For each approach, we give some descriptive details and we define three criteria for comparison: the cooperation type supported, IOWF-architectures supported and aspects of flexibility provided by each approach. The cooperation type can be planed or dynamic; planed cooperation means that partners agree together to cooperate and we don't need to discover them and to select them in the registry of publication which is necessary in a dynamic cooperation, because partners are not known a priori. Many approaches are suitable for dynamic cooperation that usually correspond to occasional and non-durable cooperation; other approaches are suitable to planed cooperation (which is our concern) that corresponds to well defined and durable cooperation which is more realistic in the B2B area, for the realization of big projects. The second criteria concern IOWF-architectures supported (on

Table 1, Type1, Type2, Type3, Type4, Type5, Type6 refer respectively to Capacity sharing, Chained execution, Subcontracting, Case transfer, Extended case transfer and Loosely coupled), we can see that all the proposed approaches support only a sub-set of the architectures implemented in our framework "S-IOWFLOW". Regarding the third criteria, we can see that the approaches suitable to dynamic cooperation provide flexible mechanisms in the phase of selection of partners; also, some of them allow internal adaptation of services. The approaches suitable to planed cooperation are rigid and are based on predefined protocols. Our framework "S-IOWFLOW" provides three aspects of flexibility: (i) the selection of the IOWF-architecture to build; (ii) the definition of composite cooperation patterns by reusing elementary ones to build more complex IOWF models, (iii) our framework is extended with adaptation and evolution modules for structural and functional adaptation of IOWF models; some adaptation and evolution patterns are described in [41], [42].

Table 3: Comparison of Approaches

Approach	Cooperation Type	Description	Types of IOWF-architectures supported						Aspects of flexibility
			Type 1	Type 2	Type 3	Type 4	Type 5	Type 6	
CrossFlow [15]	Dynamic	Based on externalization and publication of services. Requires a phase of establishing a contract before the construction of the infrastructure of execution of the global process.			✓				Requires a phase of setting parameters (new infrastructure) for any new request.
Wise [43]	Planed	Based on composition of services published by partners via internet.			✓				The interfaces of communication between the different workflows are rigid.
CrossWork [16]	Dynamic	Construction of a global WF, using services provided by several partners. Uses a pattern-based approach	✓						Not enough flexible Requires the generation of interfaces for participants for each execution.
Synchronization Point [44]	Planed	Based on cooperation rules defined in a contract. Coordination of the progression of the WF by defining synchronization points in the process model.						✓	Possibility to update the synchronization point at runtime (a adaptation of the process model)
Pyros [17]	Dynamic	Publication of services. Orchestration of services using a global WF that plays the role of central coordinator.	✓						The orchestration model is adaptable. Possibility of internal adaptation of services provided by the partners.
ACE-Flow [45]	Planed	Specification and execution of a global WF that coordinates the invocation of different WFs provided by partners. (Workflow of workflows)	✓						The relations between the WFs of providers and consumers are predefined in a global database.
FOCAS [22]	Dynamic	Approach for the construction of process oriented applications based on services orchestration. Centralize or distributed execution. Not especially dedicated for B2B applications but can support some architectures.	✓	✓	✓			✓	Possibility of functional and non-functional extensions of services
CoopFlow [9]	Dynamic	Inspired from the SOA approach. Based on publication, interconnection and control execution of workflows	✓					✓	Flexibility in the selection of workflows Provides a generic adapter allowing direct interconnection without additional code.
FErOS [46]	Dynamic	Framework for the construction of service-oriented enterprise based on dynamic composition of services. Automatic selection and semi-automatic composition of services.	✓	✓	✓				Flexibility in the selection of services based on functional and non-functional characteristics.
Multi-contextual orchestration [47]	Dynamic	Based on the SOA and Object paradigms Construction of a common collaborative process to support interoperability of services published by several partners in several contexts.	✓						Flexibility in the selection of services and the construction of a collaborative process

IX. Conclusion and Other Works

The current paper deals with WF cooperation. Our contribution consists in the definition and the implementation of a set of *cooperation patterns* based on services (called SBCP) in order to meet specific IOWF- architectures defined in the literature [7][8]; the goal is to obtain IOWF models flexible enough thanks to the SOA characteristics. These basic architectures define different cooperation schemas obeying to different modes of execution control: centralized, decentralized or hierarchized. For the development of our solution, we have adopted a pattern-based approach to define and implement the different patterns of WF cooperation. The pattern-based approach guarantees modular and reusable implementation; by reusing the elementary patterns implemented, we can particularly build generalized and composite cooperation patterns which is in our opinion, an interesting point in our contribution. Because of the length of the paper, we gave only an example of composite cooperation patterns. The proposed patterns have been implemented in a framework of cooperation called “S-IOFLOW” which is as modular as possible since we implement separate classes for each cooperation pattern. Furthermore, for the development of our framework, we adopt the MVC pattern that eases the maintainability and the extensibility of the framework and allows the separation between data and their processing.

Regarding the second issue of our research that concerns the adaptability and evolutivity of process models obeying to the SBCP defined, we have classified our adaptation patterns in three categories according to the three dimensions (services, control flow and interaction) defining a SBCP. We have implemented adaptation modules that can be interfaced with “S-IOFLOW” and composed by a set of adaptation/evolution patterns applied to BPEL process models resulting from cooperation.

Acknowledgments

We would like to thank our students Bouchekir Redouane, and Hermez Dalil for their participation in the design and implementation of the cooperation framework.

References

- [1] Van Der Aalst W. Workflow Management: Models, Methods and Systems. The MIT Press. Cambridge, Massachusetts, London, 2002.
- [2] Alonso G, Casati F, Kuno H. Web services: concepts, architectures and applications. Springer Verlag, Germany, 2004.
- [3] Papazoglou] M. P, Van Den Heuvel W. J. Service Oriented Architectures: approaches, technologies and research issues. The VLDB Journal, vol.16, pp 389-415, 2007
- [4] Voorhoeve M, Van Der Aalst W. Ad-hoc Workflow: Problems and Solutions. In R. Wagner, editor, Database and Expert Systems Applications, 8th. International Workshop, DEXA'97 Proceedings, 36–40, Toulouse, France, September 1997.
- [5] Kiepuszewski A.H.M, ter Hofstede, Bussler C. On Structured Workow Modelling. In B. Wangler and L. Bergman, editors, Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAiSE'2000), 2000, LNCS(1789), 431-445, Springer-Verlag,
- [6] Eder J, Gruber W, A meta model for structured workflows supporting workflow transformations. Proceedings of the 6th East European Conference on Advances in Databases and Information Systems (ADBIS 2002), 326–339, Bratislava, Slovakia, 2002.
- [7] Van Der Aalst W. Process oriented architectures for electronic commerce and interorganizational WF. Journal of Information systems, 1999, 24 (9).
- [8] Van Der Aalst W. Loosely Coupled Interorganizational Workflows: modeling and analyzing WFs crossing organizational boundaries. Journal of Information and Management, March 2000, 37(2) , 67-75.
- [9] Chebbi I. CoopFlow : an approach for ascendant cooperation of workflows in virtual enterprises. Phd Thesis, National Institute of Telecom, France, 2007.
- [10] Peltz C. Web Services Orchestration and Choreography. IEEE Computer, 2003, 36 (10), 46-52.
- [11] Amirereza T. Web Service Composition Based Interorganizational Workflows. Sudwestdeutscher Verlag fur Hochschulschriften edition, 2009
- [12] Jordan D, Evdemon J. Web services business process execution language V.2.0. W3C. 2006.
- [13] Leymann F, Roller D, Schmidt M.T. Web Services and Business Process Management. IBM Systems Journal 2002, 41(2).
- [14] Gorton S, Montangero C, Reiff-Marganiec S, Semini L, StPowla: SOA, Policies and Workflows. ICSOC workshops, 2009, LNCS (4907), 351-362.
- [15] Grefen P, Aberer K, Hoffer Y, Ludwig H. CrossFlow: Cross-organizational workflow management for service outsourcing in dynamic virtual enterprises. IEEE Data Engineering Bulletin, 2001, 24(1), 52–57.
- [16] Mehandjiev N. I, Stalker K, Fessl , Weichhart G. Interoperability contributions of CrossWork. In invited short paper to Proceedings of INTEROP-

- ESA'05 Conference, Geneva, February 2005. Springer-Verlag.
- [17] Belhajjame K, Vargas-Solar G, Collet C. Pyros - an environment for building and orchestrating open services. In Proceedings of the IEEE International Conference on Services Computing, USA, 2005, 155–164.
- [18] Casati F. and Shan M., Dynamic and adaptive composition of e-services. *Information Systems*, 2001, 26(3), 143–163.
- [19] Ba ĩa K, Benali K, Godart C, DISCOBOLE: A service architecture for interconnecting workflow processes. *Computers in Industry*, 2006, 57(8-9): 768-777.
- [20] Boukadi K. Interenterprise cooperation at demand: a flexible approach based on adaptable services. Phd Thesis, ENSM, Saint-Etienne. France, 2009.
- [21] Heorhi R. Service Composition in Dynamic Environments: From Theory to Practice, Phd thesis, University of Trento, december 2012.
- [22] Pedraza Ferreira G. R. FOCAS : an extensible framework for the construction of process oriented applications. Phd Thesis, University of Grenoble 1, France, 2009.
- [23] Sadiq S.W., Orlowska M.E. On capturing Exceptions in workflow process models. In proceedings of ER'2001.
- [24] Meng J, Su S.Y.W, Lam H, Helal A, Xian J, Liu X, Yang S. DynaFlow: a dynamic inter-organisational workflow management system. *Int. J. Business Process Integration and Management*, 2006, 1(2), 101–115.
- [25] LÉVESQUE E. Adaptation of collaborative processes by coordination of changes and instance migration. Phd Thesis, University of Quebec, Montréal, 2011.
- [26] He Q, Yan Y, Jin H. Adaptation of web service composition based on WF patterns. In proceedings of Service Oriented Computing- ICSOC, 2008.
- [27] Dähring M, Zimmermann B, Karg L. Flexible Workflows at design- and Runtime using BPMN2 Adaptation Patterns. In proceedings of BIS'2011- Springer, 2011.
- [28] Weber B, Reichert M, Rinderle-Ma S. Change patterns and change support features- Enhancing flexibility in PAIS. *Journal of Data & Knowledge Engineering* 2008,(66), 438-466.
- [29] Muller R, Greiner U, Rahm E. AGENT-WORK: a workflow system supporting rule-based workflow adaptation. In *journal of Data and Knowledge Engineering*, 2004, 51(2), 223-256.
- [30] Dähring M, Zimmermann B, Godehardt E. Extended workflow flexibility using rule-based adaptation patterns with eventing semantics. In *proc. of INFORMATIK'10*, 2010, 216-226.
- [31] Pesic M, Schonenberg MH, Sidorova N, Van der Aalst W. Constraint-based workflow models: Change made easy. In Proceedings of the OTM Conference CoopIS'2007. 2007. In LNCS(4803), 77–94, Springer-Verlag, Berlin,
- [32] Tragatschnig S, Zdun U. Runtime Process Adaptation for BPEL Process Execution Engines. 15th IEEE International EDOC Workshops, 2011.
- [33] Van Der Aalst W, ter Hofstede W.M.P, Kiepuszewski A.H.M, Barros, B.A.P. Workflow Patterns. *DAPD*, 2003, 14(1), 5-51.
- [34] Russell N, Van Der Aalst W, ter Hofstede W.M.P. Exception handling patterns in process-aware information systems. In: CAiSE'06 (Luxembourg), 2006, 288-302.
- [35] Khadka R. Model-Driven Development of Service Compositions: Transformation from Service Choreography to Service Orchestrations, Master thesis, University of Netherlands, 2010.
- [36] AIT-CHEIK-BIHI W. Model oriented approach for verification and performance evaluation of services interoperability and interactions. Phd Thesis, University of Belford-Montbeliard, 2012.
- [37] W. Fdhila : Optimized decentralization and synchronization of Inter-organizational business processes. Phd Thesis, University Henri Poincaré– Nancy 1, 2011.
- [38] Bernauer M, Kappel G, Kramler G, Retschitzegger W. Specification of Interorganizational Workflows — A Comparison of Approaches, 7th World Multiconference on Systemics, Cybernetics, and Informatics, Orlando, Florida, July 2003, 30-36
- [39] Boukhedouma S, Alimazighi Z, Oussalah M, Tamzalit D. SOA based approach for interconnecting workflows: application to case transfer. In proceedings of INFORSID 2011, 43-58.
- [40] Boukhedouma S, Alimazighi Z, Oussalah M, Tamzalit D. 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.
- [41] Boukhedouma S, Alimazighi Z, Oussalah M, Tamzalit D. Adaptability of service-based workflow models : the chained execution architecture. In proceedings of BIS'2012, Lithuania. W. Abramowicz et al. (Eds.) LNBIP 117, Springer-Verlag.
- [42] Boukhedouma S, Oussalah M, Alimazighi Z, Tamzalit D. Flexible loosely coupled workflows

using SOA. In proceedings of AICCSA'2013, Fes, Maroc.

- [43] Lazcano A, Alonso G, Schuldt H, Schuler C. The Wise approach to electronic commerce. *International Journal of Computer Systems Science & Engineering*, special issue on Flexible Workflow Technology Driving the Networked Economy, 2000, 15(5).
- [44] Perrin O, Godart C. A model to support collaborative work in virtual enterprises. *Data Knowledge Engineering*, 2004, 50(1), 63–86.
- [45] ACE-FLOW. Project homepage, <http://www.ifi.unizh.ch/dbtg/projects/aceflow/index.html>, 1999.
- [46] haari S. Interconnecting interenterprise processes : a service-oriented approach. Phd Thesis, EDIIS, Lyon, France, 2008.
- [47] Esper A. Integration of SOA and object approaches for modeling a coherent orchestration of services. Phd Thesis, INSA, Lyon, France, 2010.

Workflows", *International Journal of Information Technology and Computer Science(IJITCS)*, vol.6, no.4, pp.1-18, 2014. DOI: 10.5815/ijitcs.2014.04.01

Authors' Profiles

Saida Boukhedouma is a Teacher/Researcher at USTHB University, member of the ISI team in the LSI laboratory. Actually, her works are directed towards the flexibility of inter-organizational business processes using the SOA paradigm which is the main focus of her PHD thesis.

Mourad Chabane Oussalah is a full Professor of Computer Science at the University of Nantes and the head of the software architecture modeling team. His research concerns software architecture, object architecture and their evolution.

Zaia Alimazighi is a full Professor of Computer Science at USTHB University, team leader at the LSI laboratory and dean of the Electrical and Computer Science faculty. Her current research concentrates on cooperative Information Systems modeling, inter-organizational business process modeling.

Dalila Tamzalit is an Assistant Professor at the University of Nantes in France. Her main research interest concerns software evolution foundations and methodologies. These last years, she focuses on Software Architecture Evolution.

How to cite this paper: Saida Boukhedouma, Mourad Oussalah, Zaia Alimazighi, Dalila Tamzalit, "Service Based Cooperation Patterns to Support Flexible Inter-Organizational