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 reusable 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](image-url)
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.
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 sub-process into a service.

Fig. 2: Generic schemas of interaction

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) 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’s 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.

![Fig. 5: Basic Control Flow Operators](image)

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(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.

Pattern-Reference: SBCP1
Name: “Capacity Sharing” Pattern
Cooperation: Share the execution of a global business process implemented at one location.
Control: Centralized
Structure: A global orchestration of services provided by different partners
Type of interaction: Synchronous interaction between the client (the orchestrator) and the business services.
Use in practice: used in dynamic cooperation with techniques of service orchestration.

![Generic Schema of the “Capacity Sharing” Pattern](Image 321x419)

![Meta-model of the “Capacity Sharing” Pattern](Image 351x540)

**Specification Rules**

**R1.1:** Encapsulate each WF into services.

**R1.2:** Specify the global orchestration function (the control flow between services).

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 Si. 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 SBCP2.

5.3 The “Subcontracting” Pattern – SBCP3

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 SBCP3, 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. SBCP3 is described by the meta-model of Fig. 9.

To illustrate the concept of global orchestration function for SBCP3, 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. Partner 1 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.
Pattern-Reference: SBCP3
Name: “Subcontracting” Pattern
Cooperation: Externalization of services to other partners
Structure: A set of internal and external services orchestrated by a global orchestration function implemented at the main partner and a set of local orchestration functions, each of which implemented at the corresponding secondary partner.
Control: Hierarchized
Type of interaction: Synchronous
Use in practice: Fairly common between business partners with complementary skills and competencies.
Examples: Processes of pharmaceutical production, automotive processes, manufacturing and assembly of integrated circuits.

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.

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 \((\text{condition}, \text{action})\) 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 \(\text{Split}\) operator and the corresponding \(\text{Join}\) 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].

An orchestration function for this architecture uses \(\text{Seq}\) and \(\text{Exl}\) 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 \(\text{Seq} (\ldots \text{Seq (S11, Exl (S21, S22), \ldots, Exl (Sn1, Sn2))})\).

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 – SBCP6

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 below schematize the transformation of generic WF schemas into services, using the rules set in the bottom of Fig. 15.

Pattern-Reference: SBCP4 (resp. SBCP5)
Name: “Case Transfer” Pattern (resp. “Extended Case Transfer”)
Cooperation: share the execution of process instances according to the same WF model by transferring them among partners, conformably to a set of transfer rules.
Structure: a set of internal and external services orchestrated by the same orchestration function implemented at each location
Control: decentralized / mixed
Type of interaction: Synchronous or One-a-Way
Use in practice: fairly common between business partners exercising the same activity with complementary skills, competencies and resources
Example: Processes of the supply chain management with several businesses having the same profile.

Pattern-Reference: SBCP6
Name: “Loosely coupled” Pattern
Cooperation: Exchange data according to a public protocol for the execution of process instances
Structure: At each location, a set of internal/interactional services orchestrated locally by an orchestration function.
Control: Decentralized
Type of interaction: Asynchronous
Use in practice: Fairly common between business partners who need to exchange data in order to perform a global WF.
Example: Processes of production, e-commerce processes implying customers, producers, suppliers, banks...

Generic schema of the “Case Transfer” Pattern

Generic schema of the “Extended Case Transfer” Pattern

Meta-model of the “Case Transfer” Pattern

Specification Rules
R4.1: Cut the WF process into sub-processes according to the following definition (see Fig. 11):
A sub-process in a WF process is delimited: by (i) two transfer points or (ii) by the start-point and the first transfer point or (iii) by the last transfer point and the end-point.
R4.2: Encapsulate each sub-process into a service.
R4.3: Transform the WF process into invocation of local and external services according to the transfer condition attached to each transfer point.
Transfer rules are injected into the IOWF process model and are specified using exclusive activities of invocation in the WF process, according to the schema:
If (condition) invoke external service
Else invoke local service

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

Fig. 13: Transformation of a schema containing sequential blocs

Fig. 14: Description of the “Loosely Coupled” Pattern – SBCP5

Specification Rules
R6.1: Isolate each interaction activity and encapsulate it into an interactional service “invoke” or “receive”.
For the cutting of the process into sub-processes, we define the rules R2 and R3.
R6.2: In a sequential branch (see Fig. 13) A sub-process in a WF process is delimited: by (i) two interaction activities or (ii) by the start-point and the first interaction activity or (iii) by the last interaction activity and the end-point.
R6.3: In an alternative (or parallel) bloc (see Fig. 14) 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.
R6.4: Encapsulate each sub-process within an internal service.
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”.

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

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

### Table 2: Examples of Composite Patterns

<table>
<thead>
<tr>
<th>Pattern Reference</th>
<th>Pattern Name</th>
<th>Generic Schema</th>
<th>Predominant Pattern</th>
<th>Secondary Pattern</th>
<th>Control Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompSBCP13</td>
<td>One of the services composing the global IOWF corresponds to another IOWF obeying to SBCP1</td>
<td>A Subcontracting in Capacity Sharing</td>
<td>SBCP1</td>
<td>SBCP3</td>
<td>Centralized/ Hierarchized</td>
<td>Invoice the main process of the in IOWF(2) from IOWF(1)</td>
</tr>
<tr>
<td>CompSBCP3</td>
<td>The main WF subcontracts one of its activities to a group of partners implementing a global IOWF obeying to SBCP1</td>
<td>A Capacity Sharing in Subcontracting</td>
<td>SBCP3</td>
<td>SBCP1</td>
<td>Hierarchized/ Centralized</td>
<td>Invoke the global process obeying to SBCP1 considered as a secondary process from the main process</td>
</tr>
</tbody>
</table>
### 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-IOFLOW”. 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-IOFLOW” 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>
<thead>
<tr>
<th>Approach</th>
<th>Cooperation Type</th>
<th>Description</th>
<th>Types of IOWF-architectures supported</th>
<th>Aspects of flexibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>CrossFlow [15]</td>
<td>Dynamic</td>
<td>Based on externalisation and publication of services.</td>
<td>Type 1</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Requires a phase of establishing a contract before the execution of the</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>process.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Work [43]</td>
<td>Planned</td>
<td>Based on composition of services published by partners via internet.</td>
<td>Type 2</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CrossWork [16]</td>
<td>Dynamic</td>
<td>Construction of a global WF, using services provided by several partners.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Uses a partners based approach.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronization</td>
<td>Planned</td>
<td>Based on cooperation rules defined in a contract.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Polar [44]</td>
<td></td>
<td>Coordination of the progression of the WF by defining synchronization points in the process model</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Orientation of services using a global WF that plays the role of central</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>coordinator.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACE-Flow [45]</td>
<td>Planned</td>
<td>Specification and execution of a global WF that coordinates the invocations of services provided by partners.</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FOCAS [22]</td>
<td>Dynamic</td>
<td>Approaches for the construction of process oriented applications based on services orchestration.</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Centralized or distributed execution. Not especially dedicated for IS/IS applications but can support some architecture.</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Based on publication, interconnection and control of execution of workflows.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOx [46]</td>
<td>Dynamic</td>
<td>Framework for the construction of service-oriented enterprises based on dynamic composition of services.</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Automatic selection and semi-automatic composition of services.</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-contextual</td>
<td>Dynamic</td>
<td>Based on the SOA and Object paradigms.</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>orchestration</td>
<td></td>
<td>Construction of a common collaborative process to support interoperability of services published by several partners.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Comparison of Approaches
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 Hervez Dalil for their participation in the design and implementation of the cooperation framework.

References


[23] Sadiq S.W., Orlowska M.E. On capturing Exceptions in workflow process models. In proceedings of ER’01.


[42] Boukheadouma S, Oussalah M, Alimazighi Z, Tamzali D. Flexible loosely coupled workflows
using SOA. In proceedings of AICCSA’2013, Fes, Maroc.


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.