ENHANCING BUSINESS COLLABORATIONS WITH CLIENT-ORIENTED PROCESS CONTROL

SAMUIL ANGELOV, JOCHEM VONK, PAUL GREFEN
School of Industrial Engineering
Eindhoven University of Technology
Den Dolech 2, P.O. Box 513, 5600 MB
Eindhoven, The Netherlands
{s.angelov, j.vonk, p.w.p.j.grefen}@tue.nl

KRISHNAMURTHY VIDYASANKAR
Department of Computer Science
Memorial University of Newfoundland
NL A1C 5S7, P.O. Box 4200
St. John’s, Canada
vidya@mun.ca

Received 19 March 2010
Revised 6 October 2010

Nowadays, business collaborations have to be highly dynamic and flexible to allow companies to operate efficiently and effectively in complex and volatile markets. To increase the business agility of service consumers, it is fundamental that service providers enhance the visibility of parts of their collaborative processes. Service providers are required to release both the process structures of the services offered and their status during execution. To further increase the flexibility of business collaborations, certain control over the process execution has to be offered to service consumers. In this paper, we present a framework for the support of process control in cross-organizational settings. We specify the control primitives that can be used to exert control on activities and processes before, during and after their execution. These primitives empower service consumers to postpone activity and process executions, bypass minor activities, repeat their execution, etc. We describe an approach to the support of these control primitives by service providers. We demonstrate the application of our framework with a case study from the healthcare domain. A proof-of-concept prototype implementation based on Web service technology is presented.

Keywords: business process, service, process control, collaboration, inter-organizational cooperation, cross-organizational process, virtual enterprise, Web service, process interference.

1. Introduction

To stay competitive in fast-changing markets with increasingly complex products, companies engage in highly dynamic and complex business collaborations. However, engaging in dynamic and complex relationships comes at a price. Companies have to increase their agility and speed and improve their business intelligence to be able to adequately react in these new business settings. A common approach to address this need for increased efficiency and effectiveness in business collaborations is connecting the
information systems of a company with those of their partners. In the context of cooperative information systems, availability of sufficient and up-to-date information during a business relation is crucial for a company for its adequate and intelligent operation. To serve this information need of service consumers, service providers start disclosing to their consumers public (external) views on the processes that implement their services. A public view presents to a service consumer a limited view of the actual process performed by the service provider. A public view is derived by aggregating, abstracting, and hiding certain activities from the business process of the service provider (i.e., its private view). The public view allows providers to share information about the main activities that are performed during a service delivery and on the status of these activities, while omitting anything confidential or irrelevant for the consumer information. Service providers start offering the possibility for consumers to monitor the execution of the public view of their process, thereby allowing them to quickly react and adapt their own processes to changes in the business context.

Support for advanced cross-organizational process monitoring has received substantial attention in both research and industry (e.g., in Refs. [8, 31, 52, 12, and 66]). Process monitoring provides a basis for improved internal operation efficiency and effectiveness at the consumer side and for improved synchronization between the parties. However, process monitoring on its own does not suffice to fully satisfy the needs of parties in cross-organizational relationships. Parties may have to react on the information obtained through monitoring. A reaction may be caused by failure of the partner to comply with a contractual agreement, by the need to influence the process execution at the partner side due to certain internal developments, or it may be simply an expression of preference for the process execution. As an example, consider the automotive industry. The manufacturing of car parts/components is outsourced to different suppliers. The OEM (Original Equipment Manufacturer) assembles these parts into a complete car. Nowadays, many OEMs implement a just-in-time inventory strategy (also called Toyota Production System), where parts are delivered only at the time they are needed in the assembly process, hence minimizing costs for warehousing and optimizing the assembly process. Suppose the OEM discovers, through the available monitoring mechanisms, a quality problem or a delay at one of its suppliers. Until the problem gets resolved, the OEM should put all other outsourced manufacturing processes on hold so that parts are not shipped (as they cannot be stored at the OEM). After the problem is solved, these outsourced manufacturing processes can be resumed again.

We call the interference by a consumer in the process offered by a service provider “cross-organizational process control" (or shortly “process control”). Offering certain control to the service consumer over the execution of the agreed public processes is vital for the utilization of the information obtained through monitoring. By offering the possibility to control the process executed by the provider, the consumer obtains a more flexible, potentially more efficient and effective service. For the provider, offering this opportunity can mean obtaining a competitive advantage over similar services offered by others. However, research on control over execution of public processes has
been limited. Most studies consider cross-organizational process control only as a coordination mechanism in which the start of process execution is requested by the process consumer. In Ref. 20, preliminary results on advanced process control concepts are presented. The necessity of providing control to service consumers and the challenges for it in the context of Web Services are discussed in Ref. 22. Currently, an in-depth and structured research on cross-organizational process control is missing.

The importance of cross-organizational process control can be also inferred by using an analogy with studies on information systems. An information system comprises of three general components, i.e., processing, monitoring and controller components. As discussed, existing research on cross-organizational systems addresses two of these components, i.e., the processing component that transforms input into desired output (the provider process in this case) and the monitoring (feedback) component, which gathers information that can be used to evaluate the processing itself. However, the controller component that is used to steer the processing for improving the desired output is not addressed in cross-organizational collaborations. An additional complicating factor in the case of cross-organizational information systems is the presence of an external (public) and private process specifications. Service consumers exercise control over the public view of an executing process, but the control invocation has to be handled by the service provider in its actual (private) process.

In this paper, we present a framework for the support of cross-organizational process control in process-intensive business collaborations. The framework consists of two elements, i.e., a set of control primitives and a specification for their support by the process provider. These two elements are fundamental for the design of solutions for cross-organizational process control. The control primitives are possibilities for interference in a cross-organizational process that are valuable for a process consumer and which a process provider can offer. We distinguish control primitives that can be applied on single activities and on complete public processes (or subprocesses). The specification for their support defines process constructs that have to be part of the private process specification of a service provider in order to be able to handle the control primitive applied by a service consumer on its public process specification. Our framework provides the theoretical ingredient required for the realization of cross-organizational process control scenarios supported by information systems. The application of the framework for the design of process control in cross-organizational business processes is illustrated through an example from the healthcare domain. We present a software prototype that realizes our example based on existing Web service and workflow technologies. The prototype implementation demonstrates the implementability of our approach and is a first step towards the resolution of a number of run-time concerns for implementing cross-organizational process control. This paper extends Ref. 7 by providing a more thorough and complete analysis of the control primitives, adding control primitives that can be exerted on the (sub)process level, and a more detailed specification for their support by the providers. A prototype description has been added as well.
The paper is structured as follows. In Section 2, we introduce the underlying theoretical background and our approach taken to design the framework for cross-organizational process control. In Sections 3 and 4, we present the control primitives on activities and processes, respectively. In Section 5, we apply our approach to a case from the healthcare domain. Section 6 contains a description of our prototype implementation and discusses issues for the run-time support of cross-organizational process control. We discuss related work in Section 7. The paper ends with conclusions and an outlook to our future research in relation to cross-organizational process control.

2. Background and approach

This research builds on three existing research directions from the domain of cross-organizational process collaborations, i.e., “external process specification”, “electronic contracting”, and “external process monitoring”. In this section, we first briefly discuss these three domains. Based on this discussion, we next present our approach for the definition of our framework for cross-organizational process control.

2.1. Research background

In business collaborations, providers share relevant parts of their processes with consumers to allow them to be aware of the activities that will be performed by the process provider and to monitor the states of these activities during the business collaboration. However, private processes should not be directly disclosed as they may reveal company sensitive information or may contain activities that are irrelevant for the counterparty\(^\text{21,68}\). In Ref. 21, a three level framework for process specification is proposed, distinguishing external, conceptual, and internal process levels. At the conceptual level, the process specification is technology independent, specifying the private process that will be performed by the party. Process specifications at the external level contain the activities that will be disclosed to an external party. External activities are derived by hiding and aggregating activities from the conceptual level. An external activity contains all conceptual activities between the starting conceptual activity and ending conceptual activity for this external activity and each conceptual activity is part of one external activity (see Ref. 15 for a detailed description of the rules for deriving an external level process specification based on a conceptual process specification). We depict an example process specification at an external and conceptual level in Fig. 1. At the internal level, the process specification reflects an adaptation of the conceptual process specification to the specific technology used by the company. In this paper, we abstract from the technological side and consider only the conceptual and external process specifications. Existing research addresses the mapping between external and conceptual process specifications but does not address the support of control from the service consumer\(^\text{15,27,68}\).

The agreement to offer certain visibility over the process execution has to be part of the contract stipulating the terms and goals of the business relation. Electronic contracts (e-contracts) contain the terms and conditions of the collaboration agreed by the parties in
a digital, machine-interpretable format\(^5\). They specify the external process agreed to be visible to the counter-parties and to be subject to monitoring (discussed next).

During e-contract enactment, the service provider supplies information to the service consumer as agreed in the e-contract on a push or pull fashion about the states of the external activities (called messaging and polling in Ref. 40). The exchange of process status information is known as process monitoring\(^8,31,52,66\). Data obtained through process monitoring is interpreted and used by the consumer to request changes in the behavior of the provider (e.g., in the case of contract breaches) or to adapt its own behavior\(^20\).

2.2. Solution approach

In our work, we call an activity at the external level a Visibility Point (VP). We call Visibility Points at which the consumer may exert control “Interference Points” (IPs). The control primitives that are offered to the service consumer at an IP are called “Interference options” (I-options). A request from the service consumer for the exertion of an I-option is called an I-request. Obviously, a process consumer may wish to invoke controls not only over single activities but over a complete process (or a subprocess) as well. This allows for a more coarse-grained control over the provider service. We call these control primitives “Process I-options” (PI-options). A request from the service consumer for the exertion of a PI-option is called a PI-request.

![Diagram of External and Conceptual Levels with Interference Options]

**Fig. 1.** Abstract example scenario

As it can be concluded from the discussion in Section 2.1, control over the process execution of the service provider can be offered to a consumer only on the external process level. In Fig. 1, we depict the conceptual (private) and external (public) process specification of a process offered by a service provider. To illustrate our approach, we use an abstract example (with an unnamed process and activities). The conceptual and external process specifications are modeled in UML activity diagrams. We use colored
filling to denote the activities from the conceptual process specification (non-colored activities are the activities from the external process specification). The I-options are presented as text on top of the activity that can be interfered with. The dashed lines between activities from the two process specifications represent the mapping of external activities to conceptual activities. In this example, the service provider offers two I-options, i.e., START and PAUSE in the third VP on the external level. The control offered at the external level has to be addressed in the conceptual process specification (depicted in Fig. 1 with a grey rectangle) guaranteeing the execution support for it.

For the support of cross-organizational process control, the possibilities for I- and PI-options that can be offered on the external level have to be defined and a specification for their support at the conceptual level has to be provided. In this paper, we provide a solution for these two issues.

3. Specification and Support of Activity Controls

In this section, we discuss the I-options that a process provider may offer to its consumers. First, we specify the I-options. Next, we discuss their support at the conceptual level.

3.1. Specification of I-options

The process of selecting our set of I-options was not straightforward. We have used as input numerous sources from three domains. As a starting point, we have studied which cross-organizational control primitives have already been considered in literature. Next, we have studied the control primitives used in contemporary workflow systems and then the control primitives suggested in work on process flexibility. These control primitives served as an inspiration in the definition of our set of I-options. We have decomposed complex primitives into their “atomic” elements. We have reasoned on their applicability for the context of cross-organizational collaborations and eliminated those that did not fit this context. We have investigated nuances (or even differences) in the semantics of control primitives with common names, discovered commonalities of primitives having different names, and re-labeled names of primitives to fit our context. We skip the description of the process of I-options selection and definition in this section in order to avoid an unstructured and lengthy presentation of our steps. Instead, we directly present the set of I-options resulting from this process in Table 1. We provide details on the control primitives from existing publications and their mapping to our set of I-options in Section 7.

In Table 1, we classify the I-options into three groups, i.e., I-options that are applicable before the execution of an activity (group 1), I-options that are applicable during the execution of an activity (group 2), and I-options that are applicable after the execution of an activity (group 3). Each I-option is parameterized upon invocation. A parameter common for all I-options is the activity to which the I-option is applied. Other parameters may be provided, e.g., duration (for DELAY and PAUSE). Although, completeness of this list is not proven, the approach of using, extending, and adapting
directly and indirectly related research results indicates its comprehensive nature (we revisit the completeness of our set at the end of this section).

<table>
<thead>
<tr>
<th>I-option</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Group 1</strong></td>
<td></td>
</tr>
<tr>
<td>START</td>
<td>The execution of an activity is started.</td>
</tr>
<tr>
<td>DELAY/PROCEED</td>
<td>Starting execution of an activity is delayed/continued.</td>
</tr>
<tr>
<td>SKIP</td>
<td>The execution of a non-started activity is skipped.</td>
</tr>
<tr>
<td><strong>Group 2</strong></td>
<td></td>
</tr>
<tr>
<td>PAUSE/CONTINUE</td>
<td>The execution of a started activity is paused/resumed.</td>
</tr>
<tr>
<td>CANCEL</td>
<td>The execution of a started activity is stopped and the activity is considered ended. Partial results from the execution of the activity remain.</td>
</tr>
<tr>
<td>PART-RESET</td>
<td>The execution of a started activity is stopped and the activity is put back in its ready state without undoing any of the work that has been performed.</td>
</tr>
<tr>
<td>PART-UNDO</td>
<td>The execution of a started activity is stopped, what has been done is undone, and the activity is put back in its ready state.</td>
</tr>
<tr>
<td><strong>Group 3</strong></td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td>An activity that has ended is put back in its ready state. Results from previous execution are not undone.</td>
</tr>
<tr>
<td>UNDO</td>
<td>An activity that has ended is put back in its ready state, after the results from the previous execution are undone.</td>
</tr>
</tbody>
</table>

Based on the basic I-options defined in Table 1, complex I-options (combinations of several basic I-options) can be defined. The complete list of complex I-options identified through an analysis of the possible combinations is presented in Ref. 6. Here, in Table 2, we list the complex I-options that may occur on a more common basis and omit those complex I-options that are, although valid combinations of I-options, less intuitive and hence less likely to be used in practice. For each complex I-option, we list the constituent basic I-options and the sequence of their invocation (denoted with the "+" sign).

<table>
<thead>
<tr>
<th>Complex I-options</th>
<th>Constituent I-options</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSTPONE</td>
<td>DELAY+PROCEED</td>
<td>The execution of an activity is postponed.</td>
</tr>
<tr>
<td>RESTART</td>
<td>PART-RESET+START</td>
<td>A started activity is stopped and is started from the beginning.</td>
</tr>
<tr>
<td>PART-REDO</td>
<td>PART-UNDO+START</td>
<td>A started activity is stopped, undone, and started again.</td>
</tr>
<tr>
<td>TERMINATE</td>
<td>PART-UNDO+SKIP</td>
<td>A started activity is stopped and undone. The control flow is passed to the next activity.</td>
</tr>
<tr>
<td>RETRY</td>
<td>RESET+START</td>
<td>An ended activity is started from the beginning.</td>
</tr>
<tr>
<td>REDO</td>
<td>UNDO+START</td>
<td>An ended activity is undone and started again.</td>
</tr>
</tbody>
</table>
The execution of an I-request leads to a change in the state of an IP. In Fig. 2, we use UML state diagram to present the state model of an IP and the I-options that trigger the state changes. The model serves three purposes. First, we use it to clarify the I-options and illustrate their impact on the activity states. Second, we use it for the definition of the requirements on the support of the I-options at the conceptual level (see Section 3.2). Third, the model serves as a main tool for the definition and system support of I-options (see Section 5.2).

Fig. 2. The state model for an interference point. States depicted in grey reflect transition states during which internal activities needed to satisfy the I-request take place.

A state change of an IP caused by an I-request should be reflected with the corresponding state change in the conceptual process (e.g., when an IP is paused, the corresponding conceptual activity(s) should be paused as well). However, the execution of an I-request at the conceptual level may require time. Thus, an activity at the external level must have states that represent these times of transition assuring consistency between the external and conceptual process levels. We call these transition states “-ING” states, e.g., PAUSING, and show them in grey in Fig. 2. The transitions that do not have an “-ING” state take place synchronously at the external and conceptual levels.

Many activity and process state models can be found in the literature (e.g., Refs. 26, 28, 48, 49, 57). These models differ from each other as they are defined to address different problem domains, different process engines, etc. Studying these state models gave us a “reverse” approach to the problem of definition of I-options. Lack of a state in our model compared to other existing models indicated lack of an I-option that can lead to this state. Clearly, not all activity states have to be addressed in the case of an interference point. For example, a “faulted” state is not relevant as the I-option needed for the transition to this state would be senseless in our context. That is why, we have investigated the activity and process states discussed in literature, compared them to our
state model, and argued on their relevance in our context. This step allowed us to verify the completeness of our set of I-options from a different perspective.

3.2. Support of the I-options at the conceptual level in sequential process structures

To support the I-options certain mechanisms at the conceptual level have to be defined. Depending on their type (group 1, group 2, or group 3), I-options have to be applied before, during, or after the execution of a VP. This means that at the conceptual level, “handlers” for I-options should be provided before the first or after the last conceptual activity or after the execution of any of the conceptual activities comprising a VP. Obviously, I-requests on a VP may arrive just before its start (e.g., if the decision for invoking an I-option is influenced by the execution of the preceding VP). Therefore, mechanisms may be required at the conceptual level preventing the automatic start of the first conceptual activity of a VP, allowing a consumer to invoke group 1 and group 3 I-options, respectively. Hence, the activity state model used at the conceptual level has to support possibilities for interaction before and after the execution of an activity. Existing activity state models (e.g., in Refs. 26, 28, 49, 57) consider different sets of events that can be handled at different points of the life-cycle of an activity. To avoid the necessity of making a choice for a specific activity state model and fixing our approach to it, we consider the most basic activity state model in which activities are isolated (i.e., they cannot be interfered with during their execution) and explicitly model the points for capturing I-requests as “wait” states. If during a "wait" state no invocation of the I-option occurs, the process proceeds its execution after a certain time-out (the time-out or a way to derive its value is specified in the e-contract). The specific way in which these “wait” states and time-out transitions are addressed at the conceptual process specifications will vary among service providers, depending on their choice of activity state model.

Having in mind these considerations, next, we discuss the handling of I-options at the conceptual level in detail. We use $x_i$ to denote the $i^{th}$ VP and $c_{i1}, \ldots, c_{in}$ to denote the first and the last conceptual activities in the set of activities mapped to $x_i$. Note that in case of parallel execution of several first (last) activities, the first (last) conceptual activity $c_{i1}$ ($c_{in}$) represents a set of concurrently executing activities. The support of complex I-options at the conceptual level is not discussed as it can be directly derived on the basis of the support defined for the basic I-options. In this subsection, for reasons of clarity, the processes used to discuss the handling of the I-options have a simple, sequential structure. The next subsection explains the handling of I-options in processes that have parallel branches.

Group 1 I-options

For each I-option from group 1, a “wait” state preceding $c_{i1}$ must be introduced to give the consumer some time to exert the I-option. This wait state represents the READY state of $x_i$. The duration of the wait state must be specified in the e-contract, i.e., it is an explicit time-out agreement stating the duration for which the I-option is available to the
If multiple I-options are offered, the time-out specification can become rather complex, e.g. different time-outs for each of the separate I-options.

**Fig. 3. Support of “START”**

START(\(x_i\)): The “wait” state that is introduced (see Fig. 3) is automatically entered when \(x_i\) is ready to be executed and gives the consumer some time to exert the START control. The “wait” state is left only when START I-request arrives or at the agreed upon time-out. Exerting the START I-option enforces the start of the activity at the time desired by the consumer. Without the explicit START I-option, the start of the activity is determined fully by the provider.

DELAY(\(x_i\)) and PROCEED(\(x_i\)): We discuss these two I-option together due to their interrelation. The first “wait” state, as shown in Fig. 4, is introduced to give time to the consumer to invoke the DELAY I-option. If the DELAY I-option is not invoked, as soon as a time-out takes place, the process execution continues with the execution of \(c_{i1}\). If the DELAY I-option is invoked, the consumer may invoke another I-option, i.e., PROCEED. Thus, a second wait state is introduced that allows the consumer to invoke the PROCEED I-option. If the consumer does not invoke it, a time-out transition would take place. After the time-out duration has expired, or a PROCEED has been received, process execution continues with the execution of \(c_{i1}\).

**Fig. 4. Support of “DELAY” / “PROCEED”**

SKIP(\(x_i\)): The invocation of this I-option requires, in addition to the wait state, the introduction of a “split” construct before \(c_{i1}\) and a “merge” construct after \(c_{i0}\) (the WCP-4 and WCP-5 patterns\(^{3}\)). During the transition period, \(x_i\) is in state SKIPPING. The transition can be direct or certain activities might have to be executed during the skipping

\(^{3}\) In Ref. 50, workflow control patterns are presented under the abbreviation WCP, followed by their number.
Enhancing Business Collaborations with Client-oriented Process Control

(Their optional character is depicted in Fig. 5 by presenting the activity with a dashed line).

Fig. 5. Support of “SKIP”

**Group 2 I-options**

In the case of Group 2 I-options, “wait” states may not be explicitly needed. The reason is that there may be sufficient time for a consumer to invoke an I-option during the execution of the conceptual activities that precede the point where the I-option is handled. However, as the preceding activities may be of short duration, we model the "wait" states explicitly for the Group 2 I-options also. In practice, for efficiency reasons, the "wait" state may be omitted in the cases of long-lasting preceding activities at the conceptual level.

Fig. 6: Support of “PAUSE” / “CONTINUE”

PAUSE\(x_i\) and CONTINUE\(x_i\): These I-options (analogously to the DELAY and PROCEED I-options) require the introduction of a combination of two “wait” states at one or more places in the conceptual process model, in which the process can be paused (see Fig. 6 showing one such possible place). The first "wait" state guarantees the possibility for the invocation of the PAUSE I-option by consumers. If it is invoked, the consumer may invoke the CONTINUE I-option as soon as he requires the activity to proceed with its execution. Thus, the second wait state is introduced to allow the consumer to invoke the CONTINUE I-option (non-invocation of PAUSE or CONTINUE, respectively, leads to a time-out transition as agreed in the e-contract). In the cases when the PAUSE is invoked before the pausing place is reached, the external
activity $x_i$ is in state PAUSING until the “pausing place” in the conceptual process is reached.

CANCEL($x_i$): The invocation of this I-option requires the implementation of a cancellation construct on the conceptual level (WCP-19). The cancellation construct can be provided at several points between $c_{i_{1}}$ ... $c_{i_{n}}$ allowing several points for internal reaction to a CANCEL (see Fig. 7 showing one cancellation construct). During a cancellation, $x_i$ is in state CANCELLING. In the transition to the “cancelled” state, one or more activities can be executed ensuring that activities that follow the cancelled activity can still be executed (indicated as a “dashed” activity in Fig. 7).

Fig. 7. Support of “CANCEL”

PART-RESET($x_i$): Similar to the cancel I-option, a “split” is necessary at the conceptual level to “implement” this I-option (see Fig. 8). The control flow after the reset is passed to $c_{i_{1}}$ (i.e., $c_{i_{1}}$ is set in the READY state). During the transition between the started and ready states, activity $x_i$ is in RESETTING state. In Fig. 8, only one possibility for a reset I-option is shown, however, multiple reset points can be specified.

Fig. 8. Support of “PART-RESET”

PART-UNDO($x_i$): Two approaches can be used to support the PART-UNDO I-option. The first solution is comparable to the handling of the RESET I-option but in addition consists of explicit conceptual level undo point(s). At those points, the undo will be handled by executing one or more activities in the undo branch, indicated by the dashed activity in Fig. 9 that will undo the work done in the activities that have already been executed, i.e., the work done will be compensated. This is a very rigid way of
Enhancing Business Collaborations with Client-oriented Process Control

handling the undo and requires pre-specifying an undo-branch for each conceptual level undo point (which becomes complex quickly in the case of choices or loops in the process).

![Diagagram of PART-UNDO](image)

Fig. 9. Support of “PART-UNDO”

The second approach to handle the PART-UNDO I-option makes use of transaction management support (see, e.g., Refs. 14, 24, 29, 34, 63). Two well-known transaction mechanisms are atomicity\(^{19}\) and compensation\(^{18}\). If the entire set of conceptual level activities \(c_{i1}, c_{i2}, \ldots, c_{in}\) is designated as being atomic, this means that either all activities (not the ones excluded through choices in the process execution) will be performed, or none at all. This is ensured by the transaction support. For the PART-UNDO I-option, this implies that the conceptual process can simply be aborted and the transaction support undoes what has been done. Compensation on the other hand, as explained above, is used to semantically undo the work that has been done. However, instead of the rigid, explicit specification of undo points and undo branches in the process specification, compensation through transaction support operates at run-time. At the time that a PART-UNDO I-option is exerted, the transaction support will analyze what has been executed, construct a new process model consisting of activities that undo the work done in the original activities, and have that compensating process model executed by the process engine. After the compensating process has been completed, control is passed back from the transaction engine to the process engine, which will discover that the conceptual process is in the ready state again (and so the external activity must be in the ready state as well). While undoing the work done, the external level activity is in the undoing state.

**Group 3 I-options**

Similar to the I-options from Group 1, for each I-option from group 3, a “wait” state after \(c_{in}\) must be introduced to give the consumer some time to exert the I-option. This wait state represents the ENDED state of \(x_i\).

**RESET**(\(x_i\)): To support the invocation of the RESET I-option, a loop construct around \(c_{i1}, c_{i2}, \ldots, c_{in}\) has to be defined (WCP-21). The loop construct is preceded by a “wait” state (see Fig. 10). Note that in essence, the RESET I-option allows for multiple iterations of an activity (loops).
3.3. **Support of the I-options at the conceptual level in parallel process structures**

In the previous subsection, the support for the I-options on the conceptual process level has been explained, illustrated with Figures 3 till 11. For clarity reasons, the processes shown in these figure were, however, simple and sequential. In cases of parallelism at the conceptual level, the I-option will be effectuated when each parallel running branch reaches a state that supports the I-option. If such a state cannot be reached, the I-option cannot be carried out. It is therefore important to distinguish in the e-contract between those I-options that are guaranteed to be available and those that are offered but might not be effectuated. In the case of guaranteed I-option execution, the provider must make sure that the required mapping is always available, at least, at the end of the conceptual level process. Because group 1 I-options are available before the start of an execution, and group 3 I-options are available at the end of an activity execution, guarantees for I-option execution are only necessary for group 2 I-options.

**Fig. 10. Support of “RESET”**

**Fig. 11. Support of “UNDO”**
An example for support of a group 2 I-option in the case of parallel branches in a conceptual process specification is shown in Fig. 12. The example focuses on the PAUSE I-option. In the example, two possible “cuts” through the parallel conceptual branches exist (a “cut” is formed by a line passing through the “wait” states of each branch). A PAUSE can be effectuated in each of the two “cuts” shown in this example.

![Diagram showing PAUSE in parallel branches](image)

Fig. 12. Support of “PAUSE” in case of parallelism

The I-options covered in this section are all associated with one external level activity. To allow a more coarse-grained process control to a service consumer, control primitives can be associated with complete external level processes (discussed in the next section).

4. Specification and Support of Process Controls

In this section, we discuss the possible process I-options (PI-options) and support for them at the conceptual level and clarify the relationship between I- and PI-options.

4.1. Specification of PI-options

An external process can be seen as an aggregation of the activities at the external level to a single, higher-level activity. Consequently, a PI-option can be viewed as an I-option defined on an activity (the process). That is why the set of PI-options includes the set of I-options that we have defined in Section 3.1. Driven by our goal to allow the definition of process controls that influence the optional execution and the order of execution of activities we have extended the work presented in Ref. 21, concerning the external, conceptual, and internal process levels. The extension allows the definition of subprocesses of external activities without providing details on the control flow between these activities, i.e. a subprocess can contain a non-complete process specification from which the control flow is omitted. So, in addition to the activity I-options, two new control primitives that operate over sets of activities can be defined i.e., CHOICE and ORDER that can be used to control which and/or in what order the activities in a subprocess need to be executed. In Table 3, we list the PI-options and explain their semantics at a process level. Note that the CANCEL, PART-RESET, PART-UNDO,
RESET, UNDO PI-options allow the user to request partial forward recovery or partial backward recovery of the process execution.

Table 3. List of the PI-options

<table>
<thead>
<tr>
<th>PI-option</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Group 1</strong></td>
<td></td>
</tr>
<tr>
<td>START</td>
<td>The execution of a process is started.</td>
</tr>
<tr>
<td>DELAY/PROCEED</td>
<td>Starting execution of a process is delayed/continued.</td>
</tr>
<tr>
<td>SKIP</td>
<td>The execution of a non-started process is skipped.</td>
</tr>
<tr>
<td>CHOICE</td>
<td>From the set of activities in the process one or more are chosen to be executed.</td>
</tr>
<tr>
<td>ORDER</td>
<td>The order of execution of (some of) the activities in the process is set.</td>
</tr>
<tr>
<td><strong>Group 2</strong></td>
<td></td>
</tr>
<tr>
<td>PAUSE/CONTINUE</td>
<td>The execution of a started process is paused/resumed.</td>
</tr>
<tr>
<td>CANCEL</td>
<td>The execution of a started process is terminated. Partial results from the execution of the process remain.</td>
</tr>
<tr>
<td>PART-RESET</td>
<td>The execution of a started process is stopped and the process is put back in its ready state without undoing any of the work that has been performed.</td>
</tr>
<tr>
<td>PART-UNDO</td>
<td>The execution of a started process is stopped, what has been done is undone, and the activity is put back in its ready state.</td>
</tr>
<tr>
<td><strong>Group 3</strong></td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td>A process that has ended is put back in its ready state. Results from previous execution are not undone.</td>
</tr>
<tr>
<td>UNDO</td>
<td>A process that has ended is put back in its ready state, after the results from the previous execution are undone.</td>
</tr>
</tbody>
</table>

A closer scrutiny reveals that the START and DELAY/PROCEED PI-options can be expressed by an I-option and the PAUSE/CONTINUE by several I-options. In the example presented in Fig. 13, the DELAY(X) PI-option can be represented as a DELAY(X₁) I-option. The PAUSE/CONTINUE PI-options can be represented as one or more I-options, as there can be multiple points in the conceptual process specification where support for a PI-option is provided. In Fig. 13, the PAUSE (X) PI-option can be represented as the PAUSE(X₂) and PAUSE(X₃) I-options.

![Fig. 13. Relationship between PI- and I-options in the case of a sequential external process](image-url)
If a PAUSE/CONTINUE PI-option can be invoked during the parallel executions of branches at the external level (see Fig. 14), this PI-option is equivalent to the simultaneous invocation of the respective I-options in each of the parallel branches. In the example in Fig. 14, invocation of PAUSE(X) is equivalent to the simultaneous invocation of PAUSE(X₂) and PAUSE(X₃) I-options.

Fig. 14. Relationship between PI- and I-options in the case of parallelism

Offering the START and DELAY/PROCEED control primitives as PI-options instead of as I-options allows the consumer to reason for its interference possibilities at a process level. Furthermore, one PI-request may substitute numerous I-requests.

A process can be composed of several subprocesses. As a subprocess is itself a process, PI-options can be defined on subprocesses as well. Note that we consider as subprocesses constructs that have a “single-entry-single-exit” point (as defined in Ref. 47 and approached in Ref. 58). The reason for this is that the definition of PI-options on a subprocess with multiple entry/exit points may lead to non-deterministic process states. For example, consider a subprocess SP that has two entry points and is started from one of them, a PI-option SKIP (SP) is invoked and the subprocess is skipped. However, the subprocess may be started again from the second entry point. In this case, the execution of the subprocess is undefined as it is not clear what behavior is expected from the provider – whether to skip again the subprocess or execute it. The problem is analogous to the problem occurring in the case of using “non-symmetrical” process constructs (AND split followed by an XOR join) where, because of inconsistent modeling, multiple execution paths over an activity following a join occur¹.

4.2. Support of the PI-options at the Conceptual Level

The PI-options that have analogous I-options (i.e., all except CHOICE and ORDER) are in fact I-options defined over a “process” activity. Thus, the requirements for their support at the conceptual level are analogous to those for their respective I-options. Next, we discuss the support required for the CHOICE and ORDER PI-options.

CHOICE(x₁,...,xₘ): The support for the CHOICE(x₁,...,xₘ) PI-option requires the implementation of an exclusive choice construct (WCP-4), a multi-choice construct (WCP-6), or a deferred choice construct (WCP-16). The choice constructs are used to
“steer” the conceptual process according to the CHOICE PI-request. In case no PI-request is issued by the consumer, the default choice is taken (which can be random choice).

ORDER(x₁,...,xₖ): The conceptual level process requires the occurrence of an interleaved parallel routing (WCP-17) or interleaved routing (WCP-40) construct to support the ORDER(x₁,...,xₖ). The first constrains a set of activities to a partial-order execution, while the second allows any execution order between a set of activities. However, both patterns are not commonly supported by process aware information systems50. A more common but less design-efficient possibility to support this PI-option is to explicitly include all possible permutations of ordering the activities within the conceptual process specifications. The permutations should be preceded by one of the choice constructs mentioned above, so that input can be given to make the proper choice for the desired ordering of activity execution to be taken.

5. Applicability of the framework

In this section, we present an example case for the application of our framework for cross-organizational process control. The case illustrates our approach and demonstrates the framework applicability.

Our case is based on a teleradiology process for the acquisition and interpretation of medical scans of patients. The process results in a report that a medical specialist, who ordered the scan, can use to base his diagnosis and treatment on. The process starts by scheduling the patient. At the scheduled time, the required scans are acquired, after which an interpretation report is created and distributed to the service client. The process ends after financing has been handled. An extensive description of the process is provided in Ref. 61.

Fig. 15. The teleradiology process
A process designer at the service provider side (e.g., a specialized radiology clinic), first, defines the conceptual process specification. After applying aggregation and customization techniques the external specification is derived. In Fig. 15, we show the teleradiology process at the external and conceptual levels. The dashed lines represent the mapping of external activities to conceptual activities, e.g., the “Elaboration” activity consists of two conceptual activities. The ovals at the external level that encompass several activities represent subprocesses.

Next, based on the company policy, the process designer defines a set of I- and PI-options for the external level and makes adaptations, if necessary, on the conceptual specification that guarantees the support of the I-options. Although the individual handlers of I-requests are of low complexity, combining them into one conceptual process specification complicates the model significantly. That is why we depict it in three separate figures (addressing the support for the I-options, process PI-options, and subprocess PI-options, respectively). As it can be seen from Fig. 16, the service provider has specified two interference points for the teleradiology service: X-Ray Scan and Elaboration. The specified I-options are RETRY for the X-Ray Scan interference point and DELAY/PROCEED for the Elaboration interference point.

The consumer receives a copy of the scan after it has been made. The RETRY I-option for the “X-Ray Scan” activity allows consumers who are not satisfied with the result of the X-ray scan acquisition to request the X-ray scan to be taken again. Note that with the RETRY the original X-Ray scan is kept, as opposed to the REDO I-option, in which case the work done (taking the X-Ray scan) is undone after which a new X-Ray scan can be acquired.

The DELAY/PROCEED I-options allow the consumer to delay (if desired) the elaboration of the report until the load at the service provider becomes low, resulting in smaller charges for the service. Of course if the load is already low (and hence price is low) by the time of the start of the activity or the case is urgent, the consumer would not exert this control. The financial consequences of exerting an I-option under the different conditions should be included in the e-contract.

Fig. 16. I-options in the teleradiology example
At the conceptual level, the service provider addresses the two I-options by offering the respective constructs discussed in Section 3. The RETRY I-option is a complex I-option that consists of a RESET directly followed by a START I-option. That is why the wait state needed for the START I-option is omitted in the conceptual process specification (for space reasons, we represent the wait states as circles).

The service provider has defined two PI-options on the complete process (see Fig. 17). A PART-RESET PI-option is offered for invocations when the scheduling is not satisfactory (e.g., the dates scheduled for the scans are too late for the needs of the consumer). Invocation of this PI-option would enable starting the process all over. A CANCEL PI-option is offered to terminate the entire process if the customer believes that no acceptable scheduling can be achieved or if the consumer knows that reporting is not required anymore and the process should be terminated. Of course, a suitable compensation (usually financial) may be requested, as some work has been performed. The nature of the compensation is specified in the e-contract. The e-contract provides specifications for the consequences (financial and/or otherwise) from the invocation of the allowed (P)I-options. In our example, the invocation of the CANCEL I-option leads to the execution of a financial process during which the amount of financial compensation required is determined (depending on the amount of work already performed) and a bill is sent out.

As can be seen in Fig. 17, the PI-options are supported by the corresponding workflow control patterns at the conceptual level. Because of the location of these patterns in the conceptual process, the PART-RESET can only be performed after the “Schedule” activity and before the start of the “prepare patient” activity (Schedule and Scan Acquisition external level activities, respectively). Also, the CANCEL can only be performed at that same location and additionally after the scans have been acquired and the reporting has not started yet. The PART-RESET and CANCEL PI-options are...
therefore not guaranteed to happen, which should be agreed upon in the e-contract, as explained at the end of Section 3.2. Including the pattern(s) at more locations in the process, allows the PI-option to be performed at those locations as well, and thus offers a more flexible process control to the service consumer.

As illustrated in Fig. 18, the service provider has defined the ORDER and RESET PI-options on the Scan acquisition subprocess (consisting of the CT Scan, MRI Scan, and X-Ray Scan activities) and the SKIP PI-option on the Reporting subprocess (consisting of the Elaboration and Distribution external activities).

Using the ORDER PI-option, the consumer can state the order of execution of the CT Scan, MRI scan, and X-Ray Scan activities in the subprocess. The conceptual process allows for the three activities to be performed in any order defined with the I-request. The RESET PI-option allows the consumer to perform all scanning activities again (e.g., if there was a change in the patient’s condition during the scanning). The SKIP PI-option allows the consumer to skip the reporting process in case the report is not needed any more.

The complete process specification for this case can be derived by “unifying” the three process specifications presented in Fig. 16, 17, and 18 (not shown due to its overall complexity and size). As the individual handlers of the I-options are defined over block constructs, unification is straightforward. Upon unification, multiple, sequential “wait” states needed for capturing different I-options occur. These are grouped into a single “wait” state that is used to capture all the corresponding I-options.

6. The PROXE system

As a proof of concept, a prototype implementation has been realized, called the PROXE system (PROcesses in crossorganizational environments). In this section, we first present the conceptual architecture of the PROXE system. Next, the implementation architecture and the working of the system are explained.
6.1. Conceptual Architecture

The PROXE system is based on our previous work on business process web services (BPWS), as presented in Ref. 22. Web Services are the de facto standard for service collaborations. However, they are black box offerings. BPWS is an extension to standard web services, with which they are “opened up” so that their internal business process structure is exposed. The exposed business process structure coincides with the external level process specification as presented in Section 2. Each activity in the external level process is implemented as another BPWS, which can either be a regular BPWS as mentioned above, i.e., its process again being performed by another party, or it can be a leaf-BPWS, in which the conceptual level process is executed. Thus, a BPWS can be seen as a tree, in which each non-leaf node is a regular BPWS and each leaf node is a leaf-BPWS.

BPWS defines five interfaces (port types in web service terminology) facilitating collaborations for which the external level process is necessarily exposed to the service consumer (as opposed to black box collaborations). Through these interfaces, a service can be invoked, monitored, controlled, and synchronized. It is also possible to retrieve the external level process specification or e-contract specification. Our architecture does not include the synchronization interface as it is not relevant for controlling service executions.

In Fig. 19, we show the conceptual architecture of the PROXE system. Central to the architecture is the BPWS component. It is called by one or more service consumers and calls one or more service providers (a leaf BPWS does not involve service providers). Calls are made through the predefined interfaces and each call is received and handled by the component dedicated for that interface. The BPWS components are:

The process engine is used to perform the service. It instantiates the external level process as agreed upon in the e-contract (or the conceptual level process in case of a leaf-BPWS) and takes care of the process execution progress.

The contract checker contains the logic to verify an interface call against the client’s e-contract. An interface call contains the identification of the requestor as well as the request itself. This component uses that information for authentication, authorization and checking whether the request is allowed according to the specifications in the e-contract with the requestor (cf. WS-Security, WS-Policy, WS-PolicyAttachment in the Web Services domain). One client e-contract exists for each of the service consumers of the BPWS.

A service is activated/invoked through the ACT interface and handled by the invoker. After verifying the call through the contract checker the process engine is instructed to start the process. The information correlating the service consumer to the started process instance is subsequently stored in a database and the process result is passed back to the service consumer when the process ends.

The monitor component receives calls through the MON interface. It retrieves monitoring information on the service being executed. Depending on the required information, this can be retrieved directly from the process engine, by making a call to
the MON interface of the service provider(s), or through a combination of the two. The collected information is mapped (using the client e-contract and the internal process specification) into the terminology that has been agreed upon in the e-contract (so that it is interpretable by the service consumer). The monitoring possibilities that are available through the MON interface of the provider(s) are specified in the provider e-contract(s).

![Conceptual BPWS architecture](image)

The **controller** component, called through the CTRL interface, is used to process incoming (P)I-options. The controller determines, using the internal process specification how the control request can be effectuated. Similar to the monitor component, a control request could be handled by the process engine, calling the service provider(s), or a combination thereof. The provider e-contract(s) provide the controller component with the available (P)I-options for controlling the services being executed by the service provider(s). Monitoring information is used to determine whether or not the (P)I-option received is still possible, as execution might have already progressed beyond the point in the process for which the (P)I-option makes sense.

Even though the service consumer usually has its own copy of the e-contract made with the BPWS provider, it can be requested from the BPWS provider through the **specification retriever** component. It can also provide specific parts contained of the e-contract, e.g. the process specification (external level).

Because a leaf BPWS represents a conceptual level process, it is not concerned with contractual issues. Its conceptual architecture is the same as that of a regular BPWS, minus the contract checker, and client and provider e-contracts. Also, the monitor and controller are simpler, as they only have to make calls to the process engine.
6.2. Implementation

The PROXE system is implemented in Java and is built on top of a number of freely available software tools. The GlassFish Web Application Server (WAS)\textsuperscript{54} is used to deploy the business process web services. Execution of the external level process is performed by the OpenESB BPEL engine\textsuperscript{55}, which is an extension to the GlassFish WAS. The conceptual level processes are specified in YAWL (and so they are transformed into internal level processes, containing technical details related to YAWL) and are executed on the YAWL workflow system\textsuperscript{55}. As a validation case, we have used the teleradiology scenario, introduced in Section 5. Thus, the original process models are complemented with the proper workflow patterns, as shown in Section 5, so that the I-options can be effectuated during process execution.

The implementation architecture of the system is presented in Fig. 20. The bold rounded rectangles are (leaf) BPWSs and the bold rectangles are auxiliary Web services (auxiliary services are not part of the business process itself and are therefore implemented as regular Web services, and not as BPWSs). For practicality reasons the process engines have been detached from the BPWS. Although, conceptually, each process runs on its own engine, process engines are designed to execute multiple separate processes, so it is inefficient to use a separate process engine for each process. For the same reason, one single database has been used and the leaf-BPWSs share one WAS. Wrappers have been used to encapsulate the process engines and database, so that the specific process engines or database can be easily replaced with others (e.g., IBM’s WebSphere products, Apache Tomcat, Apache ODE, Intalio’s BPMS, Microsoft’s SQLServer, Oracle’s DBMS, BPEL Process Manager).

![Fig. 20. Implementation architecture of the PROXE system](image-url)

The teleradiology BPWS is run on the OpenESB BPEL engine, which invokes the other BPWSs (representing the external level activities). The internal processes are executed on the YAWL process engine. Process state information (and result) is stored in the database (via the YAWL observer and the YAWL Monitor WS). Upon completion,
the process results are retrieved by the leaf BPWS, through the YAWL Monitor WS, and returned to its caller, i.e., the BPEL engine. The BPELDB WS stores correlation information relating the service consumer and the leaf BPWS process instances to the BPEL process instance.

A collection of screenshots, illustrating the system running the teleradiology scenario, is shown in Fig. 21. Fig. 21a shows part of the teleradiology BPEL process specification (designed in NetBeans\(^{39}\)). Note that BPEL allows synchronization flows between parallel branches within the “<flow>” construct, making BPEL a not fully block structured language. As we consider only block structured languages (to support the PI-options), we disallow the synchronization flows of BPEL. Fig. 21c presents the Scan Acquisition process specification in YAWL. It has been extended with one wait activity at the end of the process to allow for the exertion of the RESET I-option. The control flow leading from the wait activity to the start of the process is part of the pattern used to support the RESET I-option, as exemplified in Fig. 10. A variable is set (by the controller component) that ensures that the YAWL engine chooses that control flow in case the RESET is requested.

Fig. 21: Collection of screenshots of PROXE
In Fig. 21b, the rudimentary user interface is shown that allows to invoke, monitor, and control the scan interpretation service and to retrieve the specification and/or e-contract of that service. This user interface is automatically generated and is useful for testing purposes. Typically, the invoke interface is called from within a service consumer process, while the other interfaces are called by a management tool employed by the service consumer, i.e., the service offered by the provider is integrated into the information systems of the service consumer. The screenshot shown in Fig. 21d illustrates a view on the database (MySQL is used), showing part of the table that contains the correlation information between the BPEL process instance and the YAWL process instances.

Currently, the PROXE system supports all of the identified (P)I-options, except for the CHOICE and ORDER PI-options. The process decomposition on the external level required for these two PI-options is not directly supported by BPEL, and so the workaround involves more complicated monitor and controller components, that will have to have more process specification knowledge (the decomposition) and also maintain the process administration associated with the execution of the activities in the process decomposition. The additional implementation for these two PI-options is scheduled for future work. Also, support for the UNDO (P)I-options is currently provided through explicitly specifying the undo branches, as illustrated in Fig. 9 and Fig. 11. A generic support of the undo I-option (and the complex I-options that use it) requires the integration of the YAWL system with an advanced transaction model, e.g., the WIDE transaction model or an adapted version of the X-transaction model, that supports simultaneous undo on the external and internal level. This is future work as well.

A consumer may invoke an I-option whenever it is available. Several, mutually exclusive I-options can be available for the same activity (e.g., SKIP(x_i) and DELAY(x_i)). Different approaches to handle multiple, potentially inconsistent I-requests are possible: allowing multiple invocations and checking them for consistency (queuing of invocations), accepting one invocation and ignoring subsequent invocations, etc. The system providing control to the consumer should handle this. Our current approach in PROXE to cope with this problem is to allow only sequentially and synchronously invoked I-request. That is, after an I-request, a response should be received before issuing another I-request. This enables checking consistency of the I-requests in a simple manner. We suggest using the state model for the visibility points, given in Fig. 2, for consistency checks. The state model defines the possible (P)I-options that can be offered by the provider for each possible state of a visibility point. At any state, only the options allowed in that state from the complete set of (P)I-options initially defined by the provider can be offered to the consumer. Sequential execution of I-requests enables transitions in the diagram to occur deterministically. More advanced handling of multiple, concurrent (P)I-request will be analyzed in future research.
7. Related Work

As already discussed in Section 3.1, we have used results from publications from a number of research domains as an inspiration in the design of our set of I- and PI-options. We show the relation between the control primitives defined in these efforts and our set of I(PI)-options in Table 4. The notation used in the table is as follows: an “X” indicates that the semantics of our (P)I-options is the same as the primitives used in these publications (although the name could be different), a “~” indicates that a semantic difference exists, or that the semantics of the primitive is not clear from the publication. An empty cell in the matrix indicates that no primitive was given in the publication that matches the (P)I-option. Next, we explain in detail their relation to our work.

<table>
<thead>
<tr>
<th>Publication source</th>
<th>START</th>
<th>DELAY/PROCEED</th>
<th>SKIP</th>
<th>PAUSE/CONTINUE</th>
<th>CANCEL</th>
<th>PART-RESET</th>
<th>PART-UNDO</th>
<th>RESET</th>
<th>UNDO</th>
<th>CHOICE</th>
<th>ORDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ref. 20 (Grefen, et al)</td>
<td></td>
<td>X</td>
<td></td>
<td>~</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ref. 25 (Hofstede, et al)</td>
<td>X</td>
<td>X</td>
<td>~</td>
<td>X</td>
<td>~</td>
<td>~</td>
<td></td>
<td></td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Ref. 28 (IBM)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ref. 32 (Khalaf, et al)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ref. 36 (Microsoft)</td>
<td>X</td>
<td>~</td>
<td>X</td>
<td>X</td>
<td>~</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ref. 38 (Mulyar)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>Ref. 48 (Russell, et al)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>~</td>
<td>X</td>
<td>~</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ref. 51 (Schonenberg, et al)</td>
<td>~</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ref. 64 (Weber, et al)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7.1. Cross-organizational process flexibility

Research into collaborative business process executions started a number of years ago and has led to a number of approaches and directions \(^{2,3,10,20,45,46}\). Monitoring process executions performed at other parties was deemed an important aspect to consider from the start \(^{8,31,52,66}\). However, research into controlling the process executions, on the basis of the information retrieved through the monitoring, has been lagging behind. Control over an outsourced process, performed by a service provider, was initially explored in the CrossFlow project \(^{20}\). The CrossFlow approach was ad hoc and a limited set of controls was devised, i.e., “stop”, “continue”, “abort”, “rollback”, “change variable”. We have used all of these controls as input for our set of I-options except the “change variable” control as it is not a process control primitive. The “stop” and “continue” primitives are
covered by PAUSE/CONTINUE. The “abort” primitive resembles the CANCEL, but does not define whether the process is rolled back. The “rollback” matches the UNDO and PART-UNDO (using the second approach described in Section 3.2). In Ref. 20, mapping the process control primitives from the contract-level process to the internal process was determined by the possibilities of the underlying WfMS.

Process control in the Web Services area typically concerns only the start of services. The necessity for control in Web Services is mentioned in Ref. 22 but no concrete control options or solution are presented.

Process orchestration, e.g. BPEL, adds the possibility for exception handling (cancellations, compensations and/or other fault handling) either natively or through the use of additional Web Service specifications, e.g., WS-transactions.

E-contracting research deals with the process of establishment and enactment of electronic contracts and determining their content to support digital collaborations between parties. The work presented in this paper complements the research in e-contracting by extending e-contracts with the allowed I- and PI-options.

7.2. Workflow process flexibility

Control over intra-organizational processes process executions is addressed in the work on process flexibility. In Ref. 51, a taxonomy of four classes of flexibility is presented, i.e., flexibility by design, flexibility by deviation, flexibility by underspecification, and flexibility by change. The flexibility by underspecification and flexibility by change are not relevant in the context of collaborative business processes executions, as they are targeted at changing the process specification. The reason to have a party perform a process on another parties’ behalf, is because that party has the relevant expertise to perform the service in a “better” way. Therefore, it does not make sense to have a less experienced party change the service specification. Moreover, business relations are pre-defined and any change in a process agreed in the e-contract would require e-contract re-negotiation. The flexibility by design class provides ways to order activities and make choices between activities (by means of control flow constructs such as parallelism, choice, iteration, interleaving, multiple instances, cancellation), while the flexibility by deviation class provides operations such as “redo”, “undo”, “skip”, “invoke”. In our work, we have considered the flexibility by design and flexibility by deviation control primitives discussed in Ref. 38 and Ref. 51 and adapted them for the collaborative settings. For example, we have combined the interleaving and parallelism control flow constructs in the ORDER I-option as from a user perspective, the distinction is not important. As discussed in Ref. 38, current standard workflow management technology can directly support only a small subset of the functionality covered by our control primitives. The approach discussed in this paper guarantees support of the full set of control primitives in the general setting, independently of the characteristics of workflow systems used internally.
In Ref. 64, an overview of patterns on flexibility in process aware information systems is given. This work proposes “adaptation patterns” and “patterns for changes to predefined regions”. Adaptation patterns are aimed at flexibility in the cases of unforeseen exceptions and require structural process changes. Hence, analogously to the flexibility by change discussed in Ref. 51, we concluded that they are not relevant for our context where the set of possible process changes has to be pre-defined and agreed by the parties. Patterns for changes to predefined regions discuss four patterns of which we found “late selection of process fragments” and “late composition of process fragments” useful for the context of cross-organizational process control. These two patterns inspired us to introduce the CHOICE and ORDER PI-options.

In Ref. 48, a set of workflow exception patterns is proposed and a language for modeling the handling of exceptions is proposed. Exceptions are defined as deviations from the normal execution of a business process. Our control primitives offer increased flexibility to the client which may allow handling of envisioned exceptional situations that are agreed in the e-contract. However, our approach does not address only handling of process deviations but provides a way for improving the effectiveness of business collaboration by handling clients’ preferences on the process execution. Furthermore, the work on exception patterns is strictly geared towards the workflow domain and the control primitives introduced reflect these specifics. For example, attention to controlling single, current and all cases of a workflow specification and allocation of work items is explicitly paid attention. Consequently the control options proposed in Ref. 48 overlap with only part of our set of I-options. More precisely, the “remove”, “suspend”, “continue”, “restart”, “rollback”, “compensate” exception handling primitives can be mapped to our set of I-options, while primitives such as “reallocate work item”, and “reoffer work item” have no equivalent in our set due to their workflow specific nature.

The work presented in Ref. 32 describes an event-notification approach for the extension of BPEL processes with monitoring and control possibilities. The authors focus on the architecture that is required to support their approach and the specific technical aspects of the problem in the context of BPEL. The types of events are not defined and only an “invoke” primitive is used as an example control option.

7.3. Process control in process engines

Next to investigating the literature on cross-organizational process control and on process flexibility, we have examined the process control possibilities offered by a number of existing process engines (or workflow systems)\cite{25,28,36}.

Process control possibilities in YAWL\cite{25} are addressed in three parts of the YAWL documentation, i.e., in the YAWL "work item state transition chart", the resource allocation mechanism (called privileges in YAWL), and the YAWL process specification designer tool. The “start”, “un/suspend”, and “cancel” transitions in the state transition diagram match with the transitions triggered by the START, DELAY/PROCEED, PAUSE/CONTINUE, and CANCEL (P)I-options. From the privileges mechanism, we have inferred that YAWL has support for the SKIP through the “skip” privilege (although
it is not specified if this is possible after the task has already partly executed and cancelled/reset) and for the PART-RESET through the “reallocate stateless” privilege (which in YAWL requires the task to be allocated to another resource). The “cancellation region” construct in the process designer tool indirectly indicates support for the UNDO and PART-UNDO I-options.

A number of transitions from the state transition diagrams used in the WebSphere Process Server can be matched to the transitions triggered by our (P)I-options (note that transitions are unnamed in the WebSphere documentation). The transition between the persistent state “inactive” and the transient state “running” indicates that a transition equivalent to the one triggered by our START I-option exists. The existence of the persistent state “skipped” implies the existence of a skip transition, matching the transition triggered by our SKIP primitive. The transition to the persistent state “terminated” is comparable to the transition triggered by our CANCEL primitive. However, it is not stated explicitly in the WebSphere documentation what happens to the work already performed in the task when it transitions into the terminated state. The transitions triggered by PAUSE/CONTINUE are equivalent to the transitions that lead to the “waiting” state in WebSphere.

The activity state model defined in the Windows Workflow Foundation (WF) contains six states: one begin state (“Initialized”), four transient states (“Executing”, “Cancelling”, “Faulting”, and “Compensating”), and one end state (“Closed”). In our state transition diagram (see Fig. 2), we do not distinguish between successful or unsuccessful termination of activities. Hence, the “Faulting” state is not explicitly covered in our model, but can be considered a sub-state of our “Ended” state. Transitions from “Initialized” to “Executing”, from “Executing” to “Cancelling”, and from “Closed” to “Compensating” are equivalent to the START, CANCEL, and UNDO (P)I-options, respectively. Note, that the PART-UNDO is not supported, as compensation can only be reached after the activity is closed (ended in our terminology). Even though it is not explicitly contained in the WF activity state model, “An activity may only close when all children are either in the Closed or Initialized states.” This implies that, if a child activity is in the Initialized state, and can be closed, such activity can be skipped. It thereby relates to our “SKIP” (P)I-option, although it only holds for child activities. Analysis of the different components in the “ComponentModel” and “Activities” reveals a support for suspending and resuming workflows (through the “DelayActivity” and “SuspendActivity” components), which correspond to the PAUSE/CONTINUE (P)I-option (the “DelayActivity” is used in the suspension for a specific amount of time, using a timeout mechanism).

7.4. Web services as a supporting technology

Collaboration types range from simple service outsourcing to forming complex, dynamic business networks. Web services are a de facto standard to offer services to parties. Web services, however, are black-box services that do not expose the process contained within. In Ref. 22, an extension is presented, which “opens up” Web services through
different interfaces, one of which is the control interface to allow for process control to the service consumer. However, details on how to effectuate this control are not given. In Ref. 30, an approach is proposed to the dynamic addition and selection of services during process execution. The approach aims at improving the service flexibility. Its application in combination with our approach will allow dynamic process control customization. Different web services could be invoked depending on the agreements on process control defined in the e-contracts (e.g., if an UNDO I-option is selected, a web service with transactional support will be invoked).

7.5. Process flexibility in management science

Controlling business processes, from the perspective of business management science, is concerned with controlling the outsourcing itself (contract breaches, partner trust, etc.), which is more relevant for long-term collaborations as opposed to the outsourced execution. Our context are virtual enterprises, in which collaborations are setup dynamically with the partners that are most suitable at that time.

8. Conclusions and Future Work

Currently, there is a technological gap in the support of cross-organizational collaborations. While invocation and monitoring of cross-organizational processes by process consumers is addressed, the need to react on the monitoring information is not adequately considered. Our framework for the support of cross-organizational process control fills this gap. In this paper, we define a set of interference options, called I- and PI-options, that can be made available to a consumer organization to exert control over the process performed by a provider organization in a business relationship. By offering interference options, a provider organization has an additional mechanism to distinguish its services from those of other service providers. For a service consumer, interference requests offer an increased flexibility in service executions. To support these I- and PI-options internally, we present a mapping from the external process to the private conceptual process of the provider. The approach is illustrated with a real-life scenario from the healthcare domain. A proof-of-concept system called PROXE has been implemented.

In this paper, we concentrate on the system support of cross-organizational process control at the service provider side, as it is critical for the realization of this innovative cross-organizational paradigm. Based on the information obtained through process monitoring, the consumer is allowed to invoke I- and PI-options. While manual invocation is possible, the true potential of the paradigm is achieved when an information system for process control is used at the consumer side as well. Cooperation between the information systems of the consumer and provider during business relations can deliver the extreme degrees of efficiency and effectiveness to a service consumer required in modern business relations.

The focus of the work presented here is on the identification of the I- and PI-options and on the operational support for them, which is a basic requirement for future work in
this area. The design and implications of (P)I-options in elaborate collaborative settings, where reciprocal outsourcing can take place, governed by contracts that are consequently dependent on each other has to be investigated, for example through extending the work presented in Refs. 9, 12, 13. Besides the outsourcing paradigm, other ways of business collaborations exist, for example process choreographies and orchestrations. The possibilities for (P)I-options in those settings, and corresponding language developments, for example “Let’s Dance”\textsuperscript{67}, WS-CDL\textsuperscript{62}, or BPMN 2.0\textsuperscript{43} is considered part of the future work in this area, especially with respect to the visibility or existence of an overall process specifications in relation to the constituting local process specification (the public views of the participants).

As indicated throughout the paper, a number of issues related to the run-time side of the cross-organizational process control require further research. The approaches for the handling of I- and PI-requests in a consistent manner have to be investigated and defined. Synchronization between provider and consumer systems for the availability of I- and PI-options at each point in time requires attention as well. Techniques, mechanisms, and algorithms to support validation of the process design that includes cross-organizational process control possibilities have to be developed. In parallel to this future research agenda, we plan to extend the PROXE system. The extensions will serve as a test bed for our future research findings. A comprehensive monitoring and e-contracting support will be included in the PROXE system, complementing it into a full-blown solution for cross-organizational process provision.

Acknowledgements

We thank Xiaohui Zhao for his valuable comments and recommendations on the drafts of this paper.

This research is supported in part by the Natural Sciences and Engineering Research Council of Canada Discovery Grant 3182.

References

34. S. Angelov, J. Vonk, K. Vidyasankar, P. Grefen

44. Oracle Corporation, MySQL Database Management System (2010).
Enhancing Business Collaborations with Client-oriented Process Control


54. Sun Microsystems, GlassFish Web Application Server v2.1 (2010).

55. Sun Microsystems, OpenESB: The Open Enterprise Service Bus (2010).


