# Semantic Interoperability in Interorganizational Workflows *

Fabio Casati
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza L. Da Vinci, 32 — I-20133 Milano, Italy
Phone +39-02-2399.3408, Fax +39-02-2399.3411
E-mail: casati@elet.polimi.it

**Abstract**

Most workflow model and systems only support the separate and independent execution of business processes. However, processes often need to interact with each other, by synchronizing the execution of their activities and by exchanging process relevant data; in particular, activities which are performed in the context of a process may need to influence activities of a different process in a way that, laying "between" the two processes, is quite difficult to design, formalize, and understand. These problems particularly occur when processes are executed in different organizations, and are managed by different systems.

In this paper we propose a workflow model that allows for a flexible semantic workflow interoperability, enabling the cooperation among interorganizational workflows; we extend traditional workflow conceptual models by allowing workflows to publish and subscribe to events, and by enabling the definition of nodes in the process execution where events should be sent or received, along with event parameters that provide for data exchange between workflows. We then present how events can be dispatched in a multi-organizational context, and finally we discuss some properties related to the analysis of interacting workflows.

## 1 Introduction and Motivations

Current WfMSs are unable to effectively support cooperation between processes, particularly when these are executed in different organizations. The only typically allowed form of interaction is the instantiation of a (sub-) workflow from within another (parent) workflow, and the collection of the results at the end of the sub-workflow instance.

While in some cases interacting processes may be unified within a single workflow, sometimes this approach is not feasible or not satisfactory, as in the case of interorganizational processes, where workflows are defined with different models, execute on top of different systems, are managed according to different business policies, and aim at different business objectives. Furthermore, the semantics of the processes may require that workflows remain separate, not to introduce constraints which are absent in the business processes they model. As an example, in the following we introduce two processes involving the management of pacemakers and patients: while the two workflows (managed by different organizations) do need to interact, they must remain separate, since a patient may change several pacemakers and a pacemaker may be implanted in several patients, thereby changing their "partner" patient process.

The problem of interoperability between interorganizational processes has many facets, ranging from technological issues about how to integrate different WfMSs of different vendors on different platforms, to security and authentication issues, and to the purely conceptual issues of specifying how the interaction should occur. In this section, we focus on *semantic interoperability*: we define a workflow model that enables interoperability among workflows and that can be easily adopted and integrated within different systems. The core of the extended model is the capability of workflows to publish and subscribe to events, that may be used in order to synchronize execution of activities and to exchange data between workflows. Then, we show how events are dispatched in a multi-organizational context.

*Some of the concepts presented in this paper partly overlaps with a presently submitted conference paper

Event dispatching services are becoming very popular, and a number of models, systems, and architectures have been proposed as a natural mechanisms for achieving interoperability between software applications (see, e.g., [1, 6, 9, 10]). This approach seems promising for workflow management as well, and we discuss how it can be integrated within workflow models and how it can be used to support workflow interoperability.

Our approach is inspired from the many research papers and systems dealing with event dispatching and event processing. They show that a clear trend towards event publish/subscribe models is occurring for supporting semantic interoperability between different applications (e.g., [1, 5, 10]), We adopted the basic philosophy of this model, with some extensions in order to make it applicable in the context of workflow management. The works of [7, 16] have also been considered as input, and have been extended to enable a flexible form of interoperability and of event dispatching among different organizations. In fact, the workflow domain has different requirements with respect to "traditional" event dispatching mechanisms, and additional issues have to be considered in an interorganizational context. In particular, workflows need to specify when in the process they send or receive an event, which are the event parameters, and possibly which is the source or destination workflow. Furthermore, when few events are produced, workflows and organizations compete for consuming events while, when few consumers are available, events (and their producer workflow or organization) compete for being delivered (served) first.

Starting from the above considerations, we extended our preliminary work on workflow interoperability presented in [4], by allowing the explicit definition of points in the workflow when events are sent or requested, along with event parameters. Furthermore, we have designed an event dispatching mechanism that delivers intra-organization events on the basis of events and request priorities, and delivers inter-organization events on the basis of busines agreements among organizations, thereby defining a framework which is practically applicable in a multi-organizational context.

We foresee that the definition of methods, model, and mechanisms for semantic workflow interoperability will have increasing importance in the future, due to the ever-growing need for inter-organizational, cooperating workflows, to the evolution of middleware platforms, and to the effort of standardizing committees such as the Workflow Management Coalition (WfMC) [8, 14, 15], the IETF SWAP working group [13], and the OMG [11, 12], that makes interoperability possible at the technical level.

## 2 Patients and pacemakers

In the following we present two examples of tightly integrated workflows. We describe a patient management system, illustrating the sequence of activities performed by patients whose heart disease is cared by means of pacemakers, and a pacemaker management system, illustrating the process of acquiring, charging, implanting, maintaining, and replacing pacemakers. The two processes require a tight interoperability, but are managed by (and executed within) different organizations (an hospital and a company that produces medical equipment).

The pacemaker lifecycle starts with the construction of a new pacemaker. As the pacemaker is built, it is charged and tested in order to verify that it works properly. If the test is successful, then the pacemaker is ready for the implant, and will be assigned to a patient. Otherwise, it is trashed. Further actions have to be taken as the pacemaker is explanted from a patient. After the explant, the pacemaker must be recharged and tested. Again, if it passes the test it is ready for implant (possibly in a different patient), otherwise it is trashed.

The patient workflow starts with the registration and examination at the hospital. If the patient needs a pacemaker, then he/she must wait for a pacemaker to become available. As a pacemaker is found, the patient can undergo the surgery implanting the pacemaker. The patient's conditions are then periodically checked. As the pacemaker needs to be changed, either because the patient conditions got better so that he/she does not need a pacemaker any more, or because the pacemaker is not working properly, then a new pacemaker is sought (if needed) and the old one is explanted.

Workflow interactions are very tight in the case of patients and pacemakers: first, the pacemaker needs to notify to patient workflow instances (also called *cases* in the following) when a new pacemaker is available. Note that the pacemaker workflow is not concerned with the selection of the patient, provided that a (needy) case of the patient workflow is notified. Then, cases of the patient workflow have to notify to cases of the pacemaker workflow when a given pacemaker has been explanted, so that it can be recharged and tested. Here, the patient case wants to notify the explant to a *specific* pacemaker case (the one managing the explanted pacemaker), not just to a generic pacemaker case. Furthermore, the patient and pacemaker conditions have to be considered when notifying pacemakers availability: if there is shortage of pacemakers, they should be assigned to patients whose conditions are critical, and analogously pacemaker that have been charged first should in general be delivered first (not to leave them for too long on the shelf, loosing their charge).

Note that with traditional workflow models it is very difficult to describe such interactions. In fact, in commercial WfMSs, workflows are typically executed in isolation with respect to other workflows. Every required synchronization and data exchange between workflows must be designed and implemented, for instance by defining ad hoc tasks that interact with the database or with the WfMS API. Furthermore, interactions would be hidden in the task implementations, instead of being explicitely specified in the workflow, with obvious disadvantages.

# 3 Defining interactions among interorganizational workflows

In order to enable workflow interoperability, we extend traditional, WfMC-like models and architectures with the capability of sending events to other processes running in the same or in different organizations.

The event notification mechanism that supports workflow interoperability is based on a publish-subscribe model. Workflows may register as *producers* (publishers) of events of a given *type* (qualified by the event name) to the local *Event Manager* (EM) of the organization where they are executed. At event registration time, the EM notifies to partner organizations that it is capable of producing events of the specified type. Similarly, workflows may subscribe as *consumers* of events of a given type. Subscriptions are also notified to the local event manager.

At execution time, cases of producer workflows may raise *instances* of the event types they are registered to as publishers, and analogously cases of consumer workflows may request instances of the event types they are registered to as consumers. New publications (requests) are always issued to the local event manager, that will deliver (satisfy) them, possibly interacting with EMs of other organizations.

In the following, we first present how event-based interaction is modeled within workflow scehmas, and then we describe how events are dispatched within and between organizations.

## 3.1 Modeling workflow interactions

From a modeling perspective, *event nodes* denote points in the workflow where an event is sent or received, thereby allowing synchronization and data exchange with other processes. Event nodes can be part of the flow structure of a workflow, just like ordinary tasks. They can be of two types: *send* (produce) nodes or *request* (consume) nodes. Send nodes notifies events to other processes; they are non-blocking, and thus they do not affect the execution of the workflow in which they are defined: the event is sent, and process execution proceeds immediately by activating the subsequent task. Request nodes correspond instead to requests of events to be received from other processes. They are blocking: as a request node is reached in the control flow of a given case, the execution (along that path) stops until an event of the specified type is received. Figure 1 shows the graphical representation adopted for event nodes.

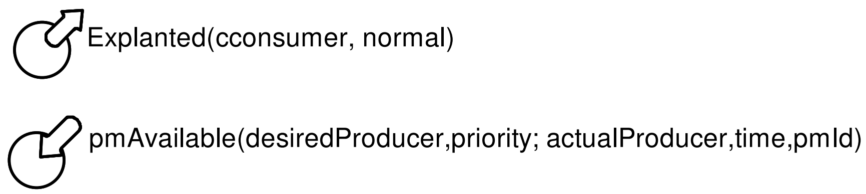


Figure 1: Graphical representation of send nodes (top) and request nodes (bottom)

Events may carry an arbitrary number of parameters. Some parameters are common to all events, while others depend on the event type. When raising an event, the sender specifies two common parameters: *consumer* and *priority*; *consumer* defines the organization (and possibly the specific case) to which the event must be notified. The distinguished value `any` indicates that no specific target is selected: for instance, pacemaker cases do not target `pmAvailable` events to a specific patient case, since they are not interested in defining who will select the pacemaker for implant, provided that it will be selected by one (and only one) `Patient` case. The selection (i.e, the dispatching) is demanded to the local EM. *priority* is a relative integer number that defines the priority level of the event instance, to determine its delivery priority with respect to other instances of the same type (higher values define higher priorities). In fact, when a consumer case makes a request for the delivery of an event of a given type, there may be several event instances ready for delivery. Thus, the EM has to choose which event should be delivered first (we allow only one event per request to be delivered, and each event only fulfills one request). The decision is based on the event priority: the dispatcher will select for delivery the event with the highest priority. If two or more events have the same priority, then the choice is based on the event occurrence time: the oldest event will be delivered first. Note that common parameters *consumer* and *priority* are not delivered to the receiver: their purpose is only to define the event dispatching modality.

In addition to the common parameters, event-specific parameters of any type known to the WfMS may be specified. Parameters can be either constants or can be associated to a workflow variable, with the meaning that the value of the variable at the time of the event raising will be used as actual parameter. Syntactically, parameters are separated by commas. Common parameters are defined first (the first value defines the *consumer*, while the second defines the priority value), and are separated by a semi-colon to event-specific parameters. For instance, the following event notifies that a pacemaker is available for implant: `pmAvailable(any,0;pmId)`. Value `any` in the *consumer* field denotes that no specific target organization or case has been selected. Value `0` in the *priority* field defines an "average" priority level. Finally, `pmId` identifies a variable of the pacemaker process, storing the pacemaker serial number; it will be delivered to the selected consumer case.

Parameters can also be specified on the receiver's side, in order to define the characteristics of the requested event and the workflow variables to which event parameters should be copied. Analogously to event publications, requests have two common parameters *producer* and *priority*. *producer* restricts the producer to be a specific organization (or even a specific case)[1], unless the `any` keyword is specified. Priority defines the "urgency" of the request: requests with higher priority will be satisfied first. Common parameters are followed by a list of workflow variables that defines where actual event parameters must be copied as the event is delivered. In the first two variables of the list, the dispatcher copies the identifier of the sender (both the organization and the case) and the timestamp of the event occurrence. Event-specific parameters will be copied in the remaining variables of the list. The formal definition of the model is provided in [3]. Figures 2 and 3 show the Patient and Pacemaker workflows, defined with the proposed model.

### 3.2 Event dispatching mechanism

In the following we present the event dispatching mechanism. We assume that event dispatching is distributed: in fact, it is unrealistic to assume that organizations agree on a centralized EM, which also has the "classic" drawbacks of centralized systems in terms of availability and performance.

Event managers receive notifications of new event publications or requests from local instances. As a new request is received, the local EM first examines the `producer` field associated to the request. If it defines a specific organization, then the EM forwards the event to the EM of the specified organization. Otherwise, the request is forwarded to EMs of other organizations, in an order that may depend on several factors: the EM may first try to look for events published from the local organization itself, it may decide on the basis of the event "cost" (for instance, a given pacemaker producer may charge more than another for delivering the event - e.g., the pacemaker), it may be predefined, or it may be non-deterministic. Different policies may be defined for each event type. Interorganizational event requests are also characterized by an attribute that defines, in case they cannot be satisfied immediately, whether the requests must be kept pending at the contacted EM site, or they can be discarded (the EM will look for events elsewhere).

Correspondingly, as an EM receives a new publication, it examines the pending requests. If the `consumer` field of the event defines a specific organization, the EM delivers the event in order to satisfy a request of the specified organization. Otherwise, the EM selects the organization whose request should be satisfied first (for

[1] We do not discuss here security and authentication issues related to the identification of the sender organization and case

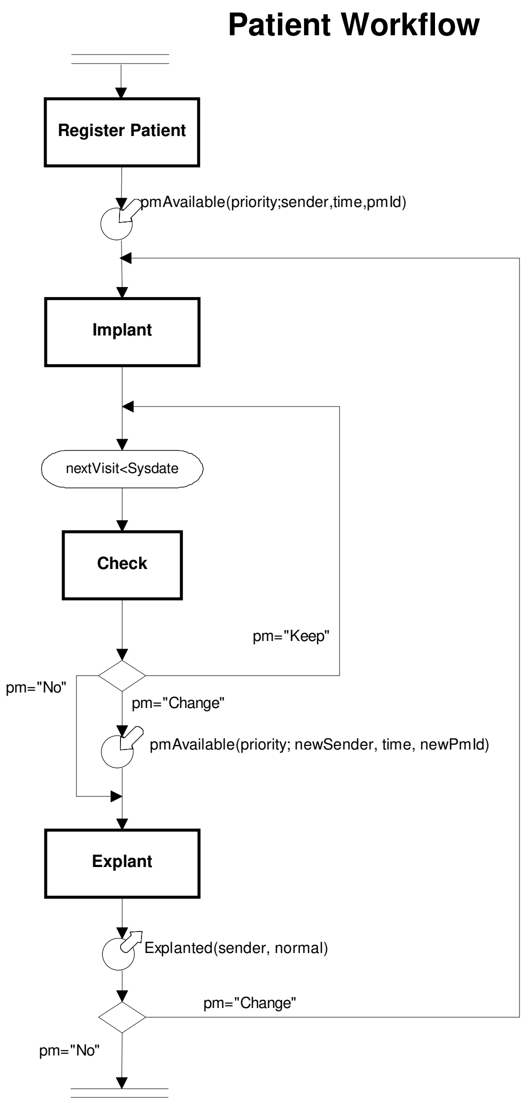


Figure 2: Patient Workflow

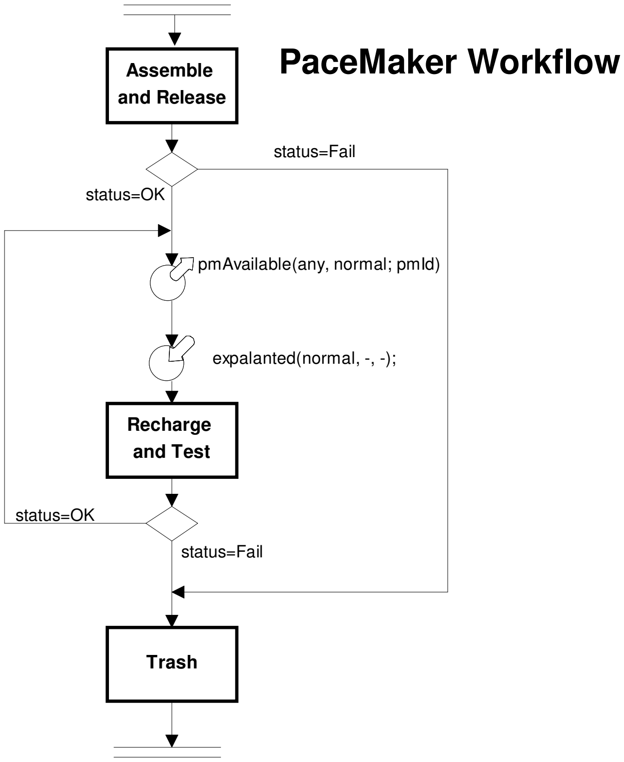


Figure 3: Pacemaker Workflow

instance according to the above mentioned factors, plus according to pending requests made from other EMs), and delivers the event to that organization. If no request has been made, the event is cached at the local EM.

Finally, as a local EM has the possibility of satisfying a request for an event of a given type coming from a local instance, it delivers the event to the requesting instance. If the number of available events is smaller than the number of requests, target instances of the local organization are selected depending first on the request priority and then on the timestamp of the request.

Note that priorities and timestamps are needed in order to compete with other instances of the same organization, while event dispatching among organization is ruled by business policies and agreements. Figure 4 exemplifies interorganizational event dispatching, where a `pmAvailable` event is delivered to the *GoodWill Hospital* organization, while no events is currently available to satisfy the request coming from *Smith Hospital*.

### 3.3 Implementation

In [3] we present how the model and the event dispatching mechanism is implemented on top of the WIDE workflow management system. The implementation is achieved by pre-compiling workflows specified in the extended model presented in the paper into a plain WIDE workflow specification and into a set of *triggers* that implement the event dispatching semantics. Triggers are executed by a trigger manager, called FAR, coupled with the WIDE engine [3]. FAR supports the definition and execution of event-condition-action rules. Rules can be triggered by several types of events (e.g., external, temporal, and data events, plus start and completion of tasks and processes), and can perform several actions, including calls to WfMS API primitives and database updates.

In the prototype implementation, each event node is translated into a “dummy”, empty task and into a trigger that, as the dummy task is activated, updates a suitable event base, signalling the occurrence of a new

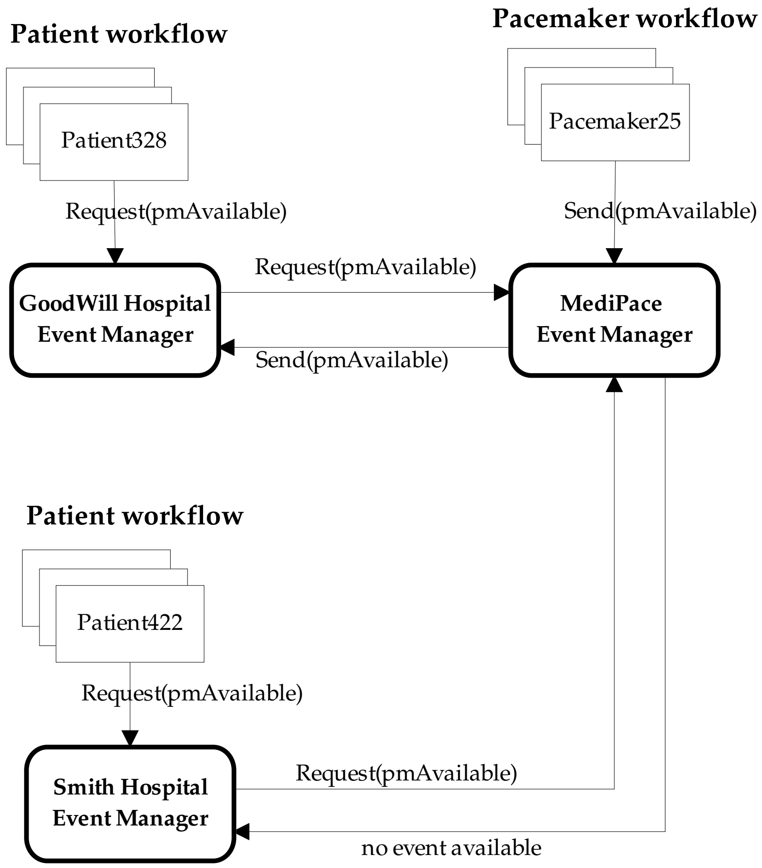


Figure 4: Event dispatching across organizations

event. Similarly, each receive node is translated into a dummy task and into a trigger that, as the dummy task is activated, queries the event base to check whether an event can be delivered and, if so, it "simulates" event delivery by canceling the dummy task (to let execution proceed), and by modifying the variables of the requesting workflow instance, storing the event parameters. Hence, in this implementation, the FAR serves as event manager, and implements the event dispatching semantics.

This approach is applicable for workflow interoperability within the same WfMS. We are presently extending this approach to enable interoperability among workflows executed on top of distributed, heterogeneous WfMSs, which is a fundamental requirement for achieving cross-organizational interoperability. To this purpose, we are augmenting the capabilities of the FAR system, which is also suited to serve as event manager in a multiorganizational context. In fact, the FAR has been engineered in order to be easiliy portable onto several operating system and database platforms, executes on top of CORBA, and is loosely coupled with the WIDE WfMS. These features allow us to create, with a limited effort, several versions of the FAR, each coupled with a WfMSs involved in the interoperation. The FAR systems of differnt organizations will then interact by exchanging events. The implementation strategy is similar to the one presented above: an extended model (allowing the definition of event nodes) is provided to the designer, which is then precompiled into the model offered by the specific WfMS and into a FAR trigger. In addition, rather than being translated into dummy tasks, event nodes are translated into tasks whose purpose is to explicitely notify to the FAR their activation (when the FAR is coupled with the WIDE engine this is not needed, since tasks activations are notified by the WfMS itself). The "send" task sends a message to the FAR and then termiantes, while the "receive" task requests an event to the FAR and waits until the event is delivered.

# 4 Analysis of interacting workflows

Interacting workflows raise a number of interesting issues related to event starvation, case starvation, deadlocks, and so on. In the following, we briefly present these issues, and we refer the interested reader to [2, 4] for a formal description of our approach.

The flow structure of a workflow defines *activation dependencies*: the completion of a node (either a task or an event node) in a given case may *activate* another node within the same case. These dependencies are *intra-case*. However, the capability of sending and receiving events introduces *inter-case activation dependencies*, where the completion of a node may activate another node in a different case. For instance, the completion of task `Assemble and Release` in a case of the `Pacemaker` workflow will cause the notification of a `pmAvailable` event, and may therefore cause the activation of the `Implant` task in a case of the `Patient` workflow. Inter-case dependencies define the desired synchronization between processes.

Activation dependencies may also be *specific*, i.e., they may occur between two nodes of two specific (but possibly different) cases. Consider for instance task `Explant` of the `Patient` workflow and task `Recharge and Test` of the `Pacemaker` workflow. The completion of the `Explant` task in a *specific* case $x$ will cause the notification of an `Explanted` event to a *specific* case $y$ of the `Pacemaker` workflow (the one managing the explanted pacemaker), that will in turn cause the activation of the `Recharge and Test` task. Thus, task `Recharge and Test` in a given case of the `Pacemaker` workflow is activation-dependent on task `Explanted` of a given case of the `Patient` workflow.

Specific inter-case activation dependencies are particularly interesting, since they imply the existence of a control flow between nodes of different cases; when they happens between two cases, we say that these cases are *bound*. Case bindings denote a very strong relationships between two (or more) cases, and correspond to the possibility of executing cases in a synchronized fashion. In the `Pacemaker-Patient` example, a `Pacemaker` case may become bound to one `Patient` case. This binding evolves dynamically, as the same patient can receive several pacemakers over time, and similarly the same pacemaker may be implanted within many patients.

When case bindings occur, cases involved in the bindings can be (conceptually) seen as instances of an *integrated* workflow. The construction and analysis of the integrated workflow may lead to the identification and removal of interference between workflows, and therefore can cause their redesign. It is important to note, however, that workflow integration is hereby proposed as a conceptual device, whereas each workflow is separately executed.

Integrated frames collect tasks from various workflows, called its *component workflows*. In the integrated workflow, new special tasks *link* and *unlink* are introduced to denote case bindings and unbinding. A link node has several incoming edges, representing component workflows, and one outgoing edge, representing the integrated frame; an unlink node has one incoming edge, representing the integrated frame, and several outgoing edges, representing component workflows.

From a methodological viewpoint, the integrated frame is built by detecting when cases of a workflow $A$ gets bound to cases of a workflow $B$. This is typically the result of an event notification, since as an event is notified, the receiver becomes aware of the identity of the sender, and activation dependencies between two (specific) cases may arise. In the `Pacemaker-Patient` example, a case of the `Pacemaker` workflow becomes bound to a case of the `Patient` workflow as a `pmAvailable` event is delivered. In fact, in that instant, the dependency between tasks `Explant` and `RechargeAndTest` arises. Case bindings are represented by link nodes, which replace the (send node, receive node) pair that generates the binding. Once workflows are linked, the designer should understand which nodes are executed by preserving the binding and identifying the pair (send node, receive node) that concludes the dependency, thus identifying a dual unlink node, that represents the logical point in the execution of the workflow where case bindings is changed or no longer effective. Again, the unlink node replaces the (send node, receive node) pair.

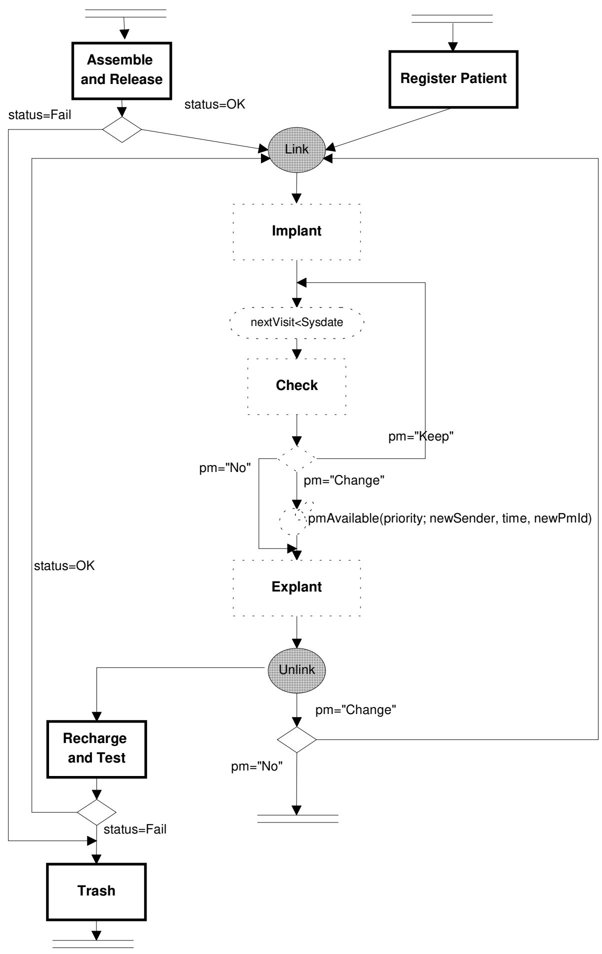


Figure 5: Frame resulting from the integration of Patient and Pacemaker workflows

In Figure 5, we illustrate an integrated view of the `Pacemaker-Patient` interacting flows. Nodes executed during the binding are depicted with dashed borders. We believe that such integrated diagram, although it has to be manually constructed (some automated support can however be provided) is a useful device for analyzing workflow interactions.

Interacting processes also raise a number of interesting problems related to their analysis, as reported in [16]. Two "classic" problems are *starvation* and *deadlock*. *Event starvation* occurs when a published event is not delivered to consumers, and remains in the event base indefinitely. *Case starvation* represents the dual problem: a case may be left indefinitely waiting for an event. Case starvation is even worse than event starvation, since a case is blocked and cannot proceed (unless flow execution has been forked on several paths).

A limited form of static analysis can be performed, by looking at activation dependencies, in order to detect the occurrence of situations leading to *deadlocks*. A sufficient condition for the existence of a deadlock occurs when a node $n$ is (transitively) activation-dependent on itself, meaning that the activation of $n$ requires a previous completion of another instance of $n$, which is impossible. At run-time, a deadlock may occur when cases of a given workflow $A$ are blocked, waiting for events produced by cases of workflow $B$, which are in turn blocked waiting for events produced by cases of $A$. However, such condition may be resolved by newly started cases of $A$ or $B$, or by cases of other workflows that are allowed to produce events consumed by $A$ or $B$. See [2] for a more formal analysis of these properties.

# 5 Concluding remarks

This paper presented a workflow model that enables a flexible form of semantic interoperability among processes executed in several organizations. Separate workflows are executed within the different organizations, and communicate by means of events. A distributed event dispatching mechanism has been presented, where event managers dispatch events according to priorities of event publications and requests, business agreements among organizations, cost, and so on. The paper also discussed issues related to the analysis of the properties of interacting processes and showed how an integrated view of interacting processes can be constructed starting from the separate workflow definitions.

The model can be used to define a variety of behaviors, by exploiting (possibly predefined) workflow fragments. For instance, in [3] it is shown how, with the proposed model, it is possible for a case to request an event, but then take an alternate actions as no event is delivered within a specified deadline. In [3] we also present an implementation of this model, based on its translation into a basic model and into a set of triggers that take care of dispatching events as appropriate.

# References


[1] C. Brown, J. Randell, and M. Wray. The Aurora event distribution system. Available at http://pooh.hpl.hp.com:80/ cb/Aurorapaper.html.

[2] F. Casati. Extending workflow technology to support workflow interoperability. Submitted for publication.

[3] F. Casati. *Models, Semantics, and Formal Methods for the Design of Workflows and Their Exceptions*. PhD thesis, Dipartimento di Elettronica e Informazione, Milano, Italy, Dec. 1998.

[4] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Semantic workflow interoperability. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT'96)*, Avignon, France, Mar. 1996. Springer-Verlag, Berlin.

[5] S. Ceri, E. Di Nitto, A. Discenza, A. Fuggetta, and G. Valetto. DERPA: a generic distributed event-based reactive processing architecture. Technical report, CEFRIEL, Mar. 1998.

[6] A. Discenza. Filtering events in a distributed architecture. Technical Report 98.079, Dipartimento di Elettronica e Informazione, 1998.

[7] C. Hagen and G. Alonso. Beyond the black box: Event-based inter-process communication in process support systems. Technical Report 303, ETH Zurich, 1998.

[8] D. Hollingsworth. The workflow reference model. Technical Report WFMC-TC-1003, 1.1, Workflow Management Coalition, 1994.

[9] H. Lam and S. Su. ECAA rules and rule services in CORBA. Technical Report cf/97-01-09, Object Management Group, 1997.

[10] *CORBAservices: Common Object Services Specification*, 1997.

[11] Object Management Group. Errata relative to Workflow Management Facility, joint revised submission. Technical Report bom/98-07-15, OMG, 1998.

[12] Object Management Group. Workflow Management Facility, joint revised submission. Technical Report bom/98-06-07, OMG, 1998.

[13] SWAP IETF working group. Simple Workflow Access Protocol. Available at http://www.ics.uci.edu/ ietf-swap/.

[14] The Workflow Management Coalition. Interoperability Abstract Specification. Technical Report WFMC-TC-1012, Workflow Management Coalition, 1996.

[15] The Workflow Management Coalition. Process definition interchange v 1.0. Technical Report WfMC-TC-1016-P, Workflow Management Coalition, 1998.

[16] W. van der Aalst. Interorganizational workflows: an approach based on message sequence charts and Petri nets. *Information Systems*. to appear.