A Model to Support Collaborative Work in Virtual Enterprise

Franck Wynen, Olivier Perrin, and Claude Godart

LORIA – INRIA – CNRS – UMR 7503
BP 239, F-54506 Vandoeuvre-lès-Nancy Cedex, France
{franck.wynen, olivier.perrin, claude.godart}@loria.fr

Abstract. Current needs for virtual enterprises projects management lead to adapt production oriented management systems (i.e. workflow) to collaboration and necessary co-decision features. Such projects are submitted to unpredictable events and complex organization where human mind is the main process. Moreover, a project management system must take care about individual participant self interests regarding on their private assets and working customs. Finally, it must federate each participant around a common point of view based on a sufficient trust level on information sharing systems. This paper presents both an extended model, based on workflow concepts where human dimension and flexible capabilities take place in front of the scene, and an architecture that supports this model.

1 Introduction

The emerging e-business Web economy requires an agile enterprise that can work more directly with suppliers and customers and respond more rapidly and intelligently to change. Technologies such as the Internet are beginning to transform traditional business models in this direction. Business pressures - margin erosion, channel proliferation, rising customer expectations, time-based competition, and faster product commoditization - are placing increased emphasis on how organizations operate and interoperate with other enterprises [4]. To enable customers to adapt to these dramatic changes in the business environment, middleware will increasingly be required to provide dynamic and flexible integration between partners in the value chain. Although new technologies will be needed to enable such integration, they will have to work seamlessly with existing inter-enterprise business processes (e.g., EDI-Electronic Data Interchange) and leverage investments in existing enterprise application integration (EAI).

This paper is about collaborative work within a virtual enterprise (VE). A virtual enterprise is distributed over the time, space, it spreads across several organizations and it is dynamic and temporal, in the sense that partners can change. It needs to manage shared resources, organizational models, process models, and also contracts between organizations/partners that belong to it. Let us see on an example what are the specific issues of VEs, in particular about their processes that are distributed across several organizations [14]. We are currently involved in a project with European car manufacturers and equipment suppliers. When a car manufacturer wants to design and produce an electronic embedded architecture, it calls upon equipment suppliers. This means that the process of defining, refining, implementing and producing such an architecture is spread across several participants and that each participant implements only a subset of activities of the overall process. In addition, the same activity can be implemented by several enterprises (several equipment suppliers for instance). This example

shows the need for both collaboration and cooperation support. It also shows that we have to coordinate and synchronize all the participants of distributed multi-enterprises processes. Last, it is important to support interoperability and interaction between different organizations, while maintaining the privacy of information and the autonomy of each participant [5]. This means that we must find a balance between competitiveness and collaboration, between transparency and confidentiality, and between autonomy and global consistency.

To summarize, main questions are: how to brainstorm and work, how to synchronize different teams and how to handle information interchange between organizations. Classical answers to these questions are to set a list of scheduled meetings (or calls). Another way is to exchange documents by e-mail, but e-mail can be lost after some time, while the content can be needed again in the future. Moreover, searching e-mail is limited to email client and e-mail is not hyper-linked and is not structured. So content cannot easily be grouped into related topics. Then, e-mail is not version controlled, and it is very difficult to look at a document history. To manage data interchange, the key technologies for enabling dynamic process integration can be EDI. But EDI, while being appropriated for electronic document exchange, suffers from its high cost and its dependency on specialized deployment skills. Therefore, its adoption in a small and dynamic context such as the virtual enterprise context is not totally accurate. However, the concepts of EDI must be considered (for instance, the concept of common language that specify how partners will interact, and the contract that define and enforce what the interaction (conversation) protocol will be).

We also claim that current workflow management systems are not totally satisfactory to answer the problems raised by virtual enterprises and cross-organizational processes. A first issue concerns the fact that definition and implementation are strongly tied. This is a problem because if a collaborative process is specifically tied to a platform, an application or a protocol, changing the implementation technology according to the needs of all the participants of the VE is not possible. Moreover, it is difficult to support and to manage interactions (the process is spread over organizations). The second issue is that current systems do not satisfactorily manage the flexibility needed for collaborative processes. A centralized management is adequate for an intra-enterprise process management but is not well suited in the context of a cross-organizational process management. When multiple parties are involved in a process, they are often separated by firewalls, have self-interests, and do not wish to share all the process data [3]. Then, the need of constantly changing and interacting with partners (and possibly new partners) is an important feature that is not yet supported. A third issue concerns the need of composability. This concept represents the ability for one participant of the cross-organizational process to describe the interactions between existing workflow activities and to obtain a composed process. Given the ability to compose a new process using activities and data from the participants of the virtual enterprise represents the opportunity of building the process, as it is needed within the virtual enterprise. For instance, how can we compose a process managed by a J2EE platform and a process managed by a .NET platform?

In the next section, we refine our motivation and an example extracted from the project we are involved in. Then, we detail in section 3 our model. Section 4 shows how this model applies in a real world example, and section 5 presents the implementation. Related works are considered in section 6. Finally, section 7 concludes.

2 Motivation

A classical process definition consists of a network of activities and their relationships [18],

[20]. This definition applies to VE, but partners from different enterprises also have to realize collaboration activities to focus on the objectives. To do so, we need a model that supports both information interchange/sharing and coordination. The synchronization point (SP) model that will be detailed in section 3 is our answer.

In the model, information are under version control and it is easy to publish intermediate results, to find out who changed what and when, to see differences between versions or to see and access previous versions of documents. Moreover, information sharing takes into account contracts that allows for filtering and delivering the right information to the partners. As an example, let us consider two processes P and Q that modify a same object. Using the traditional single process model approach, we obtain the result depicted in figure 1. In fact, long-running activities, distribution and relative autonomy increase the risk of divergence between two different versions of the same object. Thus, as shown in figure 2, processes P and Q could produce an irreducible gap and a synchronization activity, placed a posteriori, could fail (annihilating P and Q efforts).

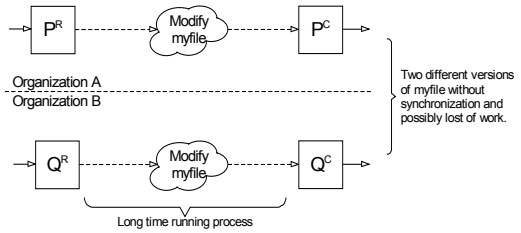

Fig. 1. Without cooperation

One of the motivations behind the synchronization point model is to allow participants of the cooperation for controlling, evaluating and reducing the gap between successive versions of an object (figure 2b). Intermediate versions of an object are available for each cooperation participants thanks to the synchronization point repository. In order to enforce virtual presence, a synchronization point also provides awareness, such as delivery notifications of intermediate object versions in a common repository.

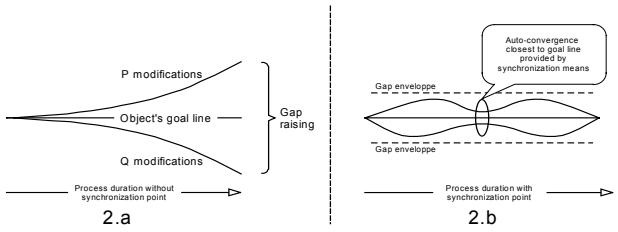

Fig. 2. Version gap

We also need a component where two or several processes can be coordinated. The definition of the coordination (i.e. synchronization) is external to both partners and is defined at the beginning of each collaboration (thanks to a contract definition). This is done in a flexible way where revision (addition, deletion or modification) should be allowed as the work goes further. So another motivation is to provide a way for coordinating internal processes of the partners, without enforcing them the use of proprietary tools or methods. Figure 3 presents our point of view. Let us now suppose that we have to coordinate former processes P and Q. A synchronization point (hexagonal process) is started when P or Q starts. This particular coordination process is external to both P and Q. When there is a need for cooperation between

two partners of the VE, we redirect each control flow (those involved in cooperation) into this process. Its end (corresponding to a synchronization) is re-branched to the initial flow (ending processes involved in cooperation).

Then, a third motivation is to be as flexible as needed, meaning that we only use a synchronization point for cooperative activities, and that we support refinement by providing the ability to add, delete or modify the control flow if needed (for instance, we support addition of a SP, addition of a partner to a SP, merge of two SP...). Thanks to the precise semantics of the SP model, we are able to guarantee the consistency and the execution of the cooperation phase between two or more partners. As this model is external, it does not call into question both the organizational models or process models of partners. Moreover, it is always possible to calculate and to control all the synchronization points.

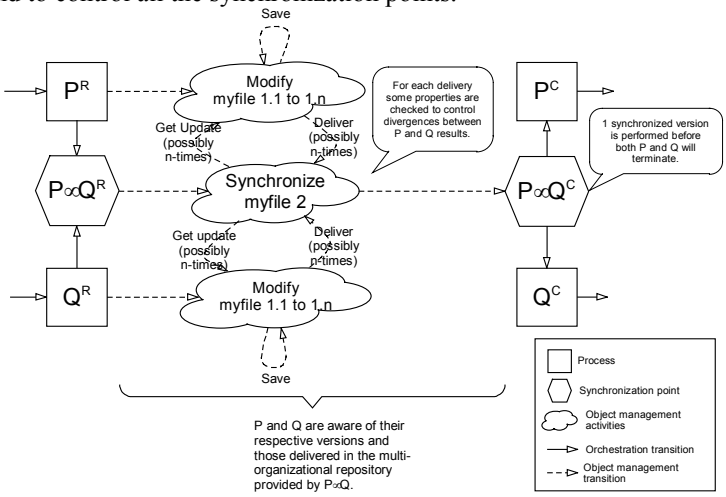

Fig. 3. Parallel cooperation

As said in the introduction, we are involved in a project with European car manufacturers and their suppliers. Here is a basic example of a virtual enterprise built on three organizations: a car manufacturer *CM* that requests some equipments providers *EP* for a wiper system that cooperate with a rubber manufacturer *RM*. First, the *CM* describes its process.

- *CM1*) Wiper physicals feature definition in accordance to windscreen and wiper motor properties, and request for equipments providers,
- *CM2*) Proposal reception, tests, integration and commercial contract negotiation.

When requested, equipment provider has to define its process.

- *EP1*) Request management with awaited features and first round of technical features definition,
- *EP2*) Request for a rubber manufacturer *RM* and common second round of technical features definition,
- *EP3*) Wiper proposition to *CM* design team, marketing action to enforce proposed product and contract negotiation with *CM* commercial team for one part and *RM* for another.

Then, for the rubber manufacturer *RM:*

- *RM1*) *EP* request treatment wiper feature and technical features definition with *EP,*
- *RM2*) Technical meeting with both *CM* and *EP*, and contract negotiation with *EP* commercial team.

When done, we separate objects of the process as follow:

- *pf*) a wiper physical feature document,
- *tf*) a wiper technical feature document,
- *cmc*) a commercial contract between *CM* and *EP*,
- *rmc*) a commercial contract between *RM* and *EP*.

And relations among objects themselves or among processes such as:

- *CM1* "create, modify and perform" *pf*
- *EP1* "follow" *CM1*
- *EP1* "create and modify" *tf*
- *EP2* and *RM1* "follow" *EP1*
- *EP2* and *RM1* "modify" *tf*
- *CM2*, *EP3* and *RM2* "follow" *EP2* and *RM1*
- *CM2*, *EP3* and *RM2* "modify" *tf*
- *CM2* and *EP3* "create, modify and perform" *cmc*
- *EP3* and *RM2* "create, modify and perform" *rmc*
- *EP1*, *EP2*, *EP3, RM1*, *RM2* and *CM2* "read" *pf*

Fig. 4. Set of relations

3 Process Model

Let us now present the synchronization point model. A synchronization point specifies the partner relationships for a given activity. In fact, we do not describe the relationship between partners but relationships between objects. In order to ensure enterprises privacy and autonomy needs, the SP does not need any information about internal process but only requires results or events notifications of the corresponding process. To preserve the autonomy of partners, we divide their contribution into parts of autonomous work. The work items - results, intermediate results or process events - are sent to the synchronization point and can be shared with the partners, while being versioned. Thus, the collaborative process becomes a set of synchronization points that coordinate the autonomous work of the partners.

Then, when each partner reaches a given milestone, the synchronization phase starts and tries to reconcile all the different versions of an object. All the partners are working with each other in order to do the reconciliation (the *do* activity). This step is not fully automatic (we provide *checks* that will help to reconcile all the different versions by using boolean expression language for conditions), the system helps to find differences but the final decision remains a human decision. If the synchronization activity succeeds, the control flow tells to partners what to do next. In the opposite case – the reconciliation fails – partners have the opportunity to modify the control flow by modifying one (or maybe several) activity or synchronization point, by adding or deleting activity or synchronization point or to reschedule (modifying delay for instance). This is the *replan* activity of the synchronization point. So, a SP must provide:

- *Check*. Control is mandatory to enhance the synchronization between partners. We need information about how their work is progressing. To control the collaborative work, we use functions that must satisfy some criteria. Then, we compare and evaluate the checked results regarding a set of predefined criteria (objective fulfillment, quality, schedules, etc.).
- *Do*. The synchronization process requires a set of actions that will reconcile the different version of an object. Due to the nature of an object, it is difficult to render this process fully automatic. The *diff and merge* activities are only available for some formats. When this activity is achieved, a reconciled version is delivered to the rest of the process.
- *Plan*. Collaboration processes involve human participation and are characterized with unexpected situations and relatively long duration. Flexibility capabilities are required and

we must support dynamic process changes. Usually, we do only ad hoc corrections that don't need a process change (i.e. a deadline). But we should be able to adjust the process by planning additional activities (possibly a SP) or by updating an activity.

In order to support the SP activity, we need some features that will help us to analyze, to support communication and to provide awareness. The analyze is based on the history of process execution (both execution traces-control and versions evolution-data). The communication is one of the essential functions for collaboration achievement. It allows for participants to communicate their results by exchanging files or any other information, and for applications to notify some events (process states for instance). Then, awareness helps to report to partners whenever any changes occurs and gives him the opportunity to check the current work progress. This allows for reducing the presence virtuality.

3.1 Definitions

In the following, we will denote a process with upper case letter such as P, Q or R and we introduce $O(P)$ the set of objects manipulated by the process P and denote its elements with lower case letters such as x, y or z. Objects could be files, data, goals, sub-processes and activities.

3.1.1 General Definitions

WFMC defines business process as a set of one ore more linked procedures or activities that collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships [20].

3.1.2 Relations

We consider that two processes cooperate if it exists a relationship between their objects (i.e. indirect cooperation). We denote this relation *rel* and use $x\ rel\ y$ notation. In our example, $EP2$ and $RM1$ modify *tf*. Without control or cooperation we will obtain two different versions of the same file. In order to express the relationship between both modified files we write:

E1) $tf \in (O(EP2) \cap O(RM1)) \mid tf_{EP2}$ *"is equals to"* tf_{RM1}

We can also describe a relationship between two different objects. Suppose that we would concatenate *tf* and *pf* in a single file named *w_features*, we should write:

E2) $tf \in O(EP2), pf \in O(RM1)$ *and* $w\_features \in (O(EP2) \cap O(RM1)) \mid w\_features =$ *tf "append to" pf*

Those examples also enlighten that object relationships are tightly coupled to object types. Merging functions could not be the same for two text files and for a video and an audio sequence. That is, we cannot define every relationship in this definition model (they should be defined in implementation model thanks to specifics tools).

3.1.3 Sat Functions

When a relationship exists between two or more objects, we define a Sat function that satisfies the relation, denoted $Sat(x\ rel\ y)$. This activity may consist on a check function that verifies if x and y realize their relation or a function that acts on one or more objects in order to satisfy the relation. In other words, we may use static functions to check whether the relation is satisfied or dynamic functions to do something for that. Obviously, in most cases, we may use both to perform complex relations among process objects.

$$\forall x, y \ (x \ rel \ y) \Rightarrow \exists \ Sat \ (x \ rel \ y) \quad (1)$$

For example E1, we may have a Sat function that first compares tf_{EP2} and tf_{RM1} as DIFF, and then proposes corrections to participants involved in processes *EP2* and *RM1*.

Moreover, as proposed above, a process may contain some sub-processes, activities or process fragments. That is, we can also consider a process relation as an object relation:

$$\forall P, Q \ (P \ rel \ Q) \Rightarrow \exists \ Sat \ (P \ rel \ Q) \quad (2)$$

3.1.4 Synchronization Point

A synchronization point is a particular process denoted $P \infty Q$ (pronounced "*P* synchronized with *Q*") such that:

$$\forall x \in O(P), y \in O(Q) \ and \ (x \ rel \ y) \ \exists P \infty Q \mid Sat \ (x \ rel \ y) \in O(P \infty Q) \quad (3)$$

A synchronization point allows for synchronizing one cooperation between one or more processes. It includes every Sat function that resolves relations between their objects. In fact, it exists a strong relation (i.e. a bijection) among the existence of a synchronization point and the existence of relation among process objects. In other words, we assume that 1) we cannot have a synchronization point without having some object relation and 2) all the Sat functions that resolve relation among process objects of two given processes *P* and *Q* are located in the $P \infty Q$ synchronization point. Moreover, we will see in 3.5 how the synchronization point is able to offer a versioned space for objects.

Considering examples E1 and E2, we have two Sat functions $Sat1(tf_{EP2} \ rel_1 \ tf_{RM1})$ and $Sat2(tf \ rel_2 \ pf)$ involved in one single synchronization point $EP2 \infty RM1$.

Let us consider a synchronization point as a process that includes parallel activities. Each of them resolves the Sat function corresponding to the relations among process objects involved in the cooperation. We separate individual behaviors of processes *P* and *Q* and the behavior of their synchronization point. In the same way, if a single process *P* owns the objects involved in a relation, we can consider the corresponding Sat function as a sub-process of $P \infty P$:

$$\forall x, y \in O(P) \ (x \ rel \ y) \Rightarrow \exists P \infty P \qquad (\text{with } P \infty P \neq P) \quad (4)$$

3.1.5 Synchronization Activities

A synchronization activity is the particular activity that resolves a Sat function. It is included in a synchronization point and is denoted $\frac{P \infty Q}{Sat(x \ rel \ y)}$ where $Sat(x \ rel \ y)$ is the Sat function that resolve the relation of two objects such that $x \in O(P)$ and $y \in O(Q)$. Thus, in previous example, $Sat(tf_{EP2} \ rel \ tf_{RM1})$ is denoted $\frac{EP2 \infty RM1}{Sat(tf_{EP2} \ rel \ tf_{RM1})}$. In fact, a synchronization point includes all the Sat functions and all the objects involved in relations between processes. We can now define the synchronization point object set such as:

$$O(P \infty Q) \supseteq \bigcup_{i=1}^{n} x_i \in O(P) \cup \bigcup_{j=1}^{m} y_j \in O(Q) \cup \bigcup_{k=1}^{n*m} Sat_k \left(\bigcup_{i=1}^{n} x_i \ rel_k \bigcup_{j=1}^{m} y_j \right) \mid \exists (x_i \ rel_k \ y_j) \quad (5)$$

We use "include or equals" operator because we allow $P \infty Q$ to have some relation independent objects or processes. Based on examples E1 and E2, we describe $EP2 \infty RM1$ object set and we add a synchronization independent sub-process *Other_process* such that:

$O(EP2 \infty RM1)=\{tf_{EP2}, tf_{RM1}, tf, pf, w\_features, Other\_process, Sat(tf_{EP2} \, rel \, tf_{RM1}), Sat(tf \, rel \, pf)\}$

3.1.6 Visibility

In the synchronization point definition, we define synchronization point activities as Sat functions. Thus, we must distinguish relations involving objects and processes and take into account that processes collaborate themselves while objects collaborate through processes or activities that involved them. To explain this difference we must keep in mind that an object (i.e. not a process or an activity) is not able to do anything (e.g. it cannot fire any event when its state change). It does not exist as itself but through the process it is involved in. Moreover, in order to preserve privacy and autonomy of process participants, we must reduce process object inter-visibility as tiny as cooperation needs. In a relationship between two processes P and Q such that $P \in O(R), Q \in O(S)$, we may have $(P \, rel \, Q) \Rightarrow P \infty Q$ but not $R \infty S$.

Considering a workflow management system activity, we could have the same difference. In fact, we cannot differentiate its behavior and the behavior of objects it manipulates. So, we prefer considering workflow management system activity as black box due to the fact that some workflow managements systems do not respect the WAPI [19] specifications. Thus, to build a relation among a workflow activity *wa* and a process P, we must before encapsulate *wa* as an object in a process Q and then declare the cooperation such as:

$$\forall P, wa \; (wa \; rel \, P) \Rightarrow \exists Q \mid wa \in O(Q) \wedge Sat(wa \, rel \, P) \in O(P \infty Q) \qquad (6)$$

For example, we can imagine a situation where *RM2* call for web services about the rubber price in order to evaluate its product cost. We create a sub-process *WSB* that is bound to a classical workflow activity. Then, we are allowed to control the beginning of this activity (using its API), to run it and to read its results (the price is extracted from SOAP/XML message response).

3.1.7 Multi-Process Relationship

We have defined above the meaning of $x \; rel \; y$ and its consequences among processes that include x and y objects.

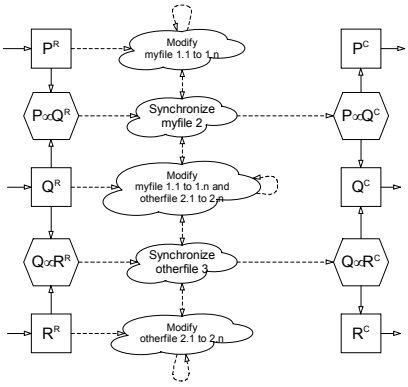

Fig. 5. Two synchronization points

We now consider relations among three or more processes to evaluate the meaning of $P \infty Q \infty R$. First, we must keep in mind that processes could be part of different organizations. That implies that the set of common shared objects must be as reduced as necessary. At process level, we have as many synchronization points as process relationships. Figure 5 shows a situation where Q is involved in two cooperation processes.

Thus, P and R are not aware of each other, preserving their autonomy. In fact, we define a privacy rule where a process is involved in a synchronization point if and only if a relation exists among one of its objects and at least one object of the synchronization point. We write:

$$\forall x \in O(P_1), n \geq 1 \quad x \in O(P_1 \infty P_2 \infty .. \infty P_n) \Rightarrow (\exists y \in O(P_2 \infty .. \infty P_n) \mid x \, rel \, y) \tag{7}$$

This definition exposes a recursive way to involve a process in a cooperation. This allows detecting when visibility agreements are needed for an incoming process. Thus, we are allowed to propose a synchronization point model where we share "*all necessary but no more and at least, under the control of each participant*". In reference of the previous example, if a relation exists among *myfile* and *otherfile* and if P and R accept to share those files with each other, we can merge our two synchronizations points in a single one. Figure 6 depicts a situation where P, Q and R work together.

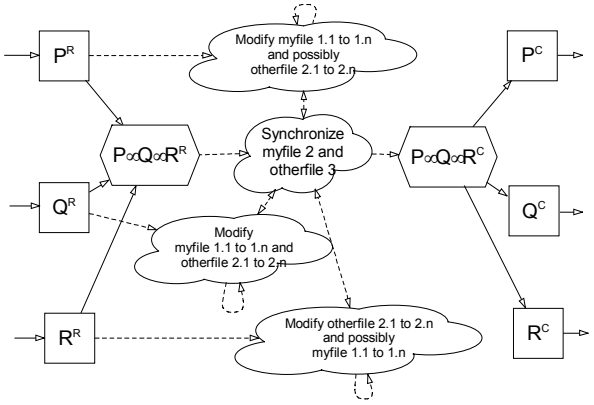

Fig. 6. Merged synchronization points

3.2 State Model

In section 3.1.6, we discussed about the differences between process and object relationships. Then, we consider in a systemic approach that processes interact with the system through their behavior and then their successive states. Those interactions allow for declaring process orchestration formulas where, for example, the final state of a process corresponds to the start state of another in a serialized form. Moreover, we define a synchronization point such as a particular process and we assume that it is also a part of the system and therefore should interact with it. On the other hand, we can consider a process object through its state to define interactions with the system. To refine our point of view, we propose two state models and we express what kind of object they represent.

3.2.1 Process

Our process state model is WfMC compliant [19]. We use the defined states: *Initiated* (I),

Running (R), *Suspended* (S), *Terminated* (T), *Active* (A) and *Complete* (C). Thus, we denote the current state of a process or an activity with an exponent letter (e.g. P^T means that the process P is terminated).

3.2.2 Object

For object, we introduce an original state model due to the synchronization point.

States

Defined (d). The object is defined but empty. We can define relations among objects and therefore may create synchronizations point. This is generally the state of an object that is involved by a process at design time.

Instantiated (i). The object now exists in a process repository but is not yet manipulated. It should be the state of a newly created object or the representation of a binding between an object and its envelope.

Modified (m). The object is currently modified by one ore more processes or activities. The process that manipulates the object may read in and write to its own repository.

Read (r). The object is in read mode. That is, if an object is involved in a relation, we may have a synchronization activity even if it does nothing else than checking the object availability. This is a final state.

Delivered (v). The object is posted to the repository whose process depends on. If addressee is a synchronization point, that corresponds to the beginning agreement of the synchronization activity which involves the object in a relation. From the process point of view, the work on the object is performed and we assume that it will never be modified. This is a final state.

Transitions

Figure 7 depicts the object state model. However, we assume that some transitions should be unavailable for some object types (e.g. a process should not modify an automatic's activity such as workflow but just read its state). We distinguish 1) save transition that consists in storing the object in its repository during process time and 2) deliver transition that achieve a modification session inside a process and proposes a result to the community involved in a synchronization point.

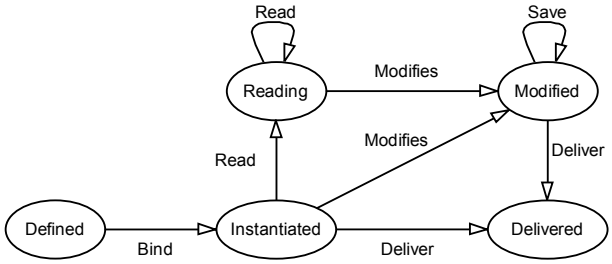

Fig. 7. Object state model

3.3 Semantics and Behavior

We have defined a process model, introduced the synchronization point concept and described process and object state model. In this context, we define a language in order to describe relations between objects and processes.

We propose the notation *(object or process)* $\overset{State}{\Rightarrow}$*Transition (object or process)* to express that a

transition will depend on an object or process state. We assume that the left part (i.e. a pre-condition) is a boolean expression and should be the result of *((object or process)* ^{State}*==State) equals true*. Thus, we extend pre-condition as well to conditions upon dates such that *(date)* $\Rightarrow$*Transition (object or process)* where, *(date)* means *((pre-defined date)>=(now)) equals true*. A clearest notation is:

$$P^{State} \Rightarrow Transition\ (Q) \qquad \text{(i.e. if } P \text{ is in state "}State\text{" then transition on } Q\text{)} \qquad \textbf{(8)}$$

Moreover, the right part is an action that involves an object or a process in a transition as defined in a state transition model. For example, if we ask for a process Q to run when another process P will complete (i.e. such as a sequential routing), we denote $P^{C} \Rightarrow Run\ (Q)$.
That corresponds to the following verifications:

- C is a state for P object's type
- State of P is equals to *Complete*
- *Run* is a valid transition for Q object type
- State of Q is compatible to a *Run* transition (i.e. Q is *initiated*, *suspended, active* or *running* and Q is a process)

Then, if all the conditions above are satisfied, Q is now in *running* state. We are able to distinguish explicit conditions that compose a scenario and implicit ones that return exceptions. Thus, first to third checks for inconsistency at process definition step while the later allows for capturing process errors at runtime.

3.4 Cooperation Contracts

We have defined object relation such as a necessary condition to the definition of a synchronization point. With three or more objects, relations may be complex and should generate more than really necessary synchronization activities. When an object is involved in several relations at the same time, we must synchronize it in a single SP or we must define two or more SP in order to serialize the modifications on the object. We assume that:

$$\forall x \in O(P), y \in O(Q), z \in O(R) x\ rel\ y \wedge x\ rel\ z \Rightarrow P \infty Q \infty R \vee (P \infty Q) \infty R \vee (P \infty R) \infty Q \qquad \textbf{(9)}$$

For example, in order to perform a three-part cooperation, we take interest in Q and R object inter-visibility. Relations *x rel y* and *x rel z* mean that P and Q for a part and P and R in another have accepted to share something but not Q and R yet together. In fact, we assume that a contract exists between P and Q and another exists between P and R. That is, if those contracts include privacy protection clauses (e.g. that protect a part of a file), we must not provide entire visibility to R in Q work and vice versa. Thus, faced to a multi-relational description, we should test whether:

- Contracts exist and allow users to share their objects in a common synchronization point. This case means that a contract also exists between Q and R and allows involving P, Q and R in a single synchronization point.
- Contracts do not exist and we create synchronizations points for each independent cooperation.
- Contracts do not exist and we meet process participants for a deal. In fact, we propose new contracts in order to reduce the number of synchronization points.

Therefore, if *x rel y*, *x rel z* and Q should process during R we are allowed to propose a global synchronization point if and only if both Q and R participants accept to work together.

Otherwise, we must serialize two different synchronization points, selecting for example $P \infty Q$ and pushing its result in a cooperation with R such as $(P \infty Q) \infty R$ (i.e. different from $P \infty Q \infty R$).

Thus, we enlighten a link between relations among objects and contracts among organizations that own them and we claim that a relation among objects corresponds to a contract and a Sat function. The first represents work group awareness through process participants, privacy policy associated to objects and communication means description and the second proposes tools designed to resolve relations among objects.

3.5 Repository and Versioning

For each process (i.e. P), we define a repository where objects involved in the process (i.e. $O(P)$) are stored during processing time. The process activities use this repository to manipulate objects. For each synchronization point, it exists a repository that allows all the process participants involved in the cooperation for sharing objects. So, given two processes P and Q that collaborate we may have three repositories for storing $O(P)$, $O(Q)$ and $O(P \infty Q)$. All the repositories encapsulate objects and give the ability to define privacy policy in a multi-organizational virtual enterprise. A repository hides objects inside a process scope.

Different versioning policies exist for synchronization point and for processes. In fact, a synchronization point repository is considered as a contractual view of one or more organizations version spaces while process and activities repositories are considered as workspaces dedicated to evolution of data (using sub-versions).

That is, synchronization points, individual processes (those which are not involved in a cooperation) or merely processes that manipulate some relation independent objects (objects that are not shared) have a direct read and/or write access to organizational repositories. Therefore, they use the versioning policy (if it exists) of the organizational repository they check in or they check out. On the other hand, processes manage independently their own versions in their own repository. So, version mismatches induced by this policy are hidden by privacy rules on process repositories. Moreover, this avoids an exponential amount of version synchronization messages between organizations over Internet. When a synchronization point completes, version spaces of the processes involved are reconciled, and the synchronization point version space is reconciled with the organizational repository. This schema for version management also applies for nested processes and their respective repositories.

4 Example

If we take the relation set presented in figure 4, we obtain:

$1)\ CM1^{R} \Rightarrow Modifies(pf)$ $\quad$ $6)\ RM1^{R} \Rightarrow Modifies(tf)$

$2)\ CM1^{C} \Rightarrow Run(EP1)$ $\quad$ $7)\ (EP2.RM1)^{C} \Rightarrow Run(CM2.EP3.RM2)$

$3)\ EP1^{R} \Rightarrow Modifies(tf)$ $\quad$ $8)\ CM2^{R} \Rightarrow Modifies(cmc,tf)$

$4)\ EP1^{C} \Rightarrow Run(EP2.RM1)$ $\quad$ $9)\ EP3^{R} \Rightarrow Modifies(cmc,rmc,tf)$

$5)\ EP2^{R} \Rightarrow Modifies(tf)$ $\quad$ $10)\ RM2^{R} \Rightarrow Modifies(rmc,tf)$

Here, the goal is to demonstrate how our language can represent an informal process description and how our model can detect inconsistencies or synchronization needs. Moreover, for best reading, we have intentionally remove pf read operations from the current

demonstration. In fact, in a real process definition, we should merely add *pf* to all process object sets (e.g. $EP1^{R} \Rightarrow Read\ (pf)$).

Figure 8 represents the current defined situation and enlightens some inconsistencies. In fact, grayed areas A and B shown that evident synchronization activities loose. We recognize in A the pattern described in figure 3 and we consider formulas 4, 5 and 6 as the language representation of this situation. That means *EP2* and *RM1* modify *tf* and are running at the same time. Then, our language allows detecting the relation on *tf* (through *EP2* and *RM1* modifications) and deducting the synchronization activity denoted $\underline{EP2 \infty RM1}_{tf}$. In the grayed area B, we detect three relations. The first is based on *tf* modifications and involve *CM2*, *EP3* and *RM2*.

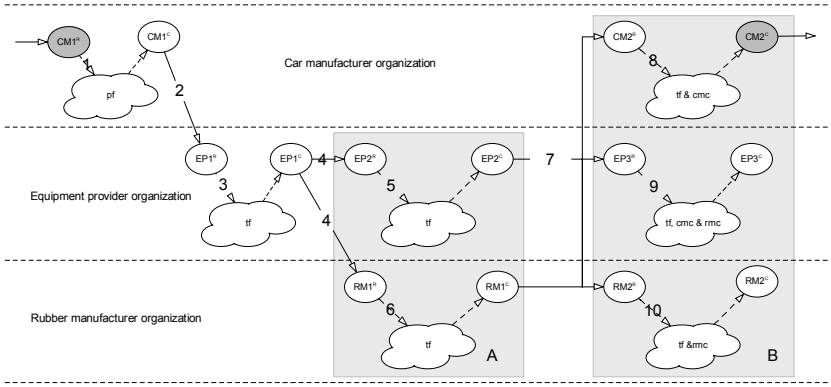

Fig. 8. A first representation

We recognize the pattern depicted in figure 6 and we deduct the $\underline{CM2 \infty EP3 \infty RM2}_{tf}$ synchronization activity. For *cmc* and *rmc* relations, we recognize the figure 5 pattern where *EP3* is involved in two different synchronizations activities while *CM2* and *RM2* run independently. We obtain $\underline{CM2 \infty EP3}_{cmc}$ and $\underline{EP3 \infty RM2}_{rmc}$ synchronization activities. That is, we distinguish four synchronization points:

$11)(EP2 \infty RM1)^{R} \Rightarrow Modifies(tf)$

$12)(CM2 \infty EP3 \infty RM2)^{R} \Rightarrow Modifies(tf)$

$13)(EP3 \infty RM2)^{R} \Rightarrow Modifies(rmc)$

$14)(EP3 \infty CM2)^{R} \Rightarrow Modifies(cmc)$

We redirect process flow in new synchronization points and reconnect them to the initial flow[1]:

$15)tf^{p}_{EP2}.tf^{p}_{RM1} \Rightarrow Run(\underline{EP2 \infty RM1}_{tf})$

$16)(EP2 \infty RM1)^{C} \Rightarrow Complete(EP2.RM1)$

$17)cmc^{p}_{EP3}.cmc^{p}_{CM2} \Rightarrow Run(\underline{CM2 \infty EP3}_{cmc})$

$18)tf^{p}_{CM2}.tf^{p}_{EP3}.tf^{p}_{RM2} \Rightarrow Run(\underline{CM2 \infty EP3 \infty RM2}_{tf})$

$19)rmc^{p}_{EP3}.rmc^{p}_{RM2} \Rightarrow Run(\underline{EP3 \infty RM2}_{rmc})$

$20)(CM2 \infty EP3 \infty RM2)^{C} \Rightarrow Complete(CM2.EP3.RM2)$

$21)(EP3 \infty RM2)^{C} \Rightarrow Complete(EP3.RM2)$

$22)(EP3 \infty CM2)^{C} \Rightarrow Complete(EP3.CM2)$

[1] In extended version of this paper, the reader could find the proof of the WfMC compliance where "."(and) and "+"(or) operators are used in AND/OR-Split/Join operations.

At the end, we obtain a global script sorted for best reading, where we separate processes behavior (i.e. process management layer):

$2)\ CM1^{C} \Rightarrow Run(EP1)$

$4)\ EP1^{C} \Rightarrow Run(EP2.RM1)$

$15) tf^{p}_{EP2}.tf^{p}_{RM1} \Rightarrow Run(\frac{EP2 \infty RM1}{tf})$

$16)(EP2 \infty RM1)^{C} \Rightarrow Complete(EP2.RM1)$

$7)(EP2 \infty RM1)^{C} \Rightarrow Run(CM2.EP3.RM2)$

$17) cmc^{p}_{EP3}.cmc^{p}_{CM2} \Rightarrow Run(\frac{CM2 \infty EP3 \infty RM2}{cmc})$

$18) tf^{p}_{CM2}.tf^{p}_{EP3}.tf^{p}_{RM2} \Rightarrow Run(\frac{CM2 \infty EP3 \infty RM2}{tf})$

$19) rmc^{p}_{EP3}.rmc^{p}_{RM2} \Rightarrow Run(\frac{EP3 \infty RM2}{rmc})$

$20)(CM2 \infty EP3 \infty RM2)^{C} \Rightarrow Complete(CM2.EP3.RM2)$

$21)(EP3 \infty RM2)^{C} \Rightarrow Complete(EP3.RM2)$

$22)(EP3 \infty CM2)^{C} \Rightarrow Complete(EP3.CM2)$

And object's manipulations are described as follow (i.e. dedicated to Sat functions resolution methods):

$1)\ CM1^{R} \Rightarrow Modifies(pf)$

$3)\ EP1^{R} \Rightarrow Modifies(tf)$

$5)\ EP2^{R} \Rightarrow Modifies(tf)$

$6)\ RM1^{R} \Rightarrow Modifies(tf)$

$11)(EP2 \infty RM1)^{R} \Rightarrow Modifies(tf)$

$8)\ CM2^{R} \Rightarrow Modifies(cmc,tf)$

$9)\ EP3^{R} \Rightarrow Modifies(cmc,rmc,tf)$

$10)\ RM2^{R} \Rightarrow Modifies(rmc,tf)$

$12)(CM2 \infty EP3 \infty RM2)^{R} \Rightarrow Modifies(tf)$

$13)(EP3 \infty RM2)^{R} \Rightarrow Modifies(rmc)$

$14)(EP3 \infty CM2)^{R} \Rightarrow Modifies(cmc)$

Figure 9 represents the new process graph. It enlightens synchronization points, their relations with involved processes and objects they manipulate.

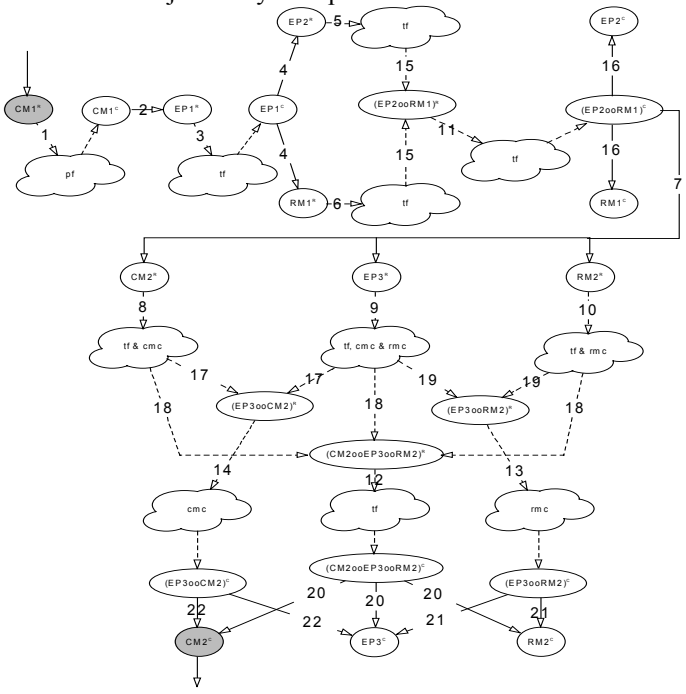

Fig. 9. Graphic's script representation

Moreover, we are allowed to modify the script if a new process or activity income. Each modification implies to revalidate all the script to avoid inconsistencies. This has a cost but we assume once again that it is the price to pay for a systemic consistency.

5 Implementation

We developed in the context of the AEE project [9] a prototype that tried to implement the first version of the model we describe [7], [8]. Remarks from our industrial partners were very fruitful and help us to design the new prototype we are currently developing. Let us now present the new architecture against the old one.

The former architecture of the prototype [12] was a centralized one, which is not really suitable when compared to our partner needs, particularly to preserve their autonomy. In fact, it is difficult to one organization to abandon information about processes, data or every piece of information involving its know-how. We also used the RDBMS Oracle 8i for managing all the data (projects, processes, activities, synchronization points, participants and documents). In order to fully exploiting XML capabilities of Oracle 8i, all these data were also accessible as XML documents.

The new version is based on a distributed architecture. Each partner hosts its part of the database and exchanges are done thanks to SOAP messages. Moreover, due to the level of abstraction we have introduced for describing organization processes, the new architecture does not require any specific way of doing things, such as describing precisely the organizational model or the process model in the system.

Then, this is not mandatory to have "workflow enabled" applications to be coupled with the architecture. This view is quite different from existing workflow systems such as CrossFlow [10] for instance, where organizational model should be replicated and where applications should be "workflow enabled" in order to be able to extract any necessary data the system may need. In our system, we use a database to store all the objects (projects, processes…) but we do not require an explicit enterprise model. In fact, we use late binding to couple one activity to respectively an application, a participant, a workflow engine or a back-end process. We use WSDL [17] for describing properties of process.

We have implemented a tier between two or several organizations. This tier can be viewed as a *broker* as it allows for storing projects, abstract descriptions of the processes, contracts and all the information about synchronization points. Contracts help us to filter what is the right information to provide to the right participant at the right time. A contract is a XML document that set up the exchange between two or more partners. It also describes what information they want to share with others. A contract also references a set of filters, which are XML documents used to describe what part of a document must be available to others. Once being processed, the original document is delivered to the synchronization point repository thanks to a SOAP message. When one organization wants to access shared data, it requests the tier which is in charge of maintaining which partner in the cooperation is able to deliver these data. Once it finds the owner, it gives to the provider the end-point of the requester and the associated filter. Then, a peer-to-peer communication is engaged between the requester and the provider. Of course, this communication is compliant to the existing contract between the two organizations. Figure 10 gives an overview of the architecture where we manage both control flow and data flow. Control flow is synchronized thanks to the synchronization activities belonging to synchronization points. This means that when an activity is achieved, its results are delivered to the synchronization point repository and that when the coordination is done, the next activity

starts if and only if all the checks are valid and the reconciling operations on shared objects are done.

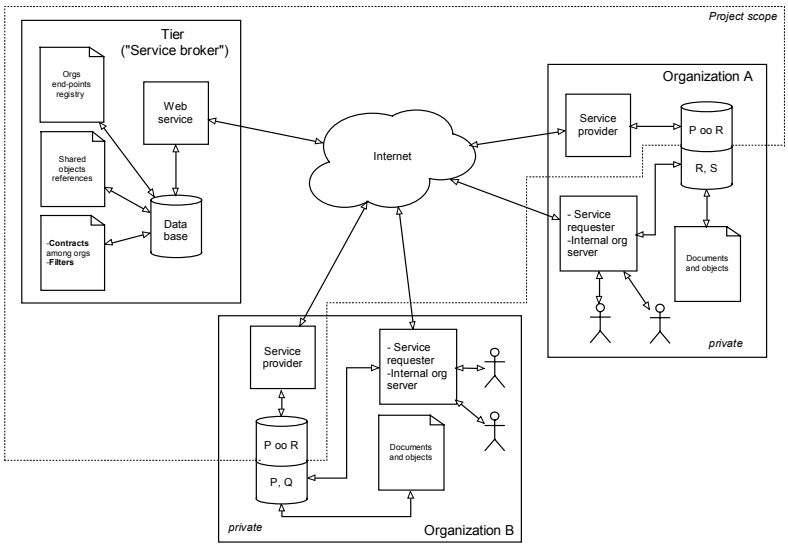

Fig. 10. Architecture diagram

Data flow is also managed. During the execution of a multi-partner cooperative activity, we manage all the different versions of one document, one objective or data. Version management is done thanks to a configuration and version management system named Toxic [15]. Nested version spaces are available as processes can be nested.

With this prototype, cooperation phases are more accurate due to the operators we described previously in the model. In fact, a cooperation phase starts when each partner decides to debut a cooperative activity with others in order to fulfill a given objective. Each partner describes its activities, and contracts are established to express conditions and terms of data exchange and share. Then, each partner works in autonomy, doing the job, but having the ability to deliver ongoing version of its work to others.

By providing a shared space for versions associated with the synchronization point, we are able to keep one participant of this synchronization point in touch with work progress of others. By having access to intermediate results, each participant can see what others did on shared objects. As objects are under version control, one can see previous revision, see differences between versions and who changed what and when. Of course, this asynchronous awareness can be considered as simple, but we believe that it can help to ensure that everyone has up to date information about how the project and the collaborative activities make progress. Moreover, each participant keeps all the control on his objects, a mandatory feature for all the partners.

Finally, one feature provided by the new prototype is the ability to trace the work that is done during both autonomy periods and collaborative periods. By storing main states of all the activities, duration of these activities, missed deadlines or decisions that are taken, we provide a view of what happened and when, and where the process failed or succeeded.

6 Related work

Component architecture (CORBA, EJB) currently provides middleware for integrating applications or components written in different languages. For interacting with others, a component need only to know the interfaces of the components it wants to interact with. These interfaces are described with a language, the IDL. Within these environments, applications are executed as ACID short-term transactions. The middleware provides several services: naming, transactions and resources allocation... A long-term application can be viewed as a sequence of independent steps manually or automatically activated (as in the WfMC model). Therefore they are not suitable for collaborative processes.

Current work in the research field on automating processes is not directly applicable in the virtual enterprises domain. In fact, there is no common shared middleware that could be used by several enterprises or that could fit in the needs for spreading across enterprises boundaries. Current propositions lead to tightly couple one organization with another, not only at an architecture level but also at the process level. Moreover, transactions models and coordination needs are not accurate. For instance, access to shared resources is very important, and locking these resources is not desirable as one organization could loss its autonomy. Then, recovery operations could not be under the responsibility of only one enterprise. To summarize, in virtual enterprises, there are two contradictory needs that are the autonomy of partners and the need to get some information about processes hold by the others. We will now present some work that is trying to answer this challenge.

Van Der Aalst [16] proposes a model that is compliant to the WfMC model. It is a global model that could be split into different parts, but there is no communication between the partners. The execution of this model is under the responsibility of a centralized architecture. However, the use of Petri nets for modeling renders possible the verification of the process consistency.

In Weske [21], there are some ideas to resolve flexibility in workflows management systems but no proposals for multi-enterprises processes. One interesting idea is the definition of a meta-schema for workflows and the use of graphs for defining workflows.

The work of Casati [2] describes data exchange and process interoperability, but in a B2B context, where exchanges are limited to peer-to-peer conversations. The concept of traces is very interesting and similar to our proposition.

Georgakopoulos [6] presents in CMI the concept of window of opportunity which allows for conciliating prescribed activities inherited from WfMS and optional activities inherited from groupware applications. Our synchronization point concept helps to provide the same kind of feature, but it provides a lot of more information and it also gives the opportunity to exchange intermediate results easily.

Some works around workflows are available. XPDL and Wf-XML [18] are used to define processes. BPQL from BPMI [1] or WSFL [11] can be used to query the states and control the execution of process instances. PIP could provide interchange between two or more organizations following Rosetta Net standard [13].

Then, propositions from HP (e-speak), OASIS (ebXML) or Microsoft (.NET) answer to some issues but fails to provide solutions for long term transactions, collaboration phases, or binding to internal processes.

7 Conclusion

In this paper, we have proposed a model and an architecture that support collaboration and cooperation for multi-enterprises processes. Furthermore, as several partners that have to realize a common objective need coordination means, we also support coordination. This work is clearly in between workflow management systems and groupware systems. In fact, it tries to offer the advantages of the both worlds. With the synchronization point model, different partners can work together, and they only need to define the cooperation rules (corresponding to contract specification). Then, the SP is able to coordinate the work progress, and it provides to all the participants accurate information on the work evolution. By including data and control flow management functions in the SP, we allow a flexible process definition - we are able to adjust the collaboration process definition by updating the SP during the work progresses.

The first feature of the SP model is the platform and system independence, thanks to a certain level of abstraction. Thus, it is easy to integrate the model with existing back-end systems of organizations and it is not tightly coupled with workflow-enabled systems as FlowMark could be for instance.

Another feature of the model is the adherence to component-based design and composability. With this model, it is possible to compose several processes viewed as components in order to obtain an overall composed process. This leads to the ability to easily manage multi-enterprises processes.

Flexibility, portability, and interoperability are also supported. These features are hardly feasible with current WfMS. For instance, it's a huge task to provide flexibility and interoperability between two WfMS, despite the existence of WAPI (unfortunately, WAPI is not supported by all the WfMS). Then, when you need to cooperate in order to achieve a given objective, you need to exchange and to provide to others intermediate results, breaking the ACID properties of classical transactional models that are used by these WfMS. The synchronization point model tries to breaks these limits.

We also support one important feature as the SP model offers an effortless enterprise information systems integration, while preserving autonomy of each partner. We do not impose to be compliant to any model (both process or organizational models) in order to use our model and our architecture. This is hardly possible when we try to do this using WfMS.

Then, we adopt a service-oriented architecture, and we rely on standards such as SOAP, WSDL and XML. This allows for accommodating and fitting both the architecture and the model without to throw away all the work ever done and to benefit from future enhancements.

Future work aims at analyzing the execution, as it could be done in workflow mining.

8 References

[1] BPMI: Business Process Management Initiative, www.bpmi.org

[2] Casati, F., Sayal, M., Dayal, U., and Shan, M.C.: Integrating Workflow Management Systems with Business-to-Business Interaction Standards, ICDE 2002, February 2002

[3] Chen, Q., Dayal, U., Hsu, M., and Griss, M.: Dynamic Agents, Workflow and XML for E-Commerce Automation, First International Conference on E-Commerce and Web-Technology (EC'2000), UK, 2000

[4] Dan, A., and Parr, F.: Long running application models and cooperating monitors. HPTS workshop, Asilomar, CA, 1999

[5] Dayal, U., Hsu, M., and Ladin, R.: Business Process Coordination: State of the Art, Trends, and Open Issues, VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy. Morgan Kaufmann 2001, ISBN 1-55860-804-4
[6] Georgakopoulos, D.: Collaboration Management Infrastructure (CMI): advancements in Coordination, Awareness, and Integration, November 2001
[7] Godart, C., Perrin, O., and Skaf, H.: COO: a workflow operator to improve cooperation modeling in virtual enterprises, in 9th IEEE RIDE, International Workshop on Research Issues in Data Engineering, Sydney, March 1999
[8] Godart, C., Perrin, O., and Skaf-Molli, H.: Cooperative Workflows to Coordinate Cooperative Asynchronous Applications in a simple Way. International Conference on Parallel and Distributed Systems (ICPADS '2000), July 2000
[9] Godart, C., and Perrin, O.: AEE process: inter-organizational process model v. 1.5 and AEE inter-organizational communication model v. 1.1. AEE project, December 2001
[10] Grefen, P., Aberer, K., Hoffner, Y., and Ludwig, H.: CrossFlow: Cross-Organizational Workflow Management in Dynamic Virtual Enterprises; International Journal of Computer Systems Science & Engineering, Vol. 15, No. 5, 2000; pp. 277-290
[11] Leyman, F.: Web Services Flow Language 1.0, IBM, May 2001
[12] Perrin, O., and Godart, C.: A Mail Based and XML based Protocol To Support Workflow Interoperability. AI 2002, February 2002
[13] Rosetta Net: PIP, Partner Interchange Process, Rosetta Net Implementation Framework, www.rosettanet.org
[14] Schuster, H., Georgakopoulos, D., Cichocki, A., and Baker, D.: Modeling and Composing Service-Based and Reference Process-Based Multi-enterprise Processes, CAiSE 2000, LNCS 1789, 2000
[15] Toxic, ECOO group
[16] Van Der Aalst, W., and Weske, M.: The P2P Approach to Inter-organizational Workflows, CAiSE 2001, and LNCS 2068, 2001
[17] Web Services Description Language (WSDL) 1.1, W3C, March 2001
[18] WfMC, Interface 1: Process Definition Interchange, Process Model, WfMC TC-1016-P, Version 1.1, October 1999
[19] WfMC, Interface 2: Workflow Client Application Programming Interface (WAPI) Specification, WFMC-TC-1009, Version 2.0, July 1998
[20] WfMC, Terminology & Glossary, WfMC-TC-1011, Version 3.0, February 1999
[21] Weske, M.: Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System, Proceedings of the 34th Hawaii International Conference on System Sciences, 2001