Contract-Driven Cross-Organizational Business Processes

Ustun Yildiz, Olivera Marjanovic, Claude Godart

Contract Driven Cross-Organizational Business Processes

Ustun Yildiz[1], Olivera Marjanovic[2] and Claude Godart[1]

[1]LORIA-INRIA
615, rue du jardin botanique
Villers-lès-Nancy, 54600, FRANCE
{yildiz, godart}@loria.fr

[2]The University of New South Wales
Sydney, NSW 2052, AUSTRALIA
o.marjanovic@unsw.edu.au

Abstract

Managing the processes of autonomous business organizations while they collaborate is a key requirement in order to guarantee that every business partner sees benefits of the collaboration. Thanks to the advances in the middleware IT technologies, some of the most important limitations of cross-organizational collaborations have been resolved. Yet, there are still a number of problems related to the distributed and heteregenous nature of the cross-organizational business environment. The monitoring and coordination, or briefly the management, of cross-organizational processes is one of the important missing aspect of the domain. In this paper, we describe a contract driven cross-organizational process monitoring and coordination mechanism. The contract based solution goes one step further comparing to inflexible and isolationistc approaches of workflows. For modeling the cross-organizational collaboration, we propose a generic contract model and next, we introduce an event-driven infrastructure for the enactment and monitoring of contract related processes.

Keywords: Business Process Managemenent, Virtual Enterprise Modeling, Cross-organizational Collaboration, E-contracting, Complex Event Processing.

I.INTRODUCTION

Recently, Business Process Management (BPM) domain is gaining considerable attention from researchers and solution provider industrials. The idea of BPM is to provide a computer based solution to the management of entities such as persons, activities, systems that are involved in the fulfillment of a predefined goal(s). BPM tools have been in the market since the early nineties. The market of BPM estimates at $214 million in 2002 and is expected to be double in 2008 (Wintergreen, 2003). The advances in the middleware IT technology make the construction of cross-organizational processes easier than in the past. In a cross-organizational process, the intra-organizational processes of business partners are interconnected in order to satisfy the mutual benefits of their owners. These kinds of processes give an organization the possibility to use the capabilities of others that it does not dispose. However, the cross-organizational collaborations suffer of the limitation of two contradictory requirements: The first is the necessity of a rigorous coordination and monitoring mechanism among collaborating partners. The second is the necessity of flexible dependencies between intra and cross-organizational processes that will enable the protection of the partner's privacy (Orlowska et al. 2005).

Workflow is an old and powerful formal concept that is widespread used for the modeling and implementation of business processes. Although a workflow is an efficient way to model processes, it is not fully adequate for non-sequential cross-organizational collaborations. First, it tightly couples the business activities with their implementation (Schuster et al. 2000), (Ellis 2003). Within the cross-organizational context, this limitation prevents the autonomous sub-process owner to choose possible alternatives for the enactment of the same goal and obliges it to couple its internal processes with its partner's. The second important limitation is that long duration collaborations can be based on on-demand linkages that are expressed with mutual commitments which can not be modeled with coercive, isolationistic, and inflexible approach of workflows.

Besides the initial construction of a cross-organizational collaboration, another requirement is the checking of the consistency between the run-time and initially planned behavior of business partners. The process instance is expected to be consistent with its model (Rinderle et al. 2005). This requires a

background process that monitors the collaboration. This need is crucial to insure that every partner sees benefits from the engaged collaboration. Due to the dynamic and unpredictable changes of the distributed environment, it is hard to prescribe all of the possible exceptions that can occur. But somehow they must be handled in order to allow the collaboration to continue processing. However, the monitoring and exception handling are complex tasks in workflow management systems (Hagen et al. 2000). By the point of view of the business partners, we can resume their requirements about the cross-organizational collaboration as follows: (i) the efficient management of complex interactions among involved business entities, (ii) the conformity of partners to their agreed behavior, (iii) ensuring a flexible mechanism that does not couple their intra-organizational processes with the collaboration.

In this paper, we propose a contract based approach to enable the cross-organizational collaboration of autonomous partners. The contract based solutions stand as an alternative to the workflow based solutions and in nowadays, they are receiving increasing attention from several researchers (Berry et al. 2005), (Xu et al. 2004), (Zdravkovic et al. 2004), (Farrell et al. 2005). Like in the traditional business collaborations, the contract prescribes the constraints that the business partners are supposed to respect during their collaboration. Contrary to workflows and workflow-like conceptions, the collaboration depicted in the contract is flexible and leaves much room for its implementation by intra-organizational processes and enables long duration collaborations. Assuming that the contract adopts an open-policy (this means that the absence of a permission to execute an action is not taken to be the same as presence of a prohibition to execute it), the monitoring of the contract with related processes is a considerable solution to ensure the required monitoring. In the rest of this paper, we introduce a business contract model that is used to model the complex interactions of cross-organizational partners. Besides the contract model, we propose an event-based infrastructure for the instrumentation of the environment that will interconnect the partners and check the conformity of their behavior. The use of event driven approaches for process interconnection and monitoring is not new, due its generic and finer-grained nature; it offers a considerable support to model the run-time features of dynamic applications (Luckham, 2002).

The remainder of this paper is organized as follows, the next section illustrates a case study that describes a real life collaboration of two autonomous business partners. In the section 3, we propose a business contract model to define the collaboration. The section 4 explains how the contract and received events are handled in order to monitor the behavior business partners and coordinate their interactions. The section 5 reviews the similar works while the last section concludes.

II.ILLUSTRATIVE EXAMPLE

As example, we present a cross-organizational collaboration scenario used in the CrossFlow Project (CrossFlow 2000). The scenario is derived from the real life co-operation between two companies. In this paper, we extended this scenario with new requirements. The first autonomous organization *(Provider)* is a logistic provider company that disposes a global network that provides domestic and international delivery services. The second organization *(Customer)* is an on-line store company such as Amazon.com that uses the delivery services of the *Provider* for the orders of its clients. Each company has its own Workflow Management System *(WfMS)*. In order to satisfy their engagements with their business partners, each company has several business policies that govern its intra-organizational processes. For example, the *Provider* aims to improve the quality of services it provides to its customers such as the flexible delivery support that allows the customer to change the details of its order, reducing costs etc. As the long duration collaboration of *Provider* and *Customer* is based on on-demand linkages, their cross-organizational relationships between them are defined by a business contract which is independent of their specific internal enactment mechanism. The contract includes several features such as the declaration, legal contacts, or URI of involved parties. In the contract, we particularly interested in the issues of the contract that define the relationships of the *Provider* and *Costumer.* Below, we explain informally the scenario and contract that we use in the rest of the paper. The aim of *Customer* is to manage the orders of its clients that use its web portal. It may initiate the cross-organizational process placing a delivery order to *Provider*. In the delivery order, *Customer* defines the characteristics of the item and its destination. As soon as the *Customer* announces a delivery service is needed, this information is distributed to all parties involved of the *Provider* by the *Provider* itself. *Provider* pick-up unit picks up the item to be delivered from *Customer*, sticks a bar-code and then sends the item to its closest warehouse. The warehouse is a node in the network of the *Provider*, it tranfers the item to another warehouse closer to its destination and proceeds the delivery. The *WfMS* of *Customer* and *Provider* are interconnected on the top of a contract repository structure that tracks the execution of the contract. So, both organizations have the same view of the contract related features before the beginning of the execution. From the beginning of this process, new information can be available. *Costumer* may change the delivery address before the delivery starts from the warehouse of the *Provider*. *Customer* is only allowed to place an additional delivery order if the resulting total amount of its unpaid orders is not above a certain limit. Any item must be delivered to its destination within a precise time if

Customer does not change its delivery address. As defined in the rules, *Customer* can influence the internal processes of *Provider*. The influence is restricted by the current state other contract statements. For example, *Customer* can start another delivery if it does not violate the rule that restricts the amount of its total unpaid orders. The role of the contract is not limited to restrict the operation of its parties; it is also an intermediary for the coordination. Let assume that the item to be delivered is in the warehouse of *Provider*, as long as the item stands in the warehouse, the *Customer* holds the flexibility to change the delivery address. Actually, in this situation, an unpredicted wait time in the warehouse which depends on an internal problem of *Provider* can be transformed a situation that *Customer* takes advantage of it. In the contract model that we present in the next section, we aim to model the rules that restrict the behavior of partners and regulate their interactions. These rules gather contract entities and define their relationships. Besides the relationships included in a single rule, the rules are interconnected by special relations that expresses the contract as a whole.

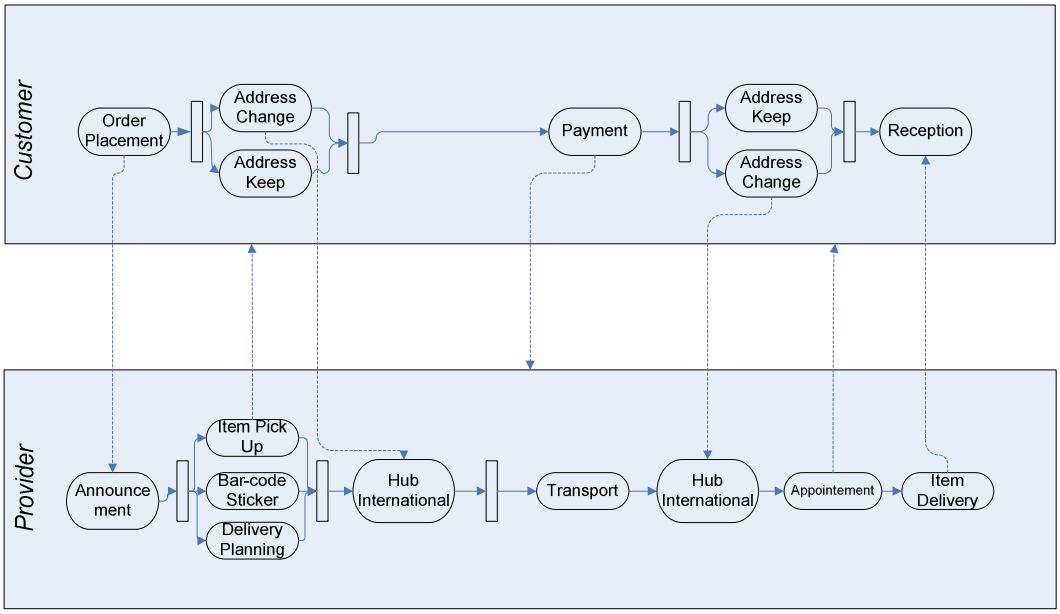

Figure 1: The Cross-Organizational Scenario

Figure 1 illustrates partially the private workflows of *Provider* and *Customer*. The dashed arrows are the interactions that can happen or must happen between them. The contract defines these interactions and the conditional, causal and temporal statements that are necessary for their existence.

III.BUSINESS CONTRACT MODEL

In this section, we define a business contract model that will be used in the rest of the paper. The contract model gathers the key elements of business processes and restricts their run-time behavior. As we mentioned in the previous section, although the contract governs the internal business processes of contract partners and the cross-organizational process, its specification and operation are different comparing to Workflow concept which is a useful way to represent business processes. The contract defines in advance the business constraints that the business partners are supposed to respect while they collaborate. Our focus is on formal representation of the contract constraints. The constraints are the key features that should be dealt for process enactment and monitoring purposes. To be able to model formally the business contract and its monitoring infrastructure, it is necessary to have a conceptual tool that be used for: (i) the expression of the complex relationships among contract entities, (ii) the conception of run-time behavior of the processes with the contract (iii) the monitoring of cross-organizational collaboration via the contract. For these reasons, we have chosen the Complex Event Processing (CEP) technology presented in (Luckham 2002), (Shanahan 1999) and essentially in (Adi et al. 2004). The CEP is conceived as a middleware support for the formalization of event-driven dynamic applications. It allows the definition of events that consist of the relevant occurrences in the observed environment and their aggregation to have semantically interested situations. We use the concept of CEP to model the contract content and then to monitor related processes. Familiarity with CEP can help the understanding of its relation with monitoring and our contribution. However, this paper is self-contained and contains all of the necessary definitions.

We consider the contract as a set of ***Constraint***s that have properties and inter-relationships. A single constraint gathers atomic elements of the contract. The atomic elements of the contract are ***Parties***, ***Operation***s, ***Metric***s, ***Date***s and ***Object***s. The contract ***Parties*** perform ***Operations***. The performed

operations operate on ***Object***s and ***Metric***s. A ***Metric*** is a numerical fact whose value can change during the collaboration. The temporal issues are modeled by ***Date***s.

Constraint Definition

In this section, we give the formal definition of a contract ***Constraint***. The definition includes the atomic elements of the contract, their relationships and also the relationships among the constraints in order to express the contract as a whole.

Definition 1 (Constraint): *A Constraint is defined by a tuple (id, executer, operands [], operators [], lifespan,) where:*

- *id ∈ ID, ID is the set of identifiers*
- *executer ∈ Parties is the contract party that is supposed to execute or respect the activities depicted in the constraint,*
- *operands [] is the list of atomic contract entities that are mentioned in the constraint such as operations, metrics, objects etc (see section …),*
- *operators[] is the list of operators that are used to aggregate the operands and restrict their behavior(see section),*
- *lifespan defines the temporal context in which the constraint is relevant (see Definition 1.1),*

Definition 1.1 (Constraint lifespan): *The lifespan of a constraint is defined by a tuple (initiator, terminator, initiationType, correlationType, terminationType) where:*

- *initiator defines the fact(s) that initiate the temporal context during which the restriction of the constraint is relevant,*
- *termination defines the fact(s) that ends the validity of the initiated constraint ,*
- *initiationType ∈ {"basic", "dependent", "complex"},*
- *correlationType ∈ {"ignore", "add", "add(k)"| k is a condition},*
- *terminationType ⊂ {"basic", "dependent", "complex", "discard", "all"},*

The proper modeling of the temporal context of a constraint is a very critical issue. The lifespan of a constraint defines the temporal context and conditions that the purpose of the constraint becomes relevant. The lifespan begins with the occurrence of an *initiator* and ends with the occurrence of a *terminator*. The *initiator* can be a simple predicate that begins to hold or a reception of a request or notification message. In these cases the *initiationType* of the constraint is "basic". The constraints that have "dependent" *initiationType* are activated according to the state changes of other constraints. For example, a penalty constraint that has the purpose to compensate the damage of another not respected constraint, is activated when the state of latter becomes violated. A constraint can be initiated by several "basic" initiators or by the state changes of another initiated constraint, then its *initiationType* is "complex". Another important point of the constraint lifespan is its repetition over the deployment. A constraint can be re-initiated if the same initiators happen while there is at least one initiated constraint instance. The *correlationType* of a constraint defines whether the constraint can be re-initiated while there is another initiated instance of it. If the *correlationType* is "ignore" the constraint will have a single instance. If the *correlationType* is "add" then another constraint instance can be created when the same initiators occur, finally the "add(k)" type defines the conditions that another constraint can be initiated. For example, each time the *Customer* places an order, the constraints that impose *Provider* to process the delivery are initiated.

The *terminationType* of a constraint defines the termination mode of the relevance of it. The constraints that have "basic", "dependent" and "complex" *terminationType*s end with the occurrence of terminators, the state changes related to another constraint or both of them. The relevance of the constraint can end while its own state changes during its lifespan. For example, if a constraint that depicts a relationship that must hold during its lifespan, begins to not hold then its state changes. This kind of termination is "discard". For the cases that a constraint has "add" *inititiatonType* and "all" *termiantionType*, if there are more than one instance of the same constraint, the relevance of the initiated constraints end if one of them ends.

Operators and Operands

To represent the temporal context and dependencies of a constraint, we used separate components to reduce the complexity. *Operators* and *Operands* are key features that are used to express the restriction purposes of the constraints. In order to model them formally, we use two techniques borrowed from CEP and Deontic Logic (Lee, 1988). The *Operators* designate structures that are used to express syntactic, algebraic and behavioral restrictions on their *Operands*. The *Operands* are the atomic elements of the contract but also their properties can be subjects of restriction.

The Operators that we consider are classified into three categories. The first is *Behavioral Operators* taken from the Deontic Logic. The Deontic Logic studies the logical relationships of entities asserting that certain of them are *Obligations*, *Permissions*, and *Prohibitions*. The operation of these constraints is

similar to their dictionary meanings: Obligations describe the facts that must happen or hold, Permissions characterize the facts that can occur, and Prohibitions are used for the facts that must not occur. The second type of operator is *Value Operators* which consist of arithmetical operators that restrict the numeric values of their operands. The last type of operator we consider are *Syntactic Operators* . They are used to express aggregation relationships among the restricted entities. In the following table we present the operators that we use in this paper.

Oper ID	Operator	Type	Operands	Semantic
D1	***P***	*Deontic*	*All*	*Permission*
D2	***O***	*Deontic*	*All*	*Obligation*
D3	***F***	*Deontic*	*All*	*Prohibition*
V1	=	*Value*	*Metric*	*Comparison*
V2	<	*Value*	*Metric*	*Comparison*
V3	>	*Value*	*Metric*	*Comparison*
V4	≠	*Value*	*Metric*	*Comparison*
S1	¬	*Syntactic*	*All*	*Negation*
S2	*Sequence*	*Syntactic*	*Operations*	*Aggregation*
S3	*All*	*Syntactic*	*Operations*	*Aggregation*
S4	*Atmost*	*Syntactic*	*Operations*	*Aggregation*
S5	*Each*	*Syntactic*	*Operations*	*Aggregation*
S6	∩	*Syntactic*	*All*	*And*
S7	∪	*Syntactic*	*All*	*Or*

Table 1. Operators and Operands

Table 1 makes the classification of the Operators and their Operands.

III. ENSURING PROCESS ENACTMENT

Besides the expression of the content of the contract, another important feature is its view and sharing by involved partners. Like in traditional business affaires, we assume that the contract is a set of rules that have an internal coherence and it is agreed by business partners before the beginning of their collaboration. So, every business partner has the same view of the contract as a reference document. By definition, the content of the contract can not change during the collaboration. A change can be considered if all of the involved parties agree about the change. We consider the contract as an entity that is taken as a reference by business partners in order to regulate the individual behavior. The related individual processes and the monitoring mechanism can be implemented differently by involved parties. There can be a trusted third party that observes the actual states of contract constraints or the partners may prefer to set up their own monitoring mechanism that makes the same task. In any case, there will be a different interpretation of the same instance for different partners. For example, *Customer* has to pay the delivery costs before the end of the delivery. The *Customer* will see the state of this constraint as an operation that it must execute and the *Provider* will see the same constraint instance as an expected operation of *Customer*. In order to set up a mechanism that will characterize the state of contract related processes and monitor their execution, there is a necessity for additional data structures. In this section, we introduce an enriched data model for executed contract content. The model we propose is similar to process data that are usedby workflows, but it is modified for contract needs.

Constraint Instance

A constraint instance is a run-time representation of a contract constraint. It has a state for every instant during the enactment. According to *correlationType* and *initiators* of the constraint, the new instance of the same constraint can be created during the enactment. As the partners will have different interpretation of the same constraint, the constraint instance will have a different semantics for each of them.

Definition 2 (Constraint Instance): *A constraint instance is defined by a tuple (instance_id, mode, current_state, inactive_state, initiation_date, termination_date) where:*

- *instance_id* $\in$ *ID, ID is the set of identifiers,*

- *number* ∈ ℵ *is the counter that illustrates the number active instance of the constraint,*
- *currentState* ∈ *{"initial", "active"," executed"," violated"," terminated" }*
- *mode* ∈ *{"request-response", "sollicit-reponse"}*
- *initiationDate* ∈ *Date*
- *terminationDate* ∈ *Date*

The properties of a constraint instance can change after its initiation. When a constraint is initiated, its *currentState* becomes *active*. An active constraint instance requires to be monitored by its executer and also the partners that invoked it or by the partners related to the constraint. Depending on the purpose of the constraint, the state of the rule can be violated, terminated or executed. If the purpose of a constraint is expressed by a permission modality, the permission can be executed while the rule is *active* or the party can prefere to not execute its permission. If the permission is executed than the state of the constraint instance becomes *executed* and contrary if the permission is not executed the states becomes *terminated* with the occurrence of a terminator. If the deontic modality of the rule is an obligation, the executer can fulfill the depicted obligation than the state of the constraint becomes *executed* or it can not fulfill the obligation than the state becomes *violated*. For the constraints that include prohibition operators, the instance can pass to the *violated* state or to *terminated* state. At each state transition a *transitionEvent* procuded by the contract monitor. The *transitionEvents* can be the *initiator* or *terminator* of other contract constraints. At the moment, a constraint instance is initiated, it has a *initiationDate* and at its state transition, it has a *terminationDate.* Besides these conventional properties, a contraint instance has a mode property. The mode of a constraint instance characterizes its semantic for the entity that holds it. If the mode of a constraint instance is "request-response" than the constraint instance concerns its holder. This means that the activated constraint instance -according to its deontic- should be processed by its holder. If the mode of the constraint is "sollicite-response", this indicates that the constraint must be respected or can be executed by a party besides the holder of this instance. So, the instance holder party should respect "request-response" instance, managing its internal processes, and it has to observe the state of the "sollicite-response" constraints to check if its partners behave as expected.

Business Events

We use the event paradigm to model the dynamic infrastructure of the execution enviroment. The events correspond to the relevant occurrences in the environment. The events can come from different sources. The first is the contract repository which is a shared space between contract parties. The second event source is the entities that are involved to the collaboration but they are not stated in the contract. For example, a banque that holds the accounts of *Provider* can produce events to *Provider* even if it is not a party of the contract depicted business. To separate event sources has a major advantage, it enables the cross-validation of constraints instances. For example, *Customer* can declare that it made the payment of the order. This event will come from contract repository as the initiation and termination of constraints are made by contract partners. But at the same time, the *Provider* can check the banque, if the banque produces an event saying the payment is not received. It gives the *Provider* to make a cross-validation of the sollicite-response constraint instance in which the *Customer* has an obligation. In the framework, the events coming from different sources and contract repository are gathered in the event middleware and forwarded to contracting parties. The contract repository produce also events while the states of the constraint instances change. These events are forwarded to contract parties to update their constraint instances. When the environement is modeled with events, we need to find way to illustrate every relevant occurrence using events. In order to proceed this mapping operation, we propose the events as below:

- *Constraint instance events:* These events are implicit events produced by constraint instance holders in order to initiate or terminate the constraint instances,
- *Metric events:* These events are produced when the value of an event changes,
- *Objects events:* Accordingly to the presence of an object, object events are produced, they describe whether an object exists or not,
- *Operation events:*Operation events characterize the occurrence of an operation, it can be the reception or the execution of the operation,

Due to the lack of space we do not detail the content of each event. Each event contains a number attributes that defines its properties such as occurrence time, detection time, the parties or fact that caused it or the data that are transferred with this event. These information enable the mapping between the events and the relevant constraint instances. Thus, the evolution of the execution can be compared to the run-time constraints.

Putting The Enactment into Practice

In the following we explain how the proposed methodology is applied to the example we presented in the previous section. Figure 2 illustrates how the constraint instances are initiated and terminated according to the behavior of the partners. The frame 1 is initiated with the order placement, the *correlationType* of this constraint is *add* with the condition which is associated with another constraint: the total amount of unpaid orders is limited. The purpose of frame 1 is to illustrate the obligation of Provider to deliver the item before a precise deadline dynamically defined when the order is placed. For the constraint which prescribes the amount, another frame is initiated in order to collect the amount of unpaid orders. We do not show this frame in the figure, its initiator is the first placed order and its *correlationType* is ignore as it will terminate when its is violated. When the product is received from *Customer*, it is permitted to change the delivery address of its order until it is taken from its first international hub. This permission is expressed by the frame 3. The initiator of this frame is the placement of the order which is defined by *Customer* and the terminator is defined by *Provider* when it takes the item out of the international hub. While the order is processed by *Provider*, *Customer* has to pay the charges of the order. It can pay the charges before the prescribed deadline which is calculated after the order placement. This obligation is initiated with the order placement; its *terminationType* is "discard" as it will finish with the payment. This obligation and similar "discard" type constraints have also an expiration date until which the constraint should be satisfied. The frame 5 depicts another constraint instance which is initiated when the item comes to its destination's hub, *Customer* can change its delivery address again if the changed address is accessible from the hub. This constraint is terminated when *Provider* contacts the final customer that purchased the item for the final delivery. Because of the violated constraints, such as a late delivery, new constraints can be initiated in order to compensate the effects of the former. The initiations and terminations of constraint instances are driven by the events we presented in the previous section. For example, the constraints that have dependent *intitiationType* are initiated by the *Constrain instance events* that occur when the state of the instance changes.

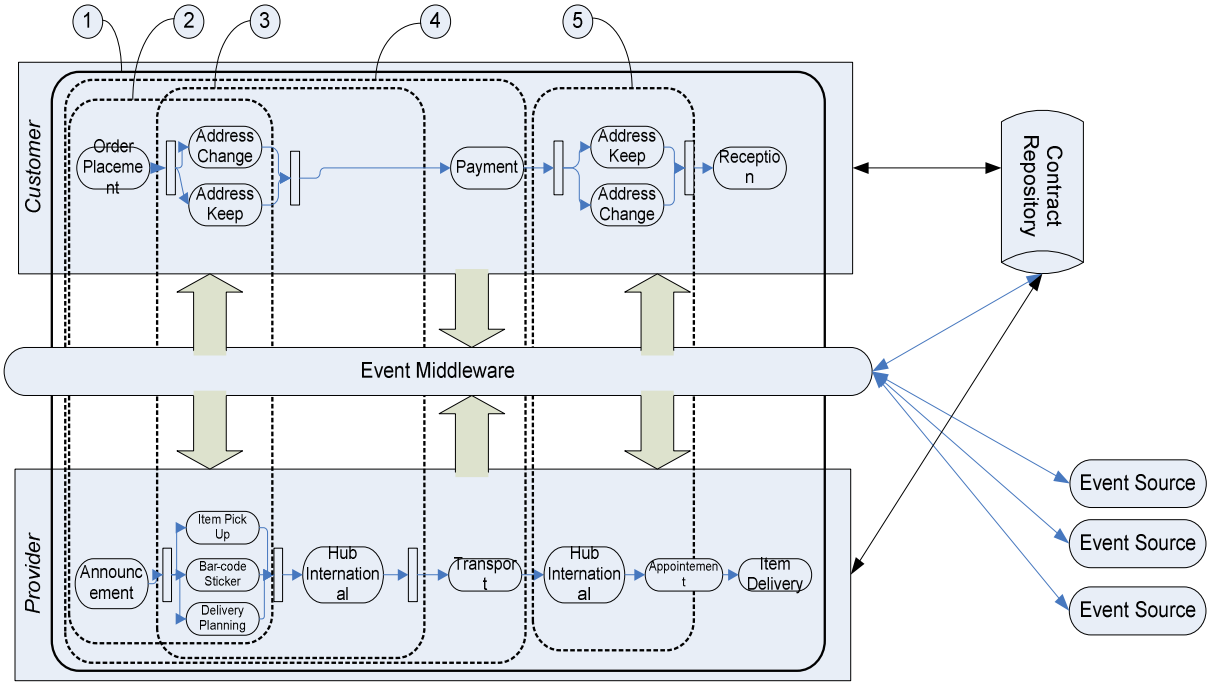

Figure 2. The environment modeled by events and constraint instances

For the constraints that depict a restriction on a shared metric, event notification mechanisms are associated to each event sources and the instance of these constraints are evaluated with the data carried with the associated event.

IV. RELATED WORK

The monitoring and coordination of collaborations of autonomous organization is a hot topic in the recent research literature (Orlowska et al. 2005) (Aalst et al. 2005). For this purpose, the use of business contracts is not new. (Xu et al. 2004) proposes a conceptual multi-party contract and a mechanism to monitor the execution of the contract statements. The proposed model is based on commitments graphs among contract parties which do not provide a flexible collaboration required in the contract based collaborations. The author is concerned by the temporal dependencies among business actions, the model does not take into account all of the process related features. The Business Contract Language (BCL)(Milosevic et al. 1995) is one the closest works to ours, the model is flexible but it does not provide generic monitoring facilities. It obliges each monitor to develop its own monitoring mechanism.

The most known use of contracts is Service Level Agreements (SLA) in the utility computing. These contracts are used to specify and manage the underlying network activities of service provider and service consumer. They define minimum obligations and expectations between participating actors. These types of contracts are conceived for network activities and they can not be an efficient support for the expression of the high-level business objectives of partners and their complex interactions.

Business process flow languages (e.g. BPEL4WS) provide a language and a structure for describing a business process and how to define assertions. However, BPEL focuses on non functional or procedural contract and does not provide any explicit support for the business contract level of abstraction. In fact, these approaches are interesting but are interested in recovery and failures support rather than prevention support.

V. CONCLUSION AND FUTURE WORK

The work presented in this paper is motivated by two major requirements of cross-organizational processes. The first is the necessity of flexibility for the protection of privacy and the second is the coordination and monitoring. We proposed a formal contract model to express the cross-organizational process enacted by autonomous organizations. We used an event based infrastructure to instrument the interactions between contract parties and relevant changes in the business environment. The use of non-sequential contracts is a considerable solution to construct flexible collaborations in which collaborating partners can protect their privacy. As the contract is the key piece of governance in the collaborations, the expression of its run-time state is a considerable and trusted solution for the monitoring. However, in the dynamic, heterogeneous and untrusted environments of the business market, the problems can be dealt when they are detected by observable and objective manners.

Service Oriented Architectures (SOA) architectures, especially Web Services, are emerging as middleware to implement cross-organizational processes. The very recent advances in this domain such as WS-Coordination, WS-Agreement, WS-Policy are actually agreements among involved Web Services. These technologies aim to specify the protocols supported by collaborating services such as the order in which service functionalities can be invoked or how the message exchanges should happen while they interact. Actually, to base the collaboration of autonomous actors on a set of constraints is to define a business contract that regulates their interaction. We aim to extend and implement our work for the composition of Web Services.

REFERENCES

(Wintergreen, 2003) Wintergreen. Business Process Management (BPM) market opportunities strategies, and forecasts, 2003 to 2008, Lexington: WinterGreen Research, (2003).

(Schuster et al. 2000)H. Schuster, D. Georgakopoulos, A. Cichocki and D. Baker, Modeling and composing service-based and reference process-based multi-enterprise processes. In: Proceedings of the 12th Conference on Advanced Information Systems Engineering (CAiSE), LNCS 1789, Springer Verlag, Stockholm, Sweden, pp. 247–263, (2000).

(Cross-flow, 2000) CrossFlow, Cross-Organizational Workflow Support in Virtual Enterprises, ESPRIT Project 28635. http://www.crossflow.org

(Orlowska et al. 2005) Maria E. Orlowska and Shazia Sadiq, Collaborative business process technologies, Data & Knowledge Engineering, Volume 52(1), (2005).

(Aalst et al. 2005) Proceedings of the 3rd International Conference of Business Process Management, LNCS 3649. Wil M.P. van der Aalst, Boualem Benatallah, Fabio Casati and Francisco Curbera editors, (2005).

(Ellis, 2003) C. Ellis, Net models supporting human and humane behaviors, Invited talk, in: Conference on Business Process Management (BPM), Eindhoven, The Netherlands, (2003).

(Rinderle et al. 2005) Stefanie Rinderle, Manfred Reichert, Peter Dadam: Correctness criteria for dynamic changes in workflow systems - a survey. Data Knowl. Eng. 50(1): 9-34 (2004).

(Hagen et al. 2000) Claus Hagen, Gustavo Alonso: Exception Handling in Workflow Management Systems. IEEE Trans. Software Eng. 26(10): 943-958 (2000).

(Berry et al. 2005) Andrew Berry, Zoran Milosevic, Extending choreography with business contract constraints, International Journal of Cooperative Information Systems (IJCIS), Volume 14, Number 2-3, 131-179, (2005).

(Milosevic et al. 1995) Zoran Milosevic, Andrew Berry, Andy Bond, Kerry Raymond, An Architecture for Supporting Business Contracts in Open Distributed Systems, the 2nd International Workshop on Services in Distributed and Networked Environments (SDNE'95), Whistler, Canada, (1995).

(Xu et al. 2004) Lai Xu, Manfred A. Jeusfeld: Detecting Violators of Multi-party Contracts. In CoopIS/DOA/ODBASE (1) 2004: 526-543, (2004).

(Zdravkovic et al. 2004) Jelena Zdravkovic, Paul Johannesson, Cooperation of Processes through Message Level Agreement. Proceedings of Advanced Information Systems Engineering, 16th International Conference, CAiSE 2004, Riga, Latvia, 564-579, (2004).

(Farrell et al. 2005) Andrew D. H. Farrell, Marek J. Sergot, Mathias Sallé, Claudio Bartolini, Using the event calculus for tracking the normative state of contracts, International Journal of Cooperative Information Systems (IJCIS), Volume 14, Number 2-3, 99-129, (2005).

(Luckham 2002) David Luckham. The Power of Events. Addison Wesley, (2002).

(Shanahan 1999) Shanahan M. "The event calculus explained", In Artificial Intelligence Today, LNCS: 1600, 409-430, Springer, (1999).

(Adi et al. 2004) Asaf Adi and Opher Etzion. Amit - the situation manager. VLDB J., 13(2):177–203, (2004).

(Lee, 1988) Lee R. Towards open electronic contracting. Electronic Markets, Vol. 8, No. 3, 10/98, (1988).