A Contract-Based Approach for Monitoring Collaborative Web Services Using Commitments in the Event Calculus

Mohsen Rouached, Olivier Perrin, and Claude Godart

LORIA-INRIA-UMR 7503,
BP 239, F-54506 Vandœuvre-lès-Nancy Cedex, France
{mohsen.rouached, olivier.perrin, claude.godart}@loria.fr

Abstract. Web services (WS) are gaining popularity for supporting business interactions in cross-organisational distributed business processes. However, current WS specifications mostly concentrate on syntactic aspects. Because multiparty collaborations in business involve complex and long-lived interactions between autonomous partners, their behaviour must be specified to ensure the reliability of the collaboration.

This paper presents an event-based framework associated with a semantic definition of the commitments expressed in the event calculus, to model and monitor multi-party contracts. This framework permits to coordinate and regulate Web services in business collaborations, by allowing detection of actual and imminent violations.

Keyword: Service Monitoring, Collaboration and Coordination, Event Calculus, Commitments.

1 Introduction

Web services are gaining popularity for supporting reusable business processes across distributed and heterogeneous environments. They are well suited to support cross-organisational interactions because such interactions require tighter communication between organizations while preserving their individual processes and practices as an element of their competiveness. However, Web services specifications mostly concentrate on lower levels and do not offer high-level abstractions to accomodate variations in service invocations and business process models.

On the other hand, business contracts seems to be an interesting technology for inter-organisational collaborations, and as such, they are increasingly taking a central role in the context of virtual entreprises. They provide a flexible way to define a protocol, which when formally defined, can be analyzed and checked. Moreover, contracts can be encoded outside the organizations, and this is interesting as most of the organizations demands are oriented towards more transparency, autonomy, and preservation of the corporate knowledge. Therefore, for reliability and efficiency, a contract could be mapped onto a number of

M. Kitsuregawa et al. (Eds.): WISE 2005, LNCS 3806, pp. 426–434, 2005.

more formalized modeling concepts which can be used to facilitate its integration with cross-organisational business processes and other enterprise systems. This includes contract monitoring features and dynamic updates to the processes and policies associated with contracts.

This paper presents our contract-based approach which makes use of business contracts for regulating Web services to support the cross-organisational nature of collaborations and to integrate contract management services into the overall business processes between organizations. It consists of:

- an event-based framework to model and monitor multi-party contracts, which permits to organizations to fulfill their needs of externalisation and autonomy,
- a commitment-based formalisation of contract clauses since commitments are now widely recognized as a satisfying representation for the interaction in multi-partners business processes,
- and a representation of commitments in terms of the Event Calculus [KS86] predicates, as Event Calculus is a formal language with well defined semantics providing an efficient mean for temporal abstractions.

The paper is organized as follows. Section 2 describes our event-based model, and shows how to express contract events in the Event Calculus. In section 3, we present the event-based monitoring for e-contracts. We explain our strategy for expressing contract clauses in terms of commitments in the Event Calculus. Section 4 discusses the related work. In section 5, we give a running example. Finally, section 6 concludes the paper and presents some future directions.

2 An Event-Based Model for e-Contracts

Using events for modeling contracts appears to be interesting, and this interest comes mostly from their effects in terms of changes that are produced by the execution operations. Indeed, specifying and detecting events play an important role in the process of analyzing, monitoring, and visualizing the behaviour of each party involved in the e-contract. Relevant events need to be recorded. After conflicts between contractual parties, these records can be used as evidence to determine what happened and to specificy the responsibilities.

2.1 Specifying Composite Events

In order to provide a computational model for events, we use the Event Calculus [KS86] which is a calculus that allows for specifying some state at particular time-points for time-varying properties (called fluents). The concepts used are inspired from the first order logic, and we find four types of objects:

- $\mathcal{A}$(variables $a, a_i, \ldots$): events, or actions,
- $\mathcal{F}$(variables $f, f_i, \ldots$): fluents, the value of fluents is time-dependent,
- $\mathcal{T}$(variables $t, t_i, \ldots$): timepoints,
- $\mathcal{X}$(variables $x, x_i, \ldots$): domain objects.

The Event Calculus also defines five basic predicates:

(1) $Happens(e,t)$: event e occurs at timepoint t,
(2) $HoldsAt(f,t)$: fluent f is true at timepoint t,
(3) $Initiates(e,f,t)$: if e occurs at t, then f is true and not released at $t+1$,
(4) $Terminates(e,f,t)$: if e occurs at t, then f is false and not released at $t+1$,
(5) $t1 < t2$: this is the standard order relation for time, $t1$ precedes $t2$.

Then, the last two axioms are *Clipped* and *Declipped*, and their definitions are: $Clipped(t1,f,t2) = \exists(a,t)[Happens(a,t) \wedge (t1 <= t < t2) \wedge Terminates(a,f,t)]$. This means that fluent f is no more true (terminated) between times $t1$ and $t2$. Respectively, we have: $Declipped(t1,f,t2) = \exists(a,t)[Happens(a,t) \wedge (t1 <= t < t2) \wedge Initiates(a,f,t)]$. This means that fluent f is true (initiated) between times $t1$ and $t2$.

With these definitions, it is easy to treat concurrent actions, as different *Happens* can refer to the same timepoint. Given this property, we can deduce that two events (with their corresponding timepoints) can be totally ordered based on the ordering of their timepoints (and predicate 5).

In our event-based mechanism, a timepoint denoted t_s of a distributed composite event e is the supremum of the set of timepoints of the constituent primitive events collected when the composite event occurs.

Temporal Orders. Let $e1$ and $e2$ be any primitive events. The temporal order of these two events is defined as follows:

1. $Happens(e1,t1)$ is said to be happen *before* $Happens(e2,t2)$ if $t1 < t2$.
2. $Happens(e1,t1)$ is said to be *concurrent* with $Happens(e2,t2)$ if $t1 = t2$.
3. $Happens(e1,t1)$ is said to be happen *after* $Happens(e2,t2)$ if $t1 > t2$.

Disjunction. The meaning of disjunction is that as soon as one of the two events occurs, the disjunctive event occurs. Disjunction of two events $e1$ and $e2$ is denoted $disj(e1,e2)$. Formally:

$$disj(e1,e2)(t) = Initiates(e_2,raised,t) \Longleftarrow Happens(e_1,t)$$
$$\vee Initiates(e_1,raised,t) \Longleftarrow Happens(e_2,t)$$

Conjunction. The meaning of a conjunction is that two events must both occur before the conjunctive event occurs, but that the order of occurrence, and any overlap of occurrence, is not relevant. Conjunction of two events $e1$ and $e2$ is denoted $conj(e1,e2)$. Formally:

$$conj(e1,e2)(t) = Initiates(e_3,raised,t) \Longleftarrow$$
$$Happens(e_1,t_1) \wedge Happens(e_2,t_2) \wedge t \geq sup(\{t_1,t_2\})$$

Sequence. Sequence is said to be strict, i.e. one event must have occurred before the next event in the sequence. Sequence of two events $e1$ and $e2$ is denoted $seq(e1,e2)$, and is defined as follows:

$$seq(e1, e2)(t) = Happens(e_2, t_2) \Longleftarrow Happens(e_1, t_1) \wedge (t_2 > t_1)$$

It is possible that after the occurrence of $e1$, $e2$ does not occur at all. To avoid this situation, we must appropriately use definite events, such as absolute temporal event or the end of the activities execution.

Negation. The meaning of a negation is that an event $e1$ does not occur in a closed interval formed by $e2$ and $e3$. It is denoted by $neg(e1, e2, e3)$. Formally:

$$neg(e1, e2, e3)(t) = \forall t_2 \in [t_1, t_3],$$
$$Happens(e_1, t_1) \wedge Happens(e_3, t_3) \wedge \neg Happens(e_2, t_2)$$

Temporal Iteration. A periodic event is a temporal event that occurs periodically. It is denoted by $P(e1, d, e2)$ where $e1$ and $e2$ are arbitrary events and d is a time slot. $e1$ occurs for every d in the interval $]e1, e2]$. Formally:

$$P(e1, d, e2)(t) = \exists t_1, (\forall t_2 \in [t_1, t], t =$$
$$t_1 + i * d \; for \; some \; i)(Happens(e1, t_1) \wedge \neg Happens(e2, t_2)$$

If the constraint "$e1$ occurs only once $e2$ occurs" exists, the previous definition becomes:

$$P(e1, d, e2)(t) = \exists t_1, (t > t_1)(Happens(e1, t_1) \wedge (Happens(e2, t))$$

Of course, it is possible to combine different operators. For instance, $disj(e1, seq(e2, e3))$ represents a composite event which occurs as a result of the disjunction of $e1$ and the sequence of $e2$ and $e3$.

3 Event-Based Monitoring for e-Contracts

To determine whether an execution of a contract is correct, we must represent not only the behavior of the different parties but also the evolution of the contractual relationships among them. These contractual relationships are naturally represented through commitments which permit to capture the obligations of one party to another. The time factor is an important element in the representation of a commitment, and to formally express a commitment, it is necessary to find a representation able to handle temporal constraints. A richer representation of the temporal content of commitments will make them more suitable for representing real situations of business contracts which commonly involve many clauses and have subtle time periods of reference. The representation of temporal properties for commitments usually use some branch of the Temporal Logic [Tl90]. However, as we are interested in analyzing and checking multiparty contracts, the monitoring mechanism needs a high level of externalisation and temporal abstraction. That is why we rely on the use of the Event Calculus introduced so far. Thus, in our monitoring model, we view each action in the contract as an operation on commitments. Then, we develop an approach for

formally representing and reasoning about commitments in the Event Calculus. Therefore, we can specify the content of the contract throught commitments of each party. Contract parties create commitments and manipulate them as a result of performing actions through the contract clauses. Further, by allowing preconditions to be associated with the initiation and termination of contract activities, different commitments can be associated with these activities to model the interactions among parties. Conceptually, these interactions are governed by the rules mentioned in the contract clauses.

In our approach, we are using three types of commitments as defined in [Sin99]: (1) a *base-level commitment* denoted $C(p1, p2, c)$ which stipulates that party $p1$ becomes responsible to party $p2$ for statisfying condition c, i.e, c holds sometime in the future,(2) a *conditional commitment* denoted $C_c(p1, p2, c1, c2)$ stipulates that if the condition $c1$ is satisfied, $p1$ will be committed to bring about condition $c2$, and (3) a *persistent commitment* denoted $C_p(p1, p2, A(c))$ is defined as a commitment from party $p1$ to party $p2$ to ensure that condition c holds on all future time points (operator $A(c)$).

Then, deontic clauses can be defined in terms of operations on commitments. The main step consists of specifying deontic constraints including specification of roles and their permissions, obligations and prohibitions using both commitments axioms and Event Calculs axioms [Sha97]. Thus, prohibitions (F), permissions (P), and obligations (O) can be rewitten as follows:

Prohibitions: a prohibition is used to express that an action is forbidden to happen for a party. If a party is prohibited to bring out a proposition p, then it has a commitment to ensure that p never holds.

$$Create(F(p1, p2, p), p2, C(p2, p1, A(\neg p)))$$

Permissions: by considering permissions as negations of prohibitions, we obtain that a party $p1$ is permitted to bring about a proposition p if it has not been prohibited from it. Permission is given by party $p2$.

$$Release(P(p1, p2, p), p1, C(p2, p1, A(\neg p)))$$

Obligations: an obligation is a prescription that a particular behaviour is required. It is fulfilled by the occurrence of the prescribed behaviour. To express that a proposition p is compulsory for a party, we use the permission's expression presented so far. Among deontic logic rules defined in [MW93], we find the following rule that establishes a relationship between permission (P) and obligation (O):

$$P(p) \longleftrightarrow \neg(O(\neg p))$$

From this rule we deduce the following relation

$$O(p) \longleftrightarrow \neg(P(\neg p))$$

As such, using commitments and event calculus, obligations are defined as:

$$Create(O(p1, p2, p), p2, C(p2, p1, A(p)))$$

Given these formal specifications of contract clauses, we can specify the content of business contracts through the partners commitments. Indeed, the specification of a contract is seen as a set of *Initiates* and *Terminates* clauses that define which activities pertaining to the contract are initiated and terminated by each party. Then, the contract execution consists of performing a set of actions that will take place at specific timepoints, that is, a set of *Happens* clauses along with an ordering of the timepoints specified in the contrat rules.

4 Running Example

As a running example, we use a business contract established for buying and selling books on the Internet. Its idea was inspired from the Netbill protocol introduced by Sirbu in [Sir98]. The contract's execution begins with a customer (BR) requesting a quote for some desired books, followed by the purchaser (BP) sending the quote. If the customer accepts the quote, then the purchaser delivers just the book's abstract and waits for an electronic payment order (EPO). After receiving the EPO, the purchaser forwards the EPO and the book to an intermediation server, which handles the funds transfer. When the funds transfer completes, the intermediation server sends a receipt back to the purchaser. As the last step, the purchaser forwards the receipt to the customer to communicate the reception date. Here, we focus on a contract that involves the customer and the purchaser and we try to formalise its operations using commitments in the event calculus. First, we define the fluents used in this contract protocol.

- **Fluents**
  - request(i): a fluent meaning that the customer has requested a quote for item i.
  - books(i): a fluent meaning that the purchaser has delivered the book i.
  - pay(m): a fluent meaning that the customer has paid the agreed upon amount m.
  - receipt(i): a fluent meaning that the purchaser has delivered the receipt for item i.
- **Commitments**
  - $C_c(CT, MR, books(i), pay(m))$ means that the customer is willing to pay if he receives the books.
  - $promiseBooks(i, m) = C_c(MR, CT, accept(i, m), books(i))$ means that the purchaser is willing to send the goods if the customer promises to pay the agreed amount.
  - $promiseReceipt(i, m) = C_c(MR, CT, pay(m), receipt(i))$ means that the purchaser is willing to send the receipt if the customer pays the agreed-upon amount.

After identifying fluents and commitments necessary for the contract's execution, we look now at how the commitments among contract parties evolve during this phase.

1. When the books are sent at time $t1$, the fluent $books(i)$ is initiated. Further, by $Create(sendBooks(i,m), BP, promiseReceipt(i,m))$, the commitment $C_c(BP, BR, pay(m), receipt(i))$ is created. So now the books have been delivered, and the purchaser is willing to send the receipt if the customer pays. Formally, we have:
   - $HoldsAt(books(i), t1)$ since we have $Initiates(sendBooks(i,m), books(i), t)$.
   - $HoldsAt(C_c(GP, GR, pay(m), receipt(i)), t2)$ after executing $Create(sendBooks(i,m), GP, promiseReceipt(i,m))$. (1)
2. By sending the EPO at time $t2 > t1$, the customer initiates the fluent $pay(m)$ at time $t3$. This ends the commitment $C_c(BP, BR, pay(m), receipt(i))$ and creates the commitment $C(BP, BR, receipt(i))$. Since no event occurred to terminate $books(i)$, it continues to hold. Formally, we have:
   - $HoldsAt(pay(m), t3)$ since $Initiates(sendEPO(i,m), pay(m), t)$ was initiated. (2)
   - $HoldsAt(C(BP, BR, receipt(i)), t3)$ by using (1) and (2).
   - $\neg\, HoldsAt(C_c(BP, BR, pay(m), receipt(i)), t2)$ by using (1) and (2). (3)
3. At time $t3 > t2$ the happens clause $Happens(sendReceipt(i), t3)$ is applicable, which initiates the fluent $receipt(i)$. This discharges the commitment $C(BP, BR, receipt(i))$ and then it will be terminated. Thus, we reach the stage where the purchaser has delivered the books and the receipt, and the customer has paid.
   - $HoldsAt(receipt(i), t4)$ by using $Initiates(sendReceipt(i,m), receipt(i), t)$
   - $\neg HoldsAt(C(BP, BR, receipt(i)), t4)$ by using (3).
4. Thus, at time $t4 > t3$, the following holds: $HoldsAt(books(i), t4) \wedge HoldsAt(pay(m), t4) \wedge HoldsAt(receipt(i), t4)$.

As such, given a business contract composed of obligations, permissions, and prohibitions, we have formalized these prescriptions using commitments. Then, by exploiting the strengths of the event calculus to specify business interactions rigorously, we showed how these commitments enabled Web service interactions will produce more flexible and reliable business process models.

5 Related Work

Because of the autonomy and decentralization of the participants, specifying and managing Web services interactions can be challenging. Conventional techniques fall into one of two extremes, being either too rigid or too unstructured. Our contract-based approach using commitments takes the middle path, emphasizing the coherence desired from the activities of autonomous decentralized entities, but allowing the entities to change their services in a controlled manner, which enables them to achieve progress in a dynamic and unpredictable environment. Below we discuss some approaches that are related to our work.

Traditionally, business contracts have been specified using formalisms such as finite state machines, or Petri Nets, that only capture the legal orderings

of actions. However, since the semantic content of the actions is not captured, the participants can not handle unexpected situations at runtime. To remedy this, our approach relies on the use of commitments and the Event Calculus. Commitments have been studied before [Cas95] but have not been used for business contract specification. One of the problems with the use of commitments is ensuring that Web services would not be able to retract them with no consequences. Sandholm et al. [SL01] describe various mechanisms for decommitting that would prevent prevent agents (in our case Web services) from decommitting at will. On the other side, the Event Calculus has been theoretically studied, but has not been used for modeling commitments or commitment-based specification. Denecker et al. [DMB92] use the Event Calculus for specifying process protocols using domain propositions to denote the meanings of actions. Then, contracts in distributed environments should respect the partners autonomy and enable them to interact flexibly to handle exceptions and contract breaches.

Paper [FSSB04] studied the automated performance monitoring of contracts, in terms of tracking contract state. In order to facilitate state tracking, the authors define an XML formalization of the Event Calculus, ecXML. This language is used to describe how a contract state evolves, according to events that are described in the contract. However, the contract monitoring and the composite event specification were not dressed.

Flores and Kremer [FK02] develop a model to specify conversation protocols using conversation policies. As in our approach, Flores and Kremer model actions as creating and discharging commitments. However, they only model base-level commitments, whereas our approach accommodates conditional commitments and reasoning rules for these commitments.

A common pattern of the related works discussed above is that all of them are in the direction of the system administration, but not business process automation and Web services regulation. The different domains have different requirements for monitoring. In our work, the Web services regulation requires semantic level monitoring, rather than system level monitoring. To achieve this goal, our e-contract monitorability relies on a purely event-based paradigm allowing a complete separation of the coordination aspects and functionality aspects which was not expressed in the works listed so far.

6 Conclusion and Future Work

This paper presents an approach to regulate and monitor collaborative Web services. We have described our solution to the problem of integrating contracts as part of cross-organisational collaborations. This solution consists of an event-based framework, associated with a semantic definition of the commitments expressed in the Event Calculus, to model and to manage multi-party contracts. First, we have developed a methodology to specify and detect composite events in busines contracts. Second, we have detailed a method to express deontic contract clauses using commitments in the Event Calculus. Finally, the related work was discussed.

In our future work, we plan to test our solution in existing Web services architectures such as ebXML or RosettaNet. This would help us to determine the expressive power of the model and its acceptability by contract domain experts and practitioners. Another alternative is to study correctness requirements in business process protocols, and then express them in terms of commitments and Event Calculus predicates.

References

[Cas95] C. Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proceedings of the International Conference on Multiagent Systems*, pages 41–48, 1995.

[DMB92] M. Denecker, L. Missiaen, and M. Bruynooghe. Temporal reasoning with abductive event calculus. In *Proceedings of the 10th European Conference and Symposium on Logic Programming (ECAI)*, pages 384–388, 1992.

[FK02] R. A. Flores and R. C. Kremer. To commit or not to commit: Modeling agent conversations for action. *Computational Intelligence 18(2)*, pages 120–173, 2002.

[FSSB04] A. D. H. Farrell, M. Sergot, M. Salle, and C. Bartolini. Using the event calculus for the performance monitoring of service-level agreements for utility computing, WEC 2004.

[KS86] R. Kowalski and M. J. Sergot. A logic-based calculus of events. *New generation Computing 4(1)*, pages 67–95, 1986.

[MW93] J. Ch. Meyer and R. J. Wieringa. *Deontic Logic in Computer Science: Normative Systems Specification*, chapter Deontic Logic: A concise Overview. John Wiley and Sons, 1993.

[Sha97] M. Shanahan. Solving the frame problem: A mathematical investigation of the commen sence law of inertia. In *Cambridge: MIT Press*. 1997.

[Sin99] M. P Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. In *Artificial Intelligence and Law 7*, pages 97–113. 1999.

[Sir98] M. A. Sirbu. Credits and debits on the internet. In *Huhns and Singh,1998*, pages 299–305. 1998. Reprinted from IEEE Spectrum, 1997.

[SL01] T. Sandholm and V. Lesser. Leveled commitment contracts and strategic breach. *Games and Economic Behavior*, 35:212–270, 2001.

[Tl90] Temporal and Modal logic. Temporal and modal logic. In *Theoretical Computer Science. Amsterdam: North-Holland*, pages 995–1072. 1990.