Proactive Contract Management through RSF Specifications

Rossella Aiello and Giancarlo Nota

Department of Mathematics and Computer Science
University of Salerno
84084 - Fisciano (SA) Italy

Abstract. The modelling and automation of e-contracts is an active research area that aims at providing a valid support to organizations for the definition and management of contractual relations. The approach adopted in this paper allows the modelling and monitoring of contracts specified in terms of RSF (Requirement Specification Formalism) rules. Starting from the planning of events relied to contract clauses established during the negotiation phase, we define a set of RSF rules that can be used as patterns for the monitoring of both: contract fulfillments and contract violations with respect to obligations, permissions and prohibitions. We also extend the semantics of the RSF language in order to allow the treatment of planned events, together with occurred and not occurred events, in a single transition rule. This enriched semantics supports the proactive behaviour of a contract management system enabling the immediate notification of fulfillments and non-compliances as well as the detection of imminent contract violations.

1 Introduction

The need to manage end-to-end processes in a efficient and effective way urges private and public companies to adopt ICT solutions based on e-services and SOA architectures. This trend can be in particular observed in the field of automated contract management where the collaboration is usually regulated by negotiated agreements subscribed between the involved parties.
An important aspect of contract management systems concerns the capability to monitor contract clauses, especially when end-to-end processes are managed using system intensive workflows that handle millions of transactions per day without human intervention.
In the last years, the research activity has been directed towards the definition of contract languages whose capabilities can provide support to obtain benefits such as: ambiguity avoidance, reduced disputes and increased performance of processes arising from contract agreements.
Some formal approaches use courteous logic [1] or Petri nets [2, 3]. In [4] a specification for deontic constraints focusing the attention on temporal constraints is defined together with a methodology for the verification of consistency. A formalization of deadline concept based on an extension of CTL can be found in [5, 6].

Many works based on Deontic Logic [7–10] handle the problem of Contrary-to-duty obligations and paradoxes. In [11, 12], the contract from natural language is first analysed and represented in a logical form using Deontic and Defeasible Logic. Then, the logical form is translated into a machine readable rule notation, based on RuleML, and implemented as executable semantics. This formalisation is used as a source of the mapping to the key policy concepts of a contract specification language called Business Contract Language (BCL) [13, 14].
The notion of commitment is the main concept used in [15, 16] to specify multi-party contracts. Commitment violations and a proactive detection of imminent contract violations are handled by means of an algorithm working on a data structure called commitment graph. In [17] a three layers architecture for e-contract enforcement is discussed. A metamodel for contract comprises several types of clauses as well as contract templates. In the second level, event-condition-action (ECA) rules are used to define business rules for the enforcement activities while the third level provides a web services implementation for the e-contract enforcement.
This work addresses the formal specification and the monitoring of contract clauses by means of RSF transition rules. We first propose a characterization of the standard deontic concepts of obligations, permissions and prohibitions oriented to the real time monitoring of contracts. Next, we introduce a set of RSF rules usable as patterns for the monitoring of clause fulfillments and violations. Finally, we extend the semantics of the RSF language in order to support the proactive behaviour of a contract management system; such extension enables the management of immediate notifications of fulfillments and non-compliances as well as the detection of imminent contract violations. The contribution of the paper is twofold: from one side we propose a set of rules that can be used as specification patterns for the monitoring of contracts. On the other side, we address the problem of the unified treatment of fulfillments, violations and proactive management that has received little coverage in the literature. This unifying approach becomes possible on the basis of the enriched version of RSF introduced in the paper.

2 The Requirement Specification Formalism

RSF is a specification language created with the aim of describing reactive discrete-event systems [18, 19]. In RSF, a system can be described writing a set of *transition rules* that allow the system to evolve through a sequence of states. The state of the system is composed of a set of events in a particular time instant.
An *event* can be represented by a triple $< p, v, t >$ in which p is a position, v a datum and t a timestamp meaning that the datum v is produced at the time t in the position q. A *position* is a place where data can be found in the system while a *timestamp* is a numeric value for time interval measured starting from a given origin in a given time unit. An RSF state assumes the following form:

$$s =< \{< p_1, v_1, t_1 >, ..., < p_m, v_m, t_m >\}, st >$$

where $\{< p_1, v_1, t_1 >, ..., < p_m, v_m, t_m >\}$ is the set of events in the state and st represents the *state proper time*, that is the instant of time at which the state refers. Note that timestamps of events can be less or greater than the state proper time. The intuitive

meaning of timestamps less than or equal to the state proper time, is that the events have already occurred in the past; while events with timestamps greater than the state proper time are events scheduled to occur in the future.
An *event set descriptor* ($eventDes$) can be defined as a set of events that satisfy the specified conditions $c_1, ..., c_k$.

$$eventDes =< p_1^{'}, v_1^{'}, t_1^{'} >, ..., < p_n^{'}, v_n^{'}, t_n^{'} > \textbf{with } c_1, ..., c_k.$$

These conditions can state both properties of the data $v_l^{'}, ..., v_n^{'}$ and time constraints about $t_l^{'}, ..., t_n^{'}$. We say that an $eventDes$ is satisfied by a state s when s contains n events such that:

– the n events match $< p_1^{'}, v_1^{'}, t_1^{'} > ... < p_n^{'}, v_n^{'}, t_n^{'} >$;
– the match yields an istantiation of variables which allow the logical conditions $c_1, ..c_k$ to hold.

We also define a state descriptor in the following way:

$$stateDes = eventDes_1 \textbf{ not-occur } eventDes_2.$$

A state descriptor is satisfied by a state s, when a subset of events satisfying $eventDes_1$ is present in s, while there is no subset of events satisfying $eventDes_2$.
The presence of specific data with suitable properties in a position at a given time instant may trigger an RSF transition rule that allows a system state transition. An RSF transition rule has the following form:

from $stateDes$
cons $eventDes_3$
produce $eventDes_4$

When a $stateDes$ of a rule R is satisfied in s, then the time instant of rule application is called application time.
The **from** part qualifies a state description and describes the properties that a state s must have in order to apply the transition rule. The **cons** part specifies the events which are consumed by the rule application; therefore, when an event is consumed, it is dropped from the state. The **produce** part lists the events created in the state as an effect of the rule application. Suppose that $eventDes_4$ has the form:

$$eventDes_4 =< p_1, v_1, d_1 >, ..., < p_k, v_k, d_k > \textbf{with } c_1, ..., c_l$$

where data $v_1, ..., v_k$ are produced respectively in the positions $p_1, ..., p_k$ with a production delay of $d_1, ..., d_k$ with respect to the rule application time. The event set states that the condition $c_1, ...c_l$ are used to assign values to the variables present in $< p_1, v_1, d_1 >$ $, ..., < p_k, v_k, d_k >$. A null delay means that the datum is produced with a negligible delay.
A particular behavior of the system as it evolves over time is described by a sequence of states $s_0, s_1, ...s_n$ called *history*. The transition rules determine which histories are effectively possible for the system (*viable histories*). Examples of transition rules are shown in sections 3 and 4.

In the following, we will use the Prolog notation for variables and constants: any word starting with an uppercase letter is assumed to be a variable while all constants start with a lowercase letter.

3 Deontic and Monitoring Constructs in RSF

3.1 Coding Deontic Constructs as RSF Events

Starting from the definitions above, it is possible to define the deontic concepts of obligation, permission and proibition, usually employed in the contract management literature [20, 21], in terms of RSF events. However, as the focus of this work is on the monitoring of contractual clauses, we consider several points of view, such as the planning of events regarding deontic concepts, the execution of activities and the monitoring of fulfillments and violations. The generic monitoring construct assumes the form:

<Role, [DeonticConcept, Action, Workproduct, Perspective, TimeReference], T>

where Action represents an action that Role must (can or cannot) perform according to the defined DeonticConcept, eventually producing a Workproduct. The variables Perspective and TimeReference are instantiated in agreement with the value of DeonticConcept, as shown in table 1.
We distinguish between *scheduled* obligation that are stated during the contract negotiation, *executed* obligation observed during the contract enactment phase and *fulfilled* obligation when an executed obligation meets the expectations of all involved roles.

Table 1. Instantiation of variables in the monitoring construct.

DeonticConcept	Perspective	TimeReference
Obligation	schedule	begin/end
	execution	begin/end
	fulfillment	-
	violation	-
Permission	grant	begin/end
	assertion	begin/end
	violation	-
Prohibition	stated	begin/end
	violation	-

When a permission is granted, the power to make something in a specific time interval is assigned to a specific role while, a prohibition statement establishes that a role is forbidden to perform a particular action. As an example of prohibition, suppose that, after the contract agreement, the supplier is forbidden to divulge every confidential information necessary for the contract accomplishment. According to the generic deontic construct presented above, the corresponding RSF event assumes the form:

<supplier, [prohibition, divulgeConfidentialInformation, _, stated, begin], 5>

Another example concerns the permission given to the supplier of requiring information about a given softwareProject within the time interval [10,30].

<supplier, [permission, requireInformation, softwareProject, grant, begin], 10>
<supplier, [permission, requireInformation, softwareProject, grant, end], 30>

3.2 Monitoring of Contractual Clauses

The contract specification expressed as a set of RSF rules states how the Contract Management System reacts to external stimuli provided by the environment in which the system operates. The set of stimuli from the environment, represented by means of RSF events $\{e_1, ...e_n\}$, are for the Contract Management System as the input data for a conventional program. When a transition rule is triggered the system evolves, and one or more events chosen from a set $\{e_1^{'}, ...e_m^{'}\}$ suitably defined may be produced toward the environment. The high level interaction between the Contract Management System and its environment is shown in fig. 1. The first rule to take into account is charged to receive

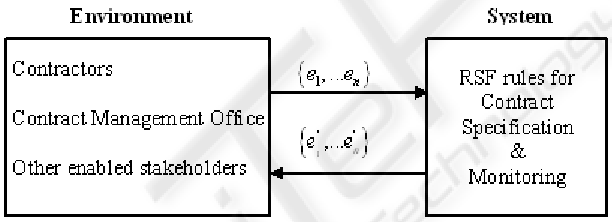

Environment
System
Contractors
Contract Management Office
Other enabled stakeholders
$\{e_1, ...e_n\}$
$\{e_1^{'}, ...e_m^{'}\}$
RSF rules for Contract Specification & Monitoring

Fig. 1. Interaction between the Environment and the System.

from the environment the events required to start the contract execution. The result of the rule application is a new state where permissions, obligations and prohibitions together with other scheduled events are planned as future events. For example, $R1$ is a simple rule that, starting from the signature of the contract, schedules an obligation that constraints the supplier to make the project in a specified period:

R1) Plans contract activities
from <signedContract,client,T_1 ><signedContract,supplier,T_2 >
cons signedContract
produce <signatories, [client,supplier],0>
        <supplier, [obligation, doProject, softwareSystem, schedule, begin], 5>
        <supplier, [obligation, doProject, softwareSystem, schedule, end], 180>

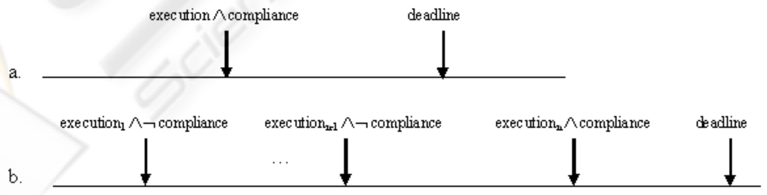
At application time, rule $R1$ produces an event <signatories,[client, supplier], 0> stating that client and supplier are considered signatories soon after both of them signed the contract. The events signedContract are consumed, while two other events are produced: the first defines that the supplier has to start the project 5 time units after the signature of the contract, while the second constraints the supplier to finish it within 180 time units, delivering a softwareSystem as a workproduct. A viable history is the following:

$s_0 =< \{\}, 0 >$ *initial state*
$s_1 =< \{$<signedContract,supplier,6> *from the environment*
<signedContract,client,6>$\}, 6 >$
$s_2 =< \{$<signatories,[client,supplier], 6>, *from the application of R1*
<supplier, [obligation, doProject, softwareSystem, schedule, begin], 11>
<supplier, [obligation, doProject, softwareSystem, schedule, end], 186>$\}, 6 >$

At time 0, the initial state s_0 is empty. At time 6, both client and supplier sign the contract; therefore, two events deriving from environment are inserted in the state s_1. In s_1 the state descriptor of R1 is satisfied, and the rule triggers producing a new state s_2 with three new events: the first records the information about the signatories and the others state the schedule of the activity “doProject”.
Most papers proposing formal approaches to contract management are focused on the violations of obligations in contracts [12, 5, 17, 21, 22]. Indeed, it is often useful to monitor clause violation as well as clause fulfillment [15].
As discussed in [23] the description of contractual exceptions, penalties and rewards is an important part of a contract. On the basis of given criteria (i.e. quality, advance delivering) a reward might be assigned.
The RSF specification that a contractual obligation has been correctly fulfilled requires the satisfaction of two conditions (fig. 2.a)):

1. A deadline (event-pattern:<Role, [obligation, A, Wp, schedule, end], T>) exists in the state of the system;

2. the corresponding $execution$ event happens before the scheduled deadline and the delivered workproduct is compliant with the expected characteristics.


execution ∧ compliance
deadline
a.
execution$_1$ ∧ ¬ compliance
execution$_{n-1}$ ∧ ¬ compliance
execution$_n$ ∧ compliance
deadline
b.
...
Fig. 2. Two possible scenarios of fulfilled obligations.

The state descriptor of rule $R2$ formally expresses these two conditions; when the rule can be applied, $R2$ consumes the event and produces an event that records the fulfillment of the obligation.

R2) Ex-post checking for fulfilled obligation
from <Role, [obligation, A_i, Wp_2, execution, end], T_1 >
<Role, [obligation, A_i, Wp_1, schedule, end], T_2 >
with $T_1 \leq T_2$, compliant(Wp_1, Wp_2)
cons <Role, [obligation, A_i, Wp_1, schedule, end], T_2 >
produce <Role, [obligation, A_i, Wp_1, fulfillment, _], 0>

Note that the check for fulfilled obligation is done at time T_2 with a delay of T_2-T_1 with respect to the delivery time of the workproduct Wp_1. A change of strategy, aiming at an immediate check of compliance between Wp_1 e Wp_2 requires a change in the semantics of RSF. This enhanced version of RSF is presented in section 4.
An obligation cannot be considered fulfilled if the obligation execution has not been accomplished before a defined deadline (fig. 3). Rule $R3$ cooperates with rule $R2$ to check violations.

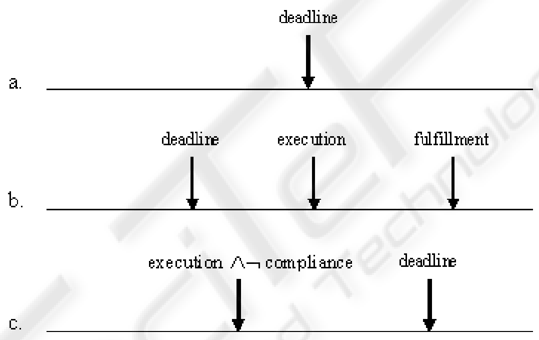


Fig. 3. Scenarios of violated obligation.

R3) Checking for violated obligation
from <Role, [obligation, A_i, Wp_1, schedule, end], T_1 >
not-occur <Role, [obligation,A_i,Wp_1, fulfillment, _],T>
produce <alert, [$Role_2$, Message], 0>

In this case, the rule application produces an alert, sending a Message to a receiver $Role_2$, that signals the violation of the obligation.
When an obligation is violated, some other obligations may be produced. This kind of obligation, known as Contrary-to-Duty, represents an enforcement that is activated when normative violations occur and are meant to "repair" violations of primary obligations [22]. The generic RSF rule for the Contrary-to-Duty assumes this form:

R4) Contrary-to-Duty
from $<$Role, [obligation, A$_i$, Wp, schedule, end],T$_1$ $>$
not-occur $<$Role, [obligation,A$_i$,Wp, fulfillment],T$>$
produce $<$Role,[obligation, enforcementAction, Wp, schedule, begin], T$_2$ $>$
$<$Role,[obligation, enforcementAction, Wp, schedule, end], T$_3$ $>$

As an example of contrary-to-duty application, the rule below handles the case when the client does not respect the scheduled payment date for the softwareSystem released, and must pay an addictional interest within 10 days:

from $<$supplier, [obligation, delivery, softwareSystem, fulfillment, _],T$_1$ $>$
$<$client, [obligation, payment, softwareSystem, schedule, end],T$_2$ $>$
not-occur $<$client, [obligation, payment, softwareSystem, fulfillment, _],T$>$
cons client
produce $<$client,[obligation, payWithInterest, softwareSystem, schedule, end], 10$>$

4 Proactive Monitoring

The approach specified by rule $R2$ for the ex-post checking of a fulfillment (done at the occurrence of a deadline) works fine for contracts that neither foresee reward criteria for early delivery nor pursue the optimization of processes arising from contract agreements. On the other side, when contracts are automatically monitored, the likelihood of violation decreases, as signatories can be alerted in advance of a real violation, and the failing partner can be forced to commit its obligations ahead of time ([15]).
In order to provide proactive monitoring capabilities, we extend the RSF transition rule in which the state descriptor assumes the form:

$$stateDes_p = \textbf{planned } eventDes_1 \textbf{ occur } eventDes_2 \textbf{ not-occur } eventDes_3$$

A state descriptor of a rule R is now satisfied by a state s when: 1) a subset of events scheduled for the future, with respect to the state proper time of s, satisfying $eventDes_1$ is present in s; 2) past events satysfing $eventDes_2$ are also present in s; 3) there is no subset of events satisfying $eventDes_3$.

R5) Early obligation fulfillment
from planned $<$Role, [obligation, A$_i$, Wp$_1$, schedule, end], T$_1$ $>$
occur $<$Role, [obligation, A$_i$, Wp$_2$, execution, end], T$_2$ $>$
with T$_2$ $\leq$ T$_1$, compliant(Wp$_1$, Wp$_2$)
cons $<$Role, [obligation, A$_i$, Wp$_1$, schedule, end], T$_1$ $>$
produce $<$Role, [obligation, A$_i$, Wp$_1$, fulfillment, _], 0$>$
$<$notice, [Role$_2$, Message], 0$>$

Rule $R6$ considers an obligation execution that happens before its deadline but still exists a gap between the provided and the expected workproduct (fig. 2.b e 3.c).

R6) Early monitoring of non-compliance
from planned $<$Role, [obligation, A_i, Wp_1, schedule, end],$T_1 >$
occur $<$Role, [obligation, A_i, Wp_2, execution, end],$T_2 >$
with $T_2 \leq T_1$, not(compliant(Wp_1, Wp_2))
cons $<$Role, [obligation, A_i, Wp_2, execution, end],$T_2 >$
produce <alert, [$Role_2$, Message], 0>

Sometimes the execution of proactive actions before imminent deadlines can be very useful to prevent the occurence of violations. Rule R7 checks the absence of a fulfillment before the deadline expires. In such a case, the first alert is sent when the system clock (represented by the keyword At) reaches the value $T_1 - T$ (a time interval before the deadline) to solicit the obligation execution. The rule iterates itself until the time list is empty or the obligation has been accomplished. For example, if a state s_i contains the event <plannedAlert, [Role, obliged, A_i, Wp_1, [30, 7, 1]], T>, the rule may send one or more alert messages with increasing priority.

R7) Imminent contract violation
from planned $<$Role, [obligation, A_i, Wp_1, schedule, end],$T_1 >$
occur <plannedAlert, [Role, obliged, A_i, Wp_1, [T|Tlist]], $T_2 >$ **with** At=$T_1 - T$
not-occur $<$Role, [obligation, A_i, Wp_2, fulfillment, _], $T_3 >$,
cons plannedAlert
produce <alert, [Role, Message, Priority], 0>
<plannedAlert, [Role, obligation, A_i, Wp_1, Tlist], 0> **with** Priority=1/T

5 Conclusions

Contract specification languages are receiving increasing attention in the last years due to their capability to obtain several benefits such as: ambiguity avoidance, reduced disputes and increased performance of processes arising from contract agreements. The paper proposes a characterization of the standard deontic concepts of obligations, permissions and prohibitions oriented to the real time and proactive monitoring of contract clauses together with a set of RSF rules usable as patterns for the monitoring of clause fulfillments and violations. Among the papers recently proposed in the literature our work has certain similarities with the approaches described in FCL [22], in [17] and in [15]. However, FCL does not explicitely provide a language construct handling the concept of non-occurrence of events that is a standard feature of RSF rules. ECA rules allow a limited treatment of planned events. This feature is very useful to define contract management systems capable of proactive behaviour. The proposed representation of deontic concepts in terms of RSF events allow the treatment of the notion of commitments discussed in [15]; furthermore, the monitoring RSF rules can handle fulfillments, violations and the proactive behaviour of a contract management system.

References

1. Grosof, B.N., Labrou, Y., Chan, H.Y.: A declarative approach to business rules in contracts: courteous logic programs in xml. In: EC '99: Proceedings of the 1st ACM conference on Electronic commerce, New York, NY, USA, ACM Press (1999) 68–77
2. Raskin, J., Tan, Y., van der Torre, L.: How to model normative behavior in petri nets (1996)
3. Lee, R.M.: Automated generation of electronic procedures: procedure constraint grammars. Decis. Support Syst. **33** (2002) 291–308
4. Marjanovic, O., Milosevic, Z.: Towards formal modeling of e-contracts. In: EDOC '01: Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing, Washington, DC, USA, IEEE Computer Society (2001) 59
5. Dignum, F., Broersen, J., Dignum, V., Meyer, J.J.C.: Meeting the deadline: Why, when and how. In Hinchey, M.G., Rash, J.L., Truszkowski, W., Rouff, C., eds.: FAABS. Volume 3228 of Lecture Notes in Computer Science., Springer (2004) 30–40
6. Broersen, J., Dignum, F., Dignum, V., Meyer, J.J.C.: Designing a deontic logic of deadlines. In Lomuscio, A., Nute, D., eds.: DEON. Volume 3065 of Lecture Notes in Computer Science., Springer (2004) 43–56
7. Prakken, H., Sergot, M.: Contrary-to-duty obligations. Studia Logica **57** (1996) 91–115
8. Carmo, J., Jones, A.: Deontic logic and contrary-to-duties (2001)
9. Governatori, G., Rotolo, A.: Logic of violations: A gentzen system for reasoning with contrary-to-duty obligations. The Australasian Journal of Logic **4** (2006) 193–215
10. Wyner, A.Z.: Sequences, obligations, and the contrary-to-duty paradox. In Goble, L., Meyer, J.J.C., eds.: DEON. Volume 4048 of Lecture Notes in Computer Science., Springer (2006) 255–271
11. Governatori, G.: Representing business contracts in *ruleml*. Int. J. Cooperative Inf. Syst. **14** (2005) 181–216
12. Governatori, G., Milosevic, Z.: Dealing with contract violations: formalism and domain specific language. In: EDOC '05: Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05), Washington, DC, USA, IEEE Computer Society (2005) 46–57
13. Neal, S., Cole, J., Linington, P.F., Milosevic, Z., Gibson, S., Kulkarni, S.: Identifying requirements for business contract language: a monitoring perspective. edoc **00** (2003) 50
14. Milosevic, Z., Gibson, S., Linington, P.F., Cole, J., Kulkarni, S.: On design and implementation of a contract monitoring facility. In Benatallah, B., Godart, C., Shan, M.C., eds.: Proceedings of WEC, First IEEE International Workshop on Electronic, IEEE Computer Society (2004) 62–70
15. Xu, L., Jeusfeld, M.A.: Pro-active monitoring of electronic contracts. In Eder, J., Missikoff, M., eds.: CAiSE. Volume 2681 of Lecture Notes in Computer Science., Springer (2003) 584–600
16. Xu, L., Jeusfeld, M.A., Grefen, P.W.P.J.: Detection tests for identifying violators of multiparty contracts. SIGecom Exch. **5** (2005) 19–28
17. Chiu, D.K.W., Cheung, S.C., Till, S.: A three-layer architecture for e-contract enforcement in an e-service environment. In: HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 3, Washington, DC, USA, IEEE Computer Society (2003) 74.1
18. Degl'Innocenti, M., Ferrari, G.L., Pacini, G., Turini, F.: Rsf: A formalism for executable requirement specifications. IEEE Trans. Softw. Eng. **16** (1990) 1235–1246
19. Abate, A.F., D'apolito, C., Nota, G., Pacini, G.: Writing and analyzing system specifications by integrated linguistic tools. International Journal of Software Engineering and Knowledge Engineering **7** (1997) 69–99

20. Linington, P., Milosevic, Z., Raymond, K.: Policies in communities: Extending the odp enterprise viewpoint (1998)
21. Steen, M., Derrick, J.: Formalising ODP Enterprise Policies. In: 3rd International Enterprise Distributed Object Computing Conference (EDOC '99), University of Mannheim, Germany, IEEE Publishing (1999)
22. Governatori, G., Milosevic, Z.: A formal analysis of a business contract language. International Journal of Cooperative Information Systems **15** (2006) 659–685
23. Maurer, W., Mathaus, R., Frey, N.: A guide to successful sla development and management. http://www.fldcu.org/irmcoord/STOTRANS-SLA%20References.pdf (2000)