RC22765 (W0304-064) April 8, 2003
Computer Science

IBM Research Report

Simple Obligation and Right Model (SORM) - for the Runtime Management of Electronic Service Contracts

Heiko Ludwig
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Markus Stolze
IBM Research Division
Zurich Research Laboratory
8803 Rueschlikon, Switzerland

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Simple Obligation and Right Model (SORM) - for the Runtime Management of Electronic Service Contracts

Heiko Ludwig
IBM T.J. Watson Research Center
hludwig@us.ibm.com

Markus Stolze
IBM Zurich Research Laboratory
mrs@zurich.ibm.com

Abstract

Online purchase and delivery of goods and services requires an electronic contracting process. Formalization of contractual content enables automatic delivery of services and monitoring of the terms and conditions of the contract at service runtime. The Simple Obligation and Right Model (SORM) provides an abstract, domain-independent model of contractual content. Model instances can be interpreted and managed by applications involved in checking contractual entitlements and delivering and supervising a service in compliance with contractual rights and obligations. It captures the main types of rights and obligations and deals with their dynamics during the life-time of a contract.

1 Introduction

Electronic representations of contracts between organizations are an important pre-requisite for conducting business in a networked economy, or e-business on demand [2]. In many cases, a plain textual, natural language representation of the contractual content that can be read and interpreted by business people and lawyers suffices all requirements, now that electronic signatures stand in court in many legislations. However, if organizations want to support or automate all or a part of the life-cycle of a contract, a formal machine-interpretable representation is needed, e.g., to advertise a service, to negotiate a contract, to monitor contractual compliance, and to handle disputes [8]. This is particularly relevant in a context of Web services, which – based on standardized description and access – bear the potential to make services available across organizational boundaries on short notice for limited periods of time. A formal representation of a contract requires a model of its content, which depends on the phase of the life-cycle and the programs and people that interpret and deal with the contract.

- A model of the **structure** of a contract's content is useful in the creation and negotiation phase of a contract life-cycle. The structural elements are clauses, paragraphs, boiler plate fragments, etc. This model supports the composition process of contracts and allows word processors and other composition tools to edit and assemble contracts.
- In the fulfillment phase of the contract, we need a model of the **subject matter** of the contract, i.e. the **promises** made. A formal representation of the promises is the basis to configure, monitor and control a system fulfilling these contractual promises both from the perspective of the promising party and the party to which is promised. A model of what is promised is also needed for consistency checking in the creation and negotiation phase.

This paper is on a model of the subject matter of the contract, its promises. A promise is directed. The party that promises enters an obligation, becoming an obligee, while the party receiving the promise becomes an obligor. The obligor holds the right that the promise will be fulfilled. Right and obligation are hence dual concepts to a promise, a different view on the same relationship.

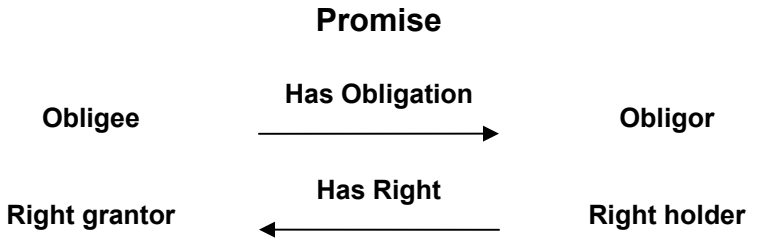

Figure 1: Terminology.

Depending on the subject domain, it is easier, or more customary, to express a contractual promise either as a right or as an obligation. For example, in contracts about the use of digital content, the promises given with respect to the digital content are usually expressed as rights that a licensing party obtains, not as an obligation of the owner of digital content to tolerate the use of the digital content by its contractual party. In the case of a contractual promise that requires one party to perform a service, this relationship is usually expressed as an obligation. Both views can be used in the same contract, e.g., stating that one party receives rights to digital contents for the obligation to pay an amount of money to the right grantor.

The design of a model of contractual promises depends on the purpose for which the model is being used. A deontic logic representation, for example, is very suitable for reasoning about promises and check consistency. However, there are no obvious constructs on how and when to check entitlements for requests or when to check promises such as compliance with a response time guarantee of an electronic service. The SORM model as proposed in this paper addresses these issues. A formal specification of contractual content should be able to be interpreted according to multiple models to suit all relevant purposes.

The objective of this paper is to propose a model of obligations and rights that is independent of a particular application domain and can be used to manage obligations and rights by a contract-implementing or supervising application as well as to serve as to serve as a contribution to a contract specification language. We proceed as follows: In the second section, we introduce an example case, derive requirements and discuss existing approaches. Subsequently, we propose a model of obligations. Section 4 introduces a model of obligation dynamics during the fulfillment of a contract. In the conclusion we give an outlook how to use the model to define formal contract languages.

2 Requirements for an obligation and rights model

2.1 Example cases

We will use two example cases to illustrate the scope of and the requirements on an obligation and rights model.

Online services

A service provider offers to supply stock quotes for particular ticker symbols through a Web services interface. In addition, the service offers to send notifications when the market price of a particular equity either goes below or above a user-defined threshold. The quotes are offered at different levels of response time. The stock quote service provider of our example targets multiple markets: It addresses small financial agents, and actively trading consumers that only request a limited amount of quotes a day as well as enterprise customers that request large volumes. Also, customers can use the stock quote values to place them on their web site for an additional fee, i.e. Web portals are also among the target customer group. Over time, the demands of the customers may change, e.g., responding to a change of interest in more near time stock information or, in the case of financial agents, growth of the business demands more stock quotes per day. Hence, some customers want to include an option to upgrade to a higher quote volume at a price set when the original contract is made.

From the service provider organization's point of view, it enters an obligation to serve stock quotes when requested by customer, hence granting the right to request to its customer. This is bound to a particular

number of requests, limiting the provider's obligation. It also guarantees a particular level of service with respect to the response time of the requests. In addition, customers may upgrade to a higher number of requests at runtime, hence modify a currently active obligation. Since the service provider addresses a heterogeneous set of customers entering in different obligations each time, it is important that the applications that offer service contracts and that monitor the services at runtime are obligation and rights-aware, addressing the particular situation of a customer. Likewise, this may be true for the service provider's customers monitoring the fulfillment of the rights they have.

Digital content providers

In another scenario, a distributor of music makes the music available for download over the Web. However, the downloading customer may only play the piece of music on the device onto which it has been downloaded to and may only do one backup copy. In addition, the piece of music may only be played for private use, not in public or for a paying audience.

In this example, the focus is on the rights of the customer that he or she acquires with a contract on digital music download. There are a number of restrictions to what the customer may do. However, in contrast to the previous example, the use of rights is more difficult to monitor since the online music provider is not involved when the music is actually played. The customers exercise their rights independently from a connection to the music site.

2.2 Requirements and design objectives

What is required to achieve the objective of an obligation and rights model that is the basis for managing those concepts in a applications implementing and supervising electronic contracts and specifying them? From the examples as well as the general understanding of contractual rights and obligations and their use in representing and processing electronic contracts, we can derive a set of general requirements in the following categories:

- **Expressiveness**; an obligation and rights model must represent all types of obligations and rights – on a sufficiently abstract level. Those main obligations and rights include the obligation to do something (send notification), the right to do something (play a piece of music) and the obligation, as well as a right, that a particular state is maintained (average response time is less than 2 seconds), each of which must be monitored in a different way. In addition, the obligation and rights model must represent how the currently active set of obligations and rights can change over the lifetime of a contractual relationship.
- **Generality**; the obligation and rights model must be independent from a particular application domain. Contract management applications, or obligation and rights-managing middleware platforms, may need to manage contracts from different application domains. However, the model should be open to refinement and extensions form particular domain.
- **Simplicity**; since the obligation and rights model must be usable across multiple domains, it is beneficial that if the model is simple, i.e. containing few concepts.

2.3 Related work

The issue of representing the concepts of right and obligation is not novel. Models have been built for different purposes.

Deontic logic extends first order logic (predicate logic) by modal operators with the semantics "may" and "must" [10]. Obligation and rights may be represented as expressions in this logic and may be reasoned about. Since it is based on first order logic, it very general and can be applied to any domain. This is a powerful instrument to *reason* about obligations, for example, check consistency. However, the semantics of rights and obligations is completely represented in the predicates of the logic expression, which is domain-specific. Hence, deontic logic expressions per se are not sufficient to derive how to monitor them. Furthermore, semantics of deontic logics is demanding and its concepts may not be suitable as a basis for writing applications or representing contractual content.

More recently, work on the **ODP Enterprise Language** resulted in a model of policies that cover the aspects of rights and obligations [1], [6], [12]. This model is general and expressive but deals separately with rights and obligations, which leads to redundancies. These redundancies make the model complex and prevented widespread adoption so far. Furthermore, it does not address the issue of obligation and rights dynamics.

In the area of digital content, a number of approaches to model rights have been developed. The **Open Digital Rights Language** (ODRM) can represent user rights to digital content to a great detail [5]. It is meant to be enforced by media players etc. and has a notation based on XML. Likewise, **XrML** [2] is a competing approach by another consortium. However, those approaches are restricted to their specific domains and, in general, do not address the issue of obligation.

In the field of service level agreements (SLAs), a number of approaches have been developed to formally represent the promises of a service provider. The **Web Service Level Agreement** (WSLA) approach explicitly deals with the concept of obligation, but only in the domain of SLAs [7], [9]. In addition, it does not address the concepts of rights.

In the field of **software engineering**, we find weaker concepts of obligation, not involving real organizations. Interface definitions such as CORBA IDL and Java interfaces may be interpreted as contracts that oblige the component implementing the interface to behave as described in its interface – an obligation to the client using the component. The approach of "design by contract" has been proposed by Meyer in the context of the Eiffel programming language [10]. However, those obligations carry very specific semantics and cannot be generalized.

3 SORM - obligation and right types

In this section we introduce the obligation and right types of the Simple Obligations and Rights Model (SORM). Since all obligations and rights are derived from promises, they are directed, the right granter owing to the right holder and the obligee owing to the obligor. Abstractly, we speak of the obliged and the beneficiary. In theory, an obligation may have multiple obliged parties and beneficiaries, though this is only customary in specific areas, e.g., shared responsibility for a loan. In this case, it must be explicitly represented which parties are obliged and which parties benefit.

All obligations and rights have the same attributes in common. This is a unique name, the set of beneficiaries and the set of obliged parties.

The objective of SORM is to provide an obligation model that is useful at runtime but independent from a particular application domain. From the point of view of managing obligations and rights at runtime, we distinguish three basic types, depending on the way they are monitored and enforced:

- **State obligation, state right:** Obligation to maintain a particular state, e.g. of a service. This is relevant, for example, in service level agreements for hosting or communication service that define a service in key indicators such as through put, jitter, delay, availability etc. An obligation and right of this type must be monitored permanently or checked each time an event occurs that may change the current state.
- **Action obligation, right to have an action performed**: Obligation of one or more parties to perform a particular action in given circumstances. This is important if a contract includes the performance of a business process or, very importantly, a payment. This is to be monitored and enforced when the circumstances dictate that an action must be performed or the performance must be completed. The performance of the action can be scheduled by the obligee and the monitoring of its correct completion can be scheduled by the obligor.
- **Option obligation, right to act:** Obligation to tolerate an action performed by another party. This includes making a service operation available, e.g., to retrieve equity quotes, and to allow a client to invoke it. This also includes the right to play digital content. From the point of view of the option obligation's beneficiary, it represents a right for this party to perform an action. It is solely up to the right holder when to execute an action, e.g., a user starts playing a piece of music on his or her device.

The entitlement to an action subject to the right must be checked upon invocation but cannot be scheduled.

Each of these types can be subtyped if appropriate.

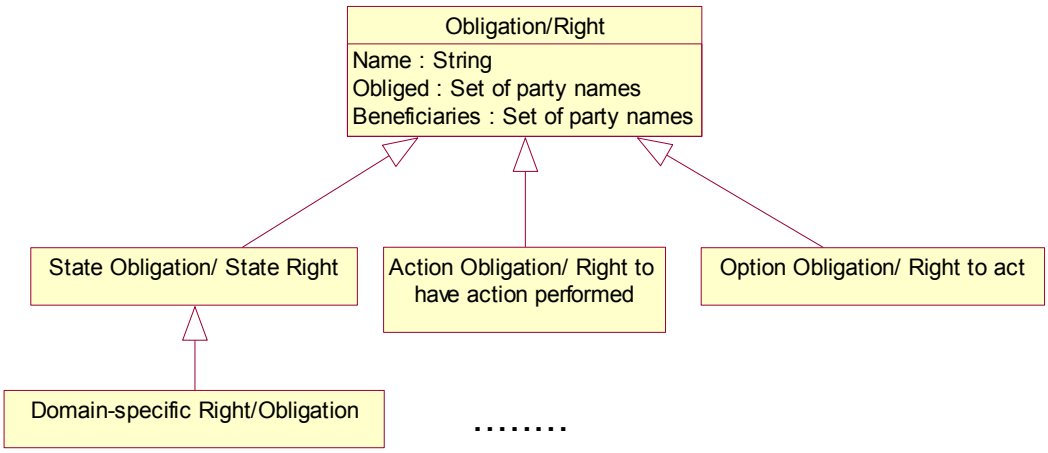

Figure 2: Obligation and right types.

In the subsequent discussion, we refer to rights and obligations mainly in their obligation form to keep it simple. Due to the duality of right and obligation, the corresponding right view is implied.

3.1 State obligation, state right

A state obligation guarantees that a particular condition holds in a defined period. We define a state obligation as a tuple:

so(name, OP, BP, c, period), where,

- *name* is the unique name of the obligation,
- *OP* is a set of obliged parties,
- *BP* is a set of beneficiary parties,
- *c* is the condition that must hold, representing a state of the world, and
- *period* is the period of time in which the condition must hold.

The following example is drawn from the case of section 2:

so(so1,
{Stock quote service provider},
{Stock quote customer},
a 95 percentile of stock quote requests, measured on an hourly basis, are returned to in 3 seconds, runtime of the contract)

In this simple example, the service provider promises to the customer a particular response time behavior throughout the runtime of the contract. In the given example, the guaranteed condition cannot be related to each request individually. Therefore, this obligation may be defined as a state obligation in the way shown above.

Another example relating to the class of service of the stock quotes:

so(so2,
{Stock quote service provider},
{Stock quote customer},

the returned stock quotes are the current NYSE market values delayed no more than ten minutes, runtime of the contract)

This example refers to the delay of the quoted values with respect to the market price setting, not the technical delay as in the first example.

3.2 Action obligation, right to have action performed

An action obligation expresses that the obliged parties must perform an action a if and when a given event occurs and defined preconditions are met. We define an action obligation as a tuple:

ao(name, OP, BP, a, e, pre, completiontime), where

- *name* is the unique name of the obligation,
- *OP* is a set of obliged parties,
- *BP* is a set of beneficiary parties,
- *a* is an action to be performed, potentially a complex, structured process comprising multiple steps,
- *e* is an event on whose occurrence the action is to be performed, if the precondition holds,
- *pre* is the precondition that must be met for the action to be executed at event e,
- *completiontime* is the specification of the point in time until the action must be completed. This specification can be either relative to the start of the action or an absolute point in time.

Again, we take an example from the stock quote service case:

ao(ao1,
{Stock quote service provider},
{Stock quote customer},
send instant message notification about price of IBM stock and then debit 10 cents to service customer's account ,
new deal closed on IBM stock at NYSE,
price of deal more than $130 per share,
1 minute after the deal was published by NYSE)

In this example, the stock quote service provider promises to the customer to send an instant message containing the stock price of IBM within 1 minute if and when a deal of more than $130 per share is closed on NYSE. The precondition must be evaluated each time a deal on IBM stock is closed on NYSE. For this service 10 cents will be debited to the customer's account with the service provider.

The example is a promise to perform a particular action and thus is well represented as an action obligation.

3.3 Option obligation, right to act

An option obligation, or right to act, expresses that the obliged parties allow the beneficiary parties to perform an action. This only makes sense if the beneficiary parties are not entitled to so anyway, even if no contract is signed. Also, the obliged parties must be able to grant this right to the beneficiaries.

We define an option obligation as a tuple:

oo(name, OP, BP, a, pre), where

- *name* is the unique name of the obligation,
- *OP* is a set of obliged parties,
- *BP* is a set of beneficiary parties, the parties that may perform the action,
- *a* is the action that the members of BP may execute, which may be complex,
- *pre* is the precondition that must be met for the action can be performed.

This type of obligation is illustrated in the following example:

oo(oo1,
{Digital content distributor},
{Digital content customer},
play digital content,
weekdays,
)

The above example represents the main service function of our case. The digital content customer may play digital content only weekdays.

Another example:

oo(oo2,
{Stock quote service provider},
{Stock quote customer},
(set new stock quote delay from 10 minutes to 5 minutes,
debit $ 10000 to service customer's account)
time of request is between 1 AM and 1 PM EST,
)

In this example, the service customer may improve his or her class of service in terms of the time delay of the stock quote. While the original contract foresaw to return quotes as of 10 minutes ago, the customer can decrease the delay to 5 minutes, for a price of $ 10000.

3.4 Representation of obligation and right content

The above definitions and examples of obligation and right types outline the structure of these constructs. However, to be able to automatically process contracts we need a formal representation of the content of the obligation and right elements, too. There are multiple ways to formalize the content, i.e. the conditions, events, time expressions, and the invocation of activities. None of these aspects is novel and suitable formal representations can be used where appropriate, depending on the particular domain. However, the contract must capture the obligation content in a way that is mutually understood by the contracting parties, based on a shared ontology. We need formal representations for the following elements of obligations and rights:

Conditions can be interpreted as Boolean functions that can be resolved to true or false at runtime. To be automatically verifiable, condition must be expressed as expressions in a formal language. In this formalism, we may use first order logic or a modal logic where appropriate. The logic variables used in the predicates are unique names of objects defined in the context of the obligations, i.e. the contract. The choice of the appropriate set of predicates to be used depends on the particular domain of the contract. Both objects and predicates must be based on a shared ontology of the contracting parties.

Example:

greater(last_ibm_price, 130)

The semantics of the object *last_ibm_price*, being the share price, and the predicate *greater* must be understood between the parties.

Points in time are either absolute date and time descriptions or they are relative to a given point in time, e.g., 60 seconds after the *getQuote* operation was received. Periods are defined by their start and end point in time. Any appropriate formal representation fits.

In action obligations we define which **action** must be performed in case the precondition holds. We assume a model of action that is similar to a function in a programming language, having a name and a set of parameters. Each reference to perform an action thus includes (1) the name of the action and (2) the marshalling of its parameters. The parameters are either names of objects of the contract or constants.

Examples:

set(stockQuoteDelay, 10)

debit(customerAccount, 10000)

In these examples, stockQuoteDelay and customerAccount are names of objects of the contract. These action descriptions can be implemented by descriptions of simple invocations of Web services or process, e.g., represented in BPEL [3].

We require a formal **event model**. Simple events could be defined by their unique name. If necessary, this model can be extended to complex, structured events.

4 SORM - Obligation and right dynamics

Within the lifetime of a contractual relationship between two parties, the currently valid set of obligations can vary. New obligations can be introduced by one contracting party exercising an option obligation, e.g. a party may choose a higher quality of service, which entail that there arises the obligation to pay an additional fee. In addition, new obligations may be introduced as a result of executing an action that was conditional on its precondition.

4.1 Actions modifying the obligation set

Changing the set of valid obligations requires action to do so. We introduce an action type that adds a new obligation:

addObligation(obligation): Obligations ! Obligations [{obligation}, where

- *Obligations* is a set of Obligations of a contract and
- *obligation* is an individual obligation.

Similarly we define an action that removes an obligations from the current set:

removeObligation(obligation): Obligations ! Obligations / obligation, where

- *Obligations* is a set of Obligations of a contract and
- *obligation* is an individual obligation to be removed from the set.

For convenience reasons, we introduce a changeObligations operation that adds and removes sets of operations with one operation.

changeObligations(ObligationsRemoved, ObligationsAdded): Obligations ! (Obligations / ObligationRemoved) [ObligationsAdded, where

- *ObligationsRemoved* is a set of Obligations of a contract that will be removed from the current set of obligations and
- *ObligationsAdded* is the set of obligations that is added to the current set.

These operations can be used in obligations like any other action.

4.2 Using modifying operations in obligations and rights

In many cases, actions executed in the context of action obligations or rights to act entail a change in the set of currently active obligations. In those cases, the addObligation, removeObligation, and changeObligations operations are used in action obligations and rights to act. This is the case, for example, in the case of our stock quote service providing its customers the option to invoke the stock quote operation. If the customer exercises the option, a new obligation arises for the provider, namely to perform an action under given conditions. This is just a new action obligation. The following example illustrates this:

oo(getStockQuote,
{Stock quote service provider},
{Stock quote customer},

```
addObligation (     ao(       ao2,
                              {Stock quote service provider},
                              {Stock quote customer},
                               return price of the requested stock,
                              on activation,
                              <no precondition>,
                              within 2 seconds
                              )
                    ),
time of request is between 9 AM and 5 PM EST,
)
```

This right to act "getStockQuote" grants the customer the right to activate a new action obligation that obliges the provider to execute the action "return price of the requested stock". This right is available between 9 AM and 5PM. The action must be executed immediately with no further pre-condition and must finish within 2 seconds. The service provider may charge $1 for each quote. In this case, two action obligations are added, the one above and an additional one obliging the customer to pay $1, for example, by the end of the month.

```
oo(     getStockQuoteForMoney,
        {Stock quote service provider},
        {Stock quote customer},
        changeObligations (         {},
                                    { ao(     ao2,
                                              {Stock quote service provider},
                                              {Stock quote customer},
                                              return price of the requested stock,
                                              on activation,
                                              <no precondition>,
                                              within 2 seconds
                                              )
                                    ),
                                     ao(      payment,
                                              {Stock quote customer},
                                              {Stock quote service provider},
                                              pay one dollar,
                                              on activation,
                                              <no precondition>,
                                              until the end of the month
                                              )
                                    }
                    ),
        time of request is between 9 AM and 5 PM EST,
)
```

The first parameter of the changeObligations operation is the empty set since we only added obligation.

Many contracts include the right to upgrade to a different level of service, which represents a change in a state obligation in our model. Also, we find scheduled changes of the obligation set, such as in the following example:

```
ao(     scheduledQoSUpgrade,
        {Stock quote service provider, Stock quote customer},
        {Stock quote customer, Stock quote service provider },
        changeObligations (         { so(     95percentileQoS,
                                              {Stock quote service provider},
```

{Stock quote customer},
a 95 percentile of stock quote requests, measured on an hourly basis, are returned to in 3 seconds, runtime of the contract)

},

{ so(98percentileQoS,
{Stock quote service provider},
{Stock quote customer},
a 98 percentile of stock quote requests, measured on an hourly basis, are returned to in 3 seconds, runtime of the contract)

}),

August 1st 2004,
<no precondition>,
before 8 AM on August 1st 2004)

In this scheduled QoS upgrade, the parties agree to upgrade the state obligation defining the percentile of requests served within 3 seconds on August 1st 2004. This means removing the old state obligation defining the 95 percentile requirement of the provider and adding the new state obligation. Since this change is scheduled, and hence an action obligation, it can be seen that both parties must implement this change of status and both parties benefit from it.

4.3 Obligation and right states

In large sets of obligations and rights, it is inconvenient to define large modifications of the current state set for each action that may cause it. In addition, the dynamics of large sets of obligations are difficult to manage on an obligation-by-obligation basis. We need an abstraction mechanism that allows us to group obligations and manage entire obligation groups.

We introduce the notion of an obligation state. An obligation state is a tuple:

orstate(Obligations), where

- *Obligations* is a set of obligations and rights that are valid when obligation and rights state (OR state) is activated.

Once the OR state is activated, smaller changes to the obligation set can be caused by *addObligation* and *removeObligation* actions.

We need a new action to change to a new obligation state:

newState(state): CurrentObligations £ NewObligations$_{state}$! NewObligations$_{state}$, where

- *state* is the state that the contract is to be set to,
- *CurrentObligations* is the current set of obligations and rights,
- *NewObligations$_{state}$* is the new set of obligations and rights, corresponding to the state.

The current set of obligations and rights is replaced by the new set.

4.4 Consolidating obligation and right states with individual dynamics

The combined use of OR states and individual changes of the obligation and rights set based on option obligations can lead to ambiguities about the currently valid set of obligations. What happens to a newly added obligation when OR states are switched? For this purpose, we distinguish two sets of obligations and rights in a contract.

- **Background obligations and rights** are obligations and rights that are in force independently of the current OR state.
- **State-based obligations** are the obligations and rights of the currently active OR state.

Individual add and remove operations on obligations and rights only modify the background obligations and rights, which are not affected by changes in the OR state.

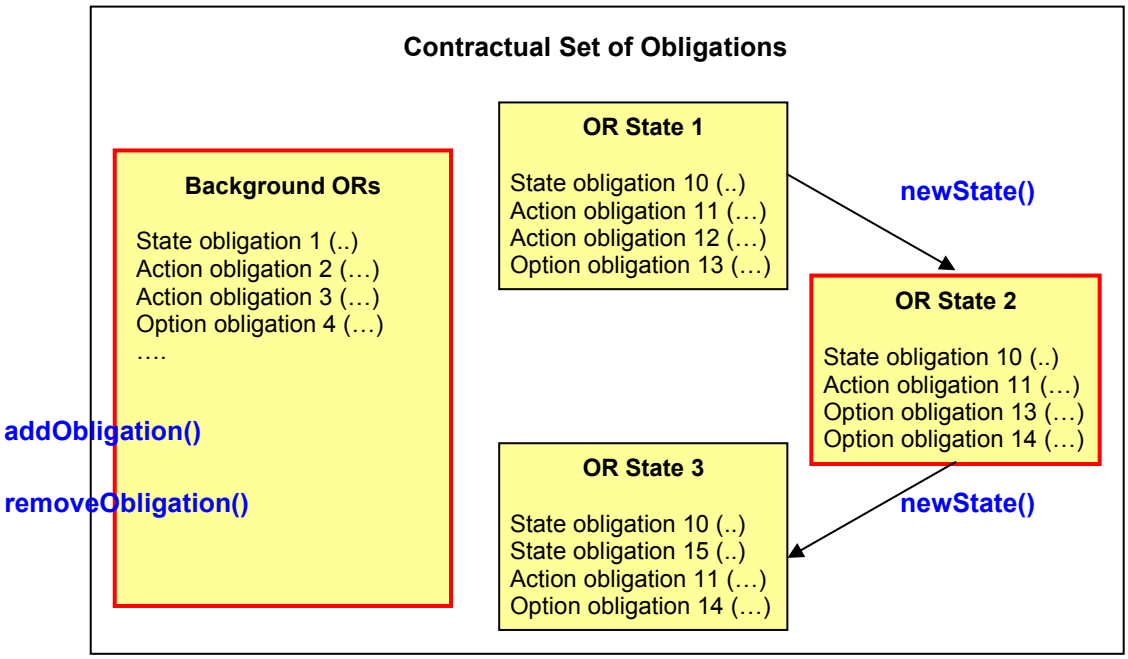

Figure 3: Dynamics of the obligation set of a contract.

The figure illustrates the use of background obligations and rights and OR states. In this example, the current set of background obligations and rights comprises state obligation 1, action obligations 2 and 3, and option obligation 4. This set of background obligations can be modified using the addObligation and removeObligation actions (in blue). In addition, the contract comprises three obligation states. States are changed using the newState action, which, for example, could be offered as an option obligation or as an action that is to be performed at a given time. Currently, OR state 2 is active. Hence, the currently valid set of obligations comprises the background obligations plus the obligations of OR state 2.

5 Summary and Conclusion

This paper proposes the Simple Obligations and Rights Model (SORM) for the domain-independent representation and management of promises in applications implementing and supervising electronic contracts, which is particularly relevant in a Web services context where business relationships may be set up at short notice. Based on the duality of obligations and rights, the model can express – abstractly – all forms of obligations and rights based on three types, state obligation - right, action obligation – right to have action performed, and option obligation – right to act, distinguishing the different ways these promises are dealt with at runtime. Using operations modifying the set of currently active obligations and rights, the model can also capture the dynamics of obligations and rights over time, hence facilitating long-running and complex contracts. SORM can be used as a basis for applications dealing with rights and obligations beyond a particular application domain.

SORM can also be used to design a formal contract language. Obligations and rights are primarily used in the context of contracts between organizations or individuals. The expressiveness of the contract language, i.e. which subset of SORM is actually used, and the specific syntax must depend on the capability of the contract-interpreting application to deal with the flexibility of potential obligation and right expressions. If an application can only deal with, say, state obligations, the language can be limited to this type. Also, a contract language must address to formalize descriptions of conditions, events, actions and times, to the extent it needs. SORM has been developed as a generalization of the Web Service Level Agreement

(WSLA) language in which many ideas of SORM have been tested in an environment of SLA management [7] [9].

Since the obligations and rights model of a contract language and a contract management application should match, SORM is a good candidate for a contributor to the overall content model of an electronic contract.

References

[1] J. Cole, J. Derrick, Z. Milosevic, and K. Raymond: *Policies in an enterprise specification*. In Morris Sloman, editor, *Proceedings of the Policy Workshop*, 2001, Bristol UK, January 2001.

[2] ContentGuard : eXtensible rights Markup Language (XrML) Specficiation 2.0 – Part 1: Primer. 20 November 2001. http://www.xrml.org/ on March 27, 2003.

[3] F. Curbera, Y. Goland, J. Klein, F. Leyman, D. Roller, S. Thatte, S. Weerawarana; *Business Process Execution Language for Web Services (BPEL4WS) 1.0*; August 2002, http://www.ibm.com/developer-works/library/ws-bpel

[4] Y. Hoffner, S. Field, P. Grefen, H. Ludwig: Contract-driven creation and operation of virtual enterprises. In *Computer Networks 37*, pp. 111 - 136, Elsevier Science B.V. 2001.

[5] R. Iannella: *Open Digital Rights Language (ODRL)*. W3C Note, 19. September 2002, http://www.w3.org/TR/odrl/.

[6] ISO/IEC JTC 1/SC 7*: Information Technology - Open Distributed Processing - Reference Model - Enterprise Language: ISO/IEC 15414 | ITU-T Recommendation X.911,. Committee Draft.* 8. July 1999.

[7] A. Keller, H. Ludwig: The WSLA Framework – Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and System Management* (11), Nr. 1, Special Issue on E-Business Management. Plenum Publishing Corporation, 2003.

[8] Ludwig, H; Hoffner, Y: The Role of Contract and Component Semantics in Dynamic E-Contract Enactment Configuration. In *Proceedings of the 9th IFIP Workshop on Data Semantics (DS9)*, pp. 26 - 40, Hong Kong, 2001.

[9] H. Ludwig, A. Keller, A. Dan, R. King, R. Franck: A Service Level Agreement Language for Dynamic Electronic Services. Electronic Commerce Research (3), Nr. 1, pp. 43 – 59, Kluwer Academic Publishers, 2003.

[10] B. Meyer: Object-oriented Software Construction. 2nd ed. Prentice-Hall, 1997.

[11] J.-J.Ch. Meyer and R.J. Wieringa (eds.): *Deontic Logic in Computer Science: Normative System Specification.* Wiley and Sons, 1993.

[12] M. W. A. Steen, J. Derrick: ODP Enterprise Viewpoint Specification. *Computer Standards and Interfaces*, 22:165--189, September 2000.