Semi-Automated Management of Web Service Contracts

Steffen Lamparter

Institute AIFB
Universitaet Karlsruhe (TH), Germany
E-mail: sla@aifb.uni-karlsruhe.de

Stefan Luckner

Institute of Information Systems and Management (IISM)
Universitaet Karlsruhe (TH), Germany
E-mail: luckner@iism.uni-karlsruhe.de

Sibylle Mutschler

Institute of Information Law
Universitaet Karlsruhe (TH), Germany
E-mail: mutschler@ira.uni-karlsruhe.de

Abstract: Service-oriented computing as a concept for providing interoperability and flexibility within heterogeneous environments has gained much attention within the last few years. Dynamically integrating external Web services into enterprise applications requires automatic contracting between service requestors and providers and automatic contract monitoring. This paper suggests a semi-automatic approach since in the current legal environment full automation is not feasible. We elaborate on the content of Web service contracts from a legal perspective and derive a set of legal requirements. Based on these requirements we propose an ontology-based representation of contract clauses as well as monitoring information. We can thus automatically evaluate whether a service execution meets the requirements expressed in a contract.

Keywords: contract representation, ontology, compliance checking.

1 INTRODUCTION

Information systems of the future will be combinations of loosely-coupled services. In service-oriented architectures (SOA), application systems are assembled as required by pulling together various services. The implementation of services is encapsulated and numerous service providers may provide the same functionality. Hence, a customer may choose from a variety of implementations depending on his preferences. The concept of

service customization enables the same service to be offered at different service levels for different prices. Usually a specification of the service levels agreed upon is called a *service level agreement* (SLA) (Ludwig et al., 2003). We call a legally binding specification of such service level agreements together with additional obligations that result from the contracting process (such as paying a certain price as compensation) a Web service contract (Hoffner and Field, 2005). This corresponds to the definition given by Reinecke et al. (1989), where "a contract is a legally enforceable agreement, in which two or more parties commit to certain obligations in return for certain rights."

Due to the cross-organizational and collaborative nature of business processes, which are supported by today's service-oriented architectures, contracts have become a key governance mechanism regulating business interactions. In spite of their importance, today's enterprizes still treat contracts merely as paper documents regulating the case where something goes wrong and without linking them to the cross-organizational interactions that they govern. Thus, a more holistic approach to contract handling is required that supports the following features (Milosevic and Governatori, 2005):

- formal contract languages that provide open, transparent and up-to-date information about contract data and the status of a contract;
- mechanisms that use information from contracts as a basis for monitoring of contract compliance and subsequent notifications and enforcement measures;
- mechanisms and tools that support management of the entire contract life cycle, including contract formation, contract execution and contract monitoring;
- tools that support personnel in meeting their obligations that arise from the contract.

That means, in a service-oriented architecture a formalized Web service contract can be directly used to govern the business interactions executed via Web service invocations. Web service contracts have to be found automatically and have to be legally reliable. This requires on the one hand a formal machine-interpretable language that enables automated contract formation, execution and compliance checking; and on the other hand the expressivity to specify all legally required clauses. This ensures that any violation of pre-agreed service levels results in a penalty for the party that is responsible for the violation.

Over the last decades a lot of work has been devoted to the formalization of contracts and legal norms in general – mainly in the areas Artificial Intelligence, Computer Science and Philosophical Logic (Daskalopulu and Sergot, 1997). However, up to now formalization of legal contracts has been restricted either to relatively simple contractual clauses expressed via a standard syntax (e.g. Milosevic, 1995; Griffel et al., 1998; Cole and Milosevic, 2001; Grosof and Poon, 2003; Angelov and Grefen, 2003; Global Grid Forum, 2006; Governatori, 2005) or relies on very complex logical formalisms (e.g. Hage, 1996; Tan and Thoen, 1998; Sergot, 2001) which are not computationally tractable or lack any support for inter-organizational interactions. Since the trade-off between expressivity and tractability cannot be easily resolved, we focus on a semi-automated approach where a natural language umbrella contract is manually closed with different service providers and only some of the terms are fully formalized. In fact, only obligations that have to be dynamically configured during the contracting process or that should be monitored automatically after contract execution have to be formalized. This eases the formalization task and allows the usage of a simpler, computationally tractable logical formalism.

In this paper, we propose the use of ontology languages for formally representing Web service contracts. Ontologies come with a logical calculus that enables representing information in a formal and standardized way. Thereby, ontologies provide interoperability, flexibility and tool support. These advantages also carry over to the contract specification. By providing an open, transparent and interoperable view on contractual data, ontology-based contract representation enables a tight integration of up-to-date contractual information with the collaborative business interactions they govern. This means, the machine-interpretable contractual information can be easily accessed by contracting and contract monitoring tools, and it can be easily shared with business partners. In addition, standard tools supporting the logical formalisms of the ontology can be used to perform sophisticated contract monitoring that involves logical inferencing.

Our paper is structured as follows. In section 2 we introduce a scenario which provides a use case for the subsequent sections. In Section 3 we discuss related work. In Section 4 the idea of a semi-automated approach to contracting and contract monitoring is presented. In this context, a informal umbrella contract is closed, which constitutes the environment that enables automated contracting of formal individual contracts on a per-invocation-basis. General design considerations of our ontology framework are described in section 5. The formalization of individual contracts as specializations of *DnS:SituationDescriptions* is then presented in Section 6. In order to support the settlement phase, Web service monitoring information has to be formally represented. How this can be realized by means of a *DnS:Situation* is outlined in Section 7. Finally, in Section 8 modeling primitives for evaluating contracts are presented. This requires knowledge how specific contractual clauses have to be interpreted. Since this knowledge is usually available only as tacit knowledge of legally educated persons, it also has to be externalized into a machine readable and executable form. Finally, we conclude the paper in Section 9.

2 SCENARIO

In order to reduce credit risk and to select profitable customers, companies rely upon credit information. The latest legal developments around risk management such as Sarbanes Oxley have forced companies to have a closer look at the management of financial risk. Financial information relating to the creditworthiness of companies, the profitability of their business or the quality of their senior management helps companies to assess the risk of doing business with each other and respond to increased or decreased risk. Such financial information is sold by companies like Dun & Bradstreet or Creditreform. Based on credit information, companies will decide whether to start business with another company or determine and adapt lines of credit. In the past, such lines of credit have often been considered too late as the buying of credit information was done manually and not always on a continuous basis. By providing this critical credit information via Web services the credit information can be integrated into existing enterprise applications facilitating risk decisions based on externally provided and permanently updated data. Often the integration of the services has to be done dynamically based on transaction-specific information, such as the origin of the business partner, type of transaction, etc.

From a technical point of view, such financial services are functions that typically take the name of a company as input and return certain financial information that can be classified according to the taxonomy shown in Figure 1.

In addition, service levels guaranteed by the Web service providers are an essential

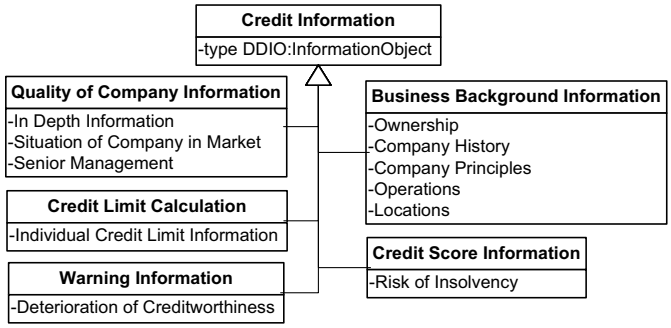

Figure 1 Credit Information Taxonomy.

decision criteria for the provider selection. Wrong financial information could lead to a wrong decision and thus to huge financial loss. In the case of financial information services typical quality of service attributes are update time of the delivered information, delivery time (response time), warranties about the accuracy of the information, etc. In addition, other relevant attributes have to be considered during service selection such as the payment terms. A customer's IT management infrastructure has to include execution monitoring of the service usage and checking for conformance with the contract (Casati et al., 2003).

In order to allow for dynamic selection of a credit information service as well as automated compliance monitoring, the contract has to be expressed using *machine-interpretable syntax*. Contracts can then be stored and parsed by computers without any involvement of humans and they can be connected to inter-organizational business process they govern. Due to the need for logical inferencing in case of contract monitoring the contract language also has to support *formal semantics* that enable legal reasoning. In an open environment where a service requestor is invoking a web service provided by a different department or company, *interoperability* becomes a major issue. It is required in order to support cross-organizational service invocations, where contracts have to be exchanged between the participants in a transaction. To be able to exchange contracts and to dynamically add new contracts to a local knowledge base, a fully *declarative contract specification* is essential. This is particularly important in open environments where we have to be able to dynamically add new contractual clauses or service properties to the contract. Moreover, the language has to be *sufficiently expressive* to specify all of the required contract clauses, while being *decidable*. Decidability makes sure that no contracts can be formulated which cannot be evaluated in finite time by our algorithms.

3 RELATED WORK

In this section, we discuss whether the existing literature on contract languages meets the requirements derived in the previous section. Table 1 summarizes the results by showing which requirements are fulfilled by different categories of contract languages. In this context, a solid circle indicates that a requirement is met, an empty circle that a requirement is not addressed, and a half-empty circle refers to categories of languages in which the requirement is only fulfilled partially or not by all approaches in the category. In the following, we discuss the results in more detail.

XML-based languages introduce machine-readable contract models (e.g. ebXML

	declarative language	machine-interpretable syntax	formal semantic	interoperability	sufficient expressivity	decidability
XML-based approaches	●	●	◐	◐	○	●
Rule-based approaches	◐	●	●	◐	○	◐
DL-based approaches	●	●	●	◐	○	●
Template-based approaches	●	○	○	○	●	●
Our Approach	●	●	●	●	●	●

Table 1 Analysis of related work with respect to requirements.

(OASIS ebXML Joint Committee, 2004)) that can be considered as a structured contract document. Some of them are designed for a Web service scenario. The Web Service Level Agreement (WSLA) project focuses on the quality of service aspect within a contract (IBM Corporation, 2003). It also addresses the monitoring of an agreement. However, the project covers only some specific, quality of service related elements of a contract. While WS-Policy (W3C, 2006), WS-Agreement (Global Grid Forum, 2006), or WSPL (Moses et al., 2003) take a similar approach, they are more general and not restricted to quality aspects. While these XML-based languages feature a declarative, machine-interpretable syntax, they require specialized interpretation engines that obstruct interoperability and flexibility (e.g. individual users cannot add proprietary terms to the vocabulary). In addition, most approaches are restricted to certain aspects (such as WSLA) and thus lack the expressivity required for fully expressing Web service contracts. Evaluation algorithms for XML-based contract languages are typically designed to be decidable which is rather straightforward given the limited expressiveness of the approaches.

Several **rule-based contract languages** have been proposed in recent years. Most of them are based on deontic logic (e.g. the ODP Enterprise Language presented in Milosevic (1995)) which extends first order logic by modal operators like 'may' and 'must'. The starting point for most work in this area is the seminal work of Hohfeld (1913), which introduces the legal relations *duty*, *right*, *power* and *liability*. A formal, more fine-grained analysis has been performed by Kangar (1972) and Lindahl (1977) yielding additional normative positions. For a good overview of normative positions the interested reader is referred to Sergot (2001). In contrast to the previous XML-based languages, these approaches rely on a clear, formal semantics and feature sophisticated legal reasoning. However, since there is no standardized serialization of these logical formalisms, they are not directly applicable to Web service contracting. RuleML[a] is a first attempt to establish a standardized syntax for exchanging logical rules. Under the umbrella of RuleML different kinds of logics can be used to express contracts such as courteous logic programs or defeasible logic. Such approaches are SweetDeal (Grosof and Poon, 2003), DR-NEGOTIATE (Skylogiannis et al., 2005; Governatori, 2005) and RBSLA (Paschke, 2006). All three are rule-based approaches use defeasible reasoning (i.e. Courteous Logic Programs or defeasible logic) to specify contracts. However, there are some issues regarding the use of a (pure) logic programming paradigm. Often such languages do not provide full-fledged declarative semantics and thus combining rules from different sources becomes highly problematic. In

[a] www.ruleml.org

fact, manual integration of the different logic programs might be required, which obstructs interoperability in a cross-organizational environment. Since in our setting contracts have to be integrated from different sources, this is a major drawback. Another problem with respect to interoperability in the Web is the fact that RuleML provides only a standard syntax for exchanging rules, but no standardized semantics for the syntax elements. From an expressivity point of view logic programming languages do not support equality reasoning required for expressing integrity constraints as well as number restrictions and lack existential quantification. The decidability of the approach depends on the concrete rule language used. In particular, modal logics that provide expressive deontic operators easily become undecidable and are therefore not applicable in our context.

Another stream of logic-based contract languages relies on the family of description logics (Baader et al., 2003). Such **DL-based approaches** have increasingly gained attention with the standardization of the Web Ontology Language (OWL) (W3C, 2004). OWL defines the decidable fragment OWL-DL which is based on the description logic *SHOIN*. Although the DL-based approaches lack some expressivity with respect to rule languages (e.g. only tree models can be expressed), they provide several major advantages: they are declarative, which enables to combine and enforce/analysis contracts from different sources in the Web; they support integrity constraints and existential quantification; and OWL comes with a standardized, Web compliant serialization which is crucial to ensure interoperability in the heterogeneous and open Web environment. Prominent examples that make use of OWL are the following policy ontologies that can be used to express constraints in contracts: An approach which expresses WS-Policy (W3C, 2006) using OWL is presented in (Kolovski et al., 2005). Similarly, in KAoS (Uszok et al., 2004) OWL concepts are use to express and reason over policies. The policy specification language REI also relies on OWL for communicating policies, however, to express constraints the extend the approach with logic-like variables and rules. Finally, there are approaches that translate WS-Agreement to an OWL ontology (Jin and Wu, 2005; Oldham et al., 2006). In Oldham et al. (2006) ontologies are also extended with rules to increase the expressivity. Since some of these approaches use proprietary extensions to OWL, interoperability and decidability becomes a problem. Furthermore, all the DL-based approaches are clearly not capable of expressing a complete Web service contract. For example, they lack mechanisms to interpret fuzzy, context-dependent clauses which typically can be found in natural language contracts. Since all logic-based approaches are based on a clear formal semantics, automated contract verification, validation and consistency checking is supported. Concrete approaches are, for example, presented in Paschke et al. (2005); Governatori et al. (2006).

The approaches discussed above all enable closing machine-interpretable contracts in an automated fashion. However, they are all not capable of fully expressing a real-world contract and thus fail in providing legally reliable contracts. An alternative is to transform formal agreements to natural language contracts by means of a **template-based approach** as presented in Hoffner and Field (2005). The problem here is that we loose the formality of the contract specification and thus cannot automate contract execution and monitoring.

Recapitulating, existing approaches either lack formality, such as the template-based approaches, and thus do not enable a high degree of automation and interoperability, or they make use of complex logics to formalize contractual information. The latter approaches lack expressiveness and typically make use of proprietary formalisms which obstruct interoperability and are not proven to be decidable. In **our approach**, we also employ a logic-based approach, where contracts are expressed by instantiating an OWL-DL ontol-

ogy. In order to be able to express obligations and permissions, we use DL-safe rules (Motik et al., 2005) which is a decidable fragment of the Semantic Web Rule Language (SWRL) with a clear syntax and semantic. Thereby, we derive a fully declarative language with a Web-compliant, machine-interpretable syntax and formal semantics. Since we rely on standardized formalisms, a high degree of interoperability can be realized. In terms of expressivity, our approach is different in that we do not strive for full automation, but augment an automatically closed contract with an umbrella contract that provides the legal basis for the automation.

4 SEMI-AUTOMATED CONTRACTING

Full automation of the contracting lifecycle has so far been investigated only for very simple contracts. Semi-automated contracting can be seen as an approach, where a contract is composed of two separate parts: an *umbrella agreement* which is directly negotiated by human beings and an *individual contract* automatically negotiated and closed by software agents.

4.1 Umbrella Contract

The umbrella agreement is presently necessary to define the legal conditions under which software agents can enter into binding agreements as not all jurisdictions acknowledge negotiating and contracting by software agents. The service requestors agree on an umbrella agreement with several Web service providers. The umbrella agreement will therefore define the framework for several software agents to negotiate the individual contracts. The umbrella agreement regulates the following issues:

- the beginning of the contractual relations between all parties, how long the umbrella agreement is valid and how and when it can be terminated;
- the types of Web services to be negotiated;
- the timeframe for negotiations (preferably 24/7);
- auxiliary duties of the parties such as maintenance or the obligation to treat customer information confidentially;

These clauses form the continuous contractual relations between the parties and span more than one Web service invocation. In particular, they are not customizable and not negotiable. Often each umbrella contract closed by a requester with different providers contains the same clauses. Differentiation between the providers is realized in the individual contract, which captures content such as price, license type, payment terms, response time guarantees, etc.

4.2 Individual contract

In an individual contract most aspects are customizable. Web service requests and offers can be seen as contract templates, where several values are possible for each attribute. In the matching and allocation phase one value has to be chosen for each attribute before

a contract can be concluded. This configuration is then used in the contract formation process that generates the appropriate provider and customer obligations of the contract.

In the following, we illustrate the general content of an individual contract for a typical information service such as a credit information service. In this context, we discuss for different content categories whether certain clauses should be in the individual contract.

Scope of Agreement (§1). Although the general trading objects which can be exchanged automatically might be defined in the umbrella agreement, it is important that the agents have some flexibility to find agreements. For example, let an umbrella agreement define the scope of the agreement as Web service providing 'Credit Information' or 'Route Information'. Then in the individual contract the exact type of information (e.g. 'Business Background Information' or 'German Route Information') can be automatically determined in the matching process.

Provider Obligation (§2). In this category obligations of the provider are defined that can be customized for each invocation of the service. This is thus the main category where service level agreements are contained. For example, it is usually price relevant how old the credit or route information is. The software agents might therefore negotiate the update periods of the provided information. Of course, also other quality of service guarantees, such as maximal response time or the period in which errors have to be corrected, can be specified here.

Use of Information (§3). The individual contract will specify how the customer may use the information. This category may also involve obligations that restrict the use of information. For example, a contract clause may grant a transferable license to use the information or a non-transferable license and define further to what extent the customer may use the credit information within its company or towards third parties.

Warranties and Liabilities (§4). Since warranties and liabilities directly influence the costs of a provider, they are highly price relevant. We let the software agents negotiate about the warranty level but not about the legal obligations resulting from a breach of warranty. The legal complexity, including the restrictions by law to contract out certain statutory warranties and liabilities, does not allow for full automation at present. For example when customizing warranty levels the following scheme can be applied: (*1*) The service provider does not give any warranty as to the accuracy of the information. (*2*) The service provider does not warrant the accuracy of the information, but warrants that it has put the information together with utmost care and state-of-the-art-methods. (*3*) The service provider guarantees that the information is 100% correct.

Delivery Time (§5). The delivery of the information can be automatically customized in a way that the service as to be provided immediately after the individual contract is concluded or at a later, negotiated time. The legal consequences of non- or late delivery however are set forth in the umbrella agreement.

Prices and Payment Terms (§6). Finally, the prices and payment terms have to be specified, which can be seen as customer obligations. While the parties defined the details of invoicing in the umbrella agreement, some parameters such as price for the individual Web service or the due date of the payment can be fixed dynamically.

After closing a contract in the settlement phase the participants monitor whether the contractual duties are fulfilled. However, full automation of the monitoring step is impossible since assessing the quality of a Web service can only be done by taking external and not quantifiable factors into account. Nevertheless, some aspects can be monitored by the

system automatically. For instance, it can be assured that an individually contracted service is provided at all and in the negotiated timeframe. For this purpose, all clauses that are relevant to evaluate whether the contract is met also have to be represented in our formal representation language (even if they are not customizable).

5 ONTOLOGY FRAMEWORK

In order to be able to pass down the contract negotiation and execution to the system level, knowledge about the contracts and their interpretation has to be expressed in a machine interpretable way. Thus, a well-defined, formal representation is required that allows heterogeneous systems to understand, close, and enforce the contracts. In recent years, ontologies emerged as state of the art for knowledge sharing in distributed, heterogeneous environments. An ontology is a set of logical axioms that formally define a shared vocabulary (Gruber, 1993). By committing to a common ontology agents can make assertions or ask queries that are understood by the other agents. By featuring logic-based representation languages ontologies provide executable calculi that allow querying and reasoning during run-time, which is required comparing attribute values in the contracting and compliance monitoring phase.

5.1 Ontology Formalism

In order to guarantee that the formal definitions are understood by other parties in the web, the underlying logic has to be standardized. The Web Ontology Language (OWL) (W3C, 2004) standardized by the World Wide Web Consortium (W3C) is a first effort in this direction. OWL-DL is a decidable fragment of OWL and is based on a family of frame-based knowledge representation formalisms called Description Logics (DL) (Baader et al., 2003). The main elements of OWL are *individuals*, *properties* that relate individuals to each other and *classes* that group together individuals which share some common characteristics. Classes as well as properties can be put into subsumption hierarchies. Furthermore, OWL allows for describing classes in terms of complex *class constructors* that pose restrictions on the properties of a class. For example, the statement $BigCity \sqsubseteq City \sqcap \exists connectedTo.Highway$ describes the class of big cities, which are cities connected to some highway.[b] The inclusion axiom '$\sqsubseteq$' means that any big city is connected to some highway, but not any city connected to a highway is also necessarily big, which would be achieved by using the equality-axiom '$\equiv$' instead. Similarly, an explicit subclass relationship can be expressed by a statement like $BigCity \sqsubseteq InterestingCity$, saying that any big city is also interesting. Individuals can be related to classes and assigned values by a statements like $BigCity(Munich)$, $locatedIn(Munich, Germany)$, and $population(Munich, 1314551)$. Besides introducing Munich as a big German city, this statement also includes a data value for the city's population, which is supported by OWL for various datatypes such as integer or string. An OWL ontology consists of statements like the ones above, considered logical axioms from which an agent can draw logical consequences. For example, given an ontology O consisting of the above statements, it follows that Munich is an interesting city, which is denoted by $O \models InterestingCity(Munich)$. For a full list of concept and role constructors the interested reader is refereed to (Baader et al., 2003;

[b] Concepts, relations and rules contained in the ontology are highlighted using the *slanted style*.

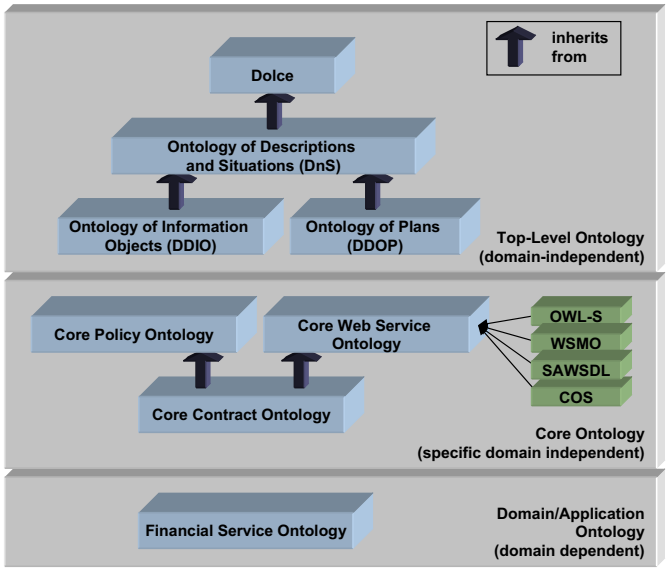

Figure 2 Ontology framework for Web service markets.

Horrocks et al., 2003; W3C, 2004).

In order to define the contract ontology, we require additional modeling primitives not provided by OWL (e.g. triangle relations between concepts). The Semantic Web Rule Language (SWRL) (Horrocks and Patel-Schneider, 2004) allows us to combine rule approaches with OWL and thus model such knowledge. Since reasoning with knowledge bases that contain arbitrary SWRL expressions usually become undecidable (Horrocks and Patel-Schneider, 2004), we restrict ourselves to DL-safe rules (Motik et al., 2005). To query and reason over a knowledge base containing OWL-DL as well as DL-safe SWRL axioms we use the KAON2 inference engine.[c] We define DL axioms either in DL abstract syntax (Baader et al., 2003) (denoted by (A1) to (An)). Moreover, for the reader's convenience we illustrate the modeling approach informally via UML class diagrams (Brockmans et al., 2004). For representing rules we rely on the standard rule syntax as done in (Horrocks and Patel-Schneider, 2004) (denoted by (R1) to (Rn)).

5.2 Overview

Ontologies can be categorized into four major classes: top-level ontologies, core ontologies, domain/task ontologies and application ontologies. The ontology framework is structured according to these categories by providing a stack of ontology modules. This structure is illustrated in Figure 2 and comprises the following three layers:

Top-level Ontology. The framework is based on a philosophically sound formalization of domain-independent concepts and relations that are captured by the top-level ontology DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) (Masolo et al., 2002). By capturing typical *ontology design patterns* (e.g. location in space and time), foundational ontologies provide basic vocabulary for structuring and formalization

[c] Available at `http://kaon2.semanticweb.org`.

of application ontologies. Reusing these building blocks considerably reduces modeling effort. Furthermore, they provide precise concept definitions through a high axiomatization. Thereby, foundational ontologies facilitate the conceptual integration of different languages and thus ensure interoperability in heterogenous environments. By providing additional modules DOLCE supports important ontology design patterns such as the Ontology of Descriptions & Situations (DnS), Ontology of Plans (OoP), Ontology of Information Objects (OIO) (Gangemi et al., 2004). Descriptions & Situations provides a theory of contextualization by introducing the distinction between descriptive and ground entities. Descriptive entities are aggregated by a *DnS:SituationDescription*[d] and represent non-physical objects like product descriptions or legal norms. Ground entities derived from DOLCE constitute a *DnS:Situation* that represents information about a concrete state of affaire in the world such as a concrete web service invocation or a legal case. Furthermore, the *DnS:satisfies*-relation between a *DnS:Situation* and a *DnS:SituationDescription* specifies if the descriptive entities "describe" the *DnS:Situation* according to specified rules. Moreover, we rely on the ontology module Ontology of Plans (DDOP) to describe social and cognitive plans such as goals or tasks and the module Ontology of Information Objects (DDIO) which introduces primitives for describing information items.

Core Ontologies. By means of the DOLCE vocabulary additional ontology modules can be defined that capture general information not specific to a certain domain and application. For expressing the formal Web service model, three core ontology modules are required:
(*i*) The *Core Policy Ontology* (CPO) introduced in (Oberle et al., 2006; Lamparter et al., 2006) formalizes the notion of policies and thus enables the representation of obligations. The structure of the Core Policy Ontology is in line with the ontology design pattern Descriptions & Situations. In general, the ontology distinguishes between a *CPO:PolicyDescription* specializing a *DnS:SituationDescription* and a concrete *CPO:Configuration* modeled as *DnS:Situation*. A *CPO:Configuration* can be evaluated according to the *CPO:PolicyDescription*. A *CPO:PolicyDescription* contains *CPO:PolicyObjects* and *CPO:PolicyObjects* which are modeled as *DnS:Functional Roles* played by concrete entities in the world. Using the concept of *DnS:Functional Roles* allows specifying policies on an abstract level without referring to concrete entities. This is also essential for modeling contracts, since they are usually applicable in many different settings. Further, the *CPO:PolicyDescription* determines the *OoP:Task* that is regulated as well as *CPO:Attributes* defining the constraints that have to be met to fulfill a certain *CPO:PolicyDescription*. By refining the *DnS:attitudeTowards*-relation we can model different deontic relations between *CPO:PolicyObject* and *OoP:Task*, e.g. we can say that a *CPO:PolicyObject* either has the *DnS:rightTo* or is *DnS:obligedTo* execute a certain *OoP:Task*. In order to find out whether a certain *DnS:Situation* conforms to a *CPO:PolicyDescription* a specialization of the *DnS:satisfies*-relation is used.
(*ii*) The *Core Software Ontology* (CSO) (Oberle, 2005; Oberle et al., 2006) introduces means for describing software systems. It provides a clear distinction between information (e.g. software or data) and the digital realization of information in information systems by introducing *CSO:ComputationalObjects* and *CSO:ComputationalTasks*. By augmenting the Core Software Ontology with service ontologies such as OWL-S (Sycara et al., 2003), WSMO (Dumitru et al., 2005) or COS (Oberle, 2005) further Web service specific aspects can be modeled, including service functionality and quality of service criteria.
(*iii*) Based on these two modules this paper contributes the *Core Contract Ontology* (CCO)

[d] For concepts and relations that are introduced directly in the contract ontology, namespaces are omitted. For those that are derived from other ontologies, the corresponding namespace is mentioned explicitly.

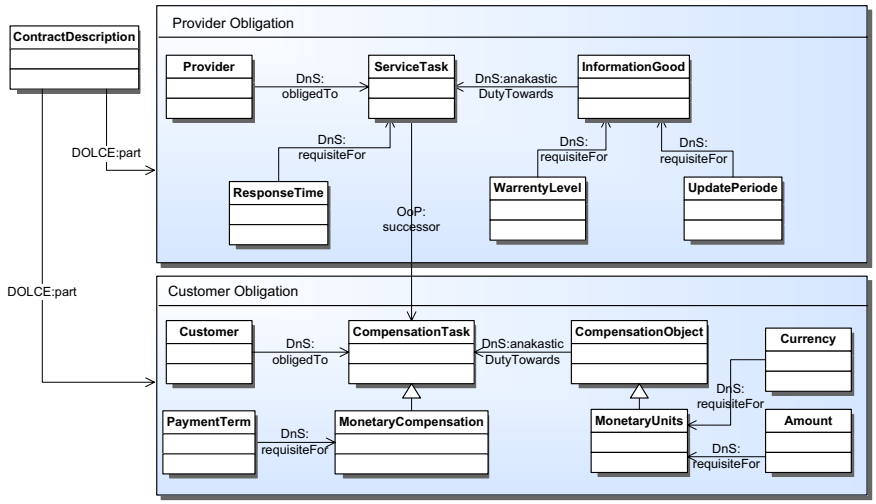

Figure 3 Contract ontology. Note that plotting UML classes within an *Obligation*-class illustrates a *DnS:defines*-relation between the *Obligation* and the contained classes.

which can be used to describe Web service contracts and to represent information Web service executions.

Domain and Application Ontologies. While the first two layers contain domain-independent off-the-shelf ontologies, the third layer comprises ontologies for customizing the framework to specific domains (e.g. an ontology for modeling credit information). While the core ontologies provide us with basic primitives to implement the Web service contract model, domain ontologies are particularly required for expressing attribute values.

In the following section we present the Core Contract Ontology. Section 6 introduces the modeling primitives for representing Web service contracts. Subsequently, in Section 7 we discuss how Web service executions can be modeled. Finally, Section 8 discusses how such executions can be checked for compliance with respect to a specific contract and thus shows how contract evaluation can be implemented based on the ontology.

6 CONTRACT REPRESENTATION

In this section, we show how contract information can be represented by reusing the Core Policy Ontology. In doing so, goal policies are used to represent obligations and permissions in a contract. After introducing this general contract ontology, we exemplify its usage by modeling the content of the individual contract identified in Section 4.

6.1 General contract ontology

A contract can be seen as a set of obligations and rights that are binding on all contractors. In the case of Web services we restrict ourselves to contracts between exactly two parties, namely Provider and Customer. We model this by introducing a *ContractDescription* as a *DnS:SituationDescription* containing only *Obligations* and *Permission* (Axiom (A1)). *Obligations* and *Permissions* are *CPO:PolicyDescriptions* that represent obligations

and permissions for *ContractParties*. An *Obligation* is a *CPO:PolicyDescription* where the *DnS:attitudeTowards*-relation is refined to *DnS:obligedTo* (Rule (R1)) and a *Permission* a *CPO:PolicyDescription* where it is refined to *DnS:rightTo* (Rule (R2)). A *ContractParty* is an active *CPO:PolicyObject* that is played by *DnS:Agents*. This can be formalized using the following axioms and rules:

$$\begin{aligned} \mathit{ContractDescription} \equiv\ & \mathit{DnS{:}SituationDescription} \sqcap \\ & \forall \mathit{DOLCE{:}part}.(\mathit{Obligation} \sqcup \mathit{Permission}) \end{aligned} \tag{A1}$$

$$\mathit{ContractParty} \sqsubseteq \mathit{CPO{:}PolicyObject} \sqcap \forall \mathit{DnS{:}playedBy}.\mathit{DnS{:}Agent} \tag{A2}$$

$$\begin{aligned} \mathit{Obligation}(x) \leftarrow\ & \mathit{CPO{:}PolicyDescription}(x), \mathit{DnS{:}defines}(x,y), \mathit{ContractParty}(y), \\ & \mathit{DnS{:}defines}(x,z), \mathit{CPO{:}PolicyTask}(z), \mathit{DnS{:}obligedTo}(y,z) \end{aligned} \tag{R1}$$

$$\begin{aligned} \mathit{Permission}(x) \leftarrow\ & \mathit{CPO{:}PolicyDescription}(x), \mathit{DnS{:}defines}(x,y), \mathit{ContractParty}(y), \\ & \mathit{DnS{:}defines}(x,z), \mathit{CPO{:}PolicyTask}(z), \mathit{DnS{:}rightTo}(y,z) \end{aligned} \tag{R2}$$

A *ContractDescription* defines a set of *Obligations* (Axiom (A1)), where each *Obligation* specifies a *ContractParty* and a *CPO:PolicyDescription*. Consequently, as depicted in Figure 3 the most elementary contract about purchasing Web services in exchange for money results in two simple Obligations:

Provider Obligation. A *ProviderObligation* specifies that the provider is obliged to make certain functionality accessible to the customer (Axiom (A3)). This functionality is represented by a *CPO:PolicyTask ServiceTask*, which is played by a *COS:WebService* in a *DnS:Situation* (Axiom (A5)). In addition, a *CPO:PolicyObject Provider* is introduced that is *DnS:obligedTo* provide the *ServiceTask* (Axiom (A4)). A *CPO:PolicyObject InformationGood* is used to represent the information that has to be returned by the *COS:WebService* playing the *ServiceTask*. Note that the distinction between *ServiceTasks* and *InformationGood* allows modeling the functionality of a service using either explicit or implicit capability representation (Sycara et al., 2003). This enables our contract ontology to support major efforts striving for semantic web service descriptions such as WSMO (Dumitru et al., 2005), OWL-S (Sycara et al., 2003) and WSDL-S (Patil et al., 2004).

$$\mathit{ProviderObligation} \sqsubseteq \mathit{Obligation} \sqcap \exists \mathit{DnS{:}defines}.\mathit{Provider} \tag{A3}$$

$$\mathit{Provider} \sqsubseteq \mathit{ContractParty} \sqcap \exists \mathit{DnS{:}obligedTo}.\mathit{ServiceTask} \tag{A4}$$

$$\mathit{ServiceTask} \sqsubseteq \mathit{CPO{:}PolicyTask} \sqcap \forall \mathit{DnS{:}sequences}.\mathit{Service} \tag{A5}$$

$$\mathit{InformationGood} \sqsubseteq \mathit{CPO{:}PolicyObject} \sqcap \tag{A6}$$

$$\forall \mathit{DnS{:}playedBy}.\mathit{OIO{:}InformationObject} \tag{A7}$$

Customer Obligation. A *CustomerObligation* which specifies that the customer is obliged to compensate the provider for using the Web service (Axiom (A8)). This activity is called *CompensationTask* and mostly involves the transfer of a certain amount of money. To define a *CompensationTask* the *CPO:PolicyTask* is specialized to a *CompensationTask* (Axiom A10). A *Customer* is a *ContractParty* that is obliged to carry out a *CompensationTask* (Axiom A9). Moreover, a *CompensationTask* may involve a *CPO:PolicyObject CompensationObject*, which refers to a passive physical or social entity (*DOLCE:NonAgentiveSocialObject* or *DOLCE:NonAgentivePhysicalObject*) such as money or a patent (Axiom (A11)).

$$CustomerObligation \sqsubseteq Obligation \sqcap \exists DnS{:}defines.Customer \sqcap \exists DnS{:}defines.CompensationTask \quad \text{(A8)}$$

$$Customer \sqsubseteq ContractParty \sqcap \exists DnS{:}obligedTo.CompensationTask \quad \text{(A9)}$$

$$CompensationTask \sqsubseteq CPO{:}PolicyTask \sqcap \forall DnS{:}anakasticDutyTowards^{-}.CompensationObject \quad \text{(A10)}$$

$$CompensationObject \sqsubseteq CPO{:}PolicyTask \sqcap \exists .DnS{:}anakasticDutyTowards.CompensationTask \sqcap \forall DnS{:}playedBy.(DOLCE{:}NonAgentiveSocialObject \sqcup DOLCE{:}NonAgentivePhysicalObject) \quad \text{(A11)}$$

Usually contracts also specify in which sequence obligations have to be fulfilled and rights are obtained. In the basic contract outlined above the *ServiceTask* has to be executed before the *CompensationTask*. Hence, means for representing sequences of *OoP:Tasks* are required. We reuse the Ontology of Plans which provides primitives for modeling complex processes, e.g. *Sequential Tasks*, *Parallel Tasks*, *Loop Tasks*, etc. In this context, the primary ordering relation for *OoP:Tasks* are *OoP:directSuccessor* and its transitive version *OoP:successor.*

As illustrated in Figure 3, concrete obligations are expressed via policies specifying *CPO:Attribute* for *ServiceTask* and *InformationGood* or *CompensationTask* and *MonetaryUnit*, respectively. How this can be realized for the individual contract clauses identified in Section 4 is shown in the next section.

6.2 Individual Contract Clauses

As discussed above, a contract imposes further obligations and permissions that have to be fulfilled by the contractors. These obligations and permissions are modeled within a *CPO:PolicyDescription* by introducing specialized *CPO:Attribute* concepts and specifying the allowed *CPO:AttributeValue* for this *CPO:Attribute*. In the following, we briefly discuss some examples how the obligations that have to be defined in an individual contract can be formalized. Methodologically this is realized by transforming a natural language contractual clause into a formalized goal policy. However, note that this is no exhaustive enumeration. Depending on the service types and scenarios a wide range of different *CPO:Attributes* are possible.

Scope of Agreement (§1) The scope of an agreement defines the type of *InformationGoods* or *CompensationObjects* that are valid according to the contract. This specification can be done by specializing the concepts *InformationGood* or *CompensationObject* using a domain ontology as presented in Figure 1 for credit information.

Provider Obligation (§2) As specified above, in this category service levels a provider has to meet can be specified. We exemplify this by considering the *CPO:Attribute UpdatePeriod* which is warranted by the provider. A legal text negotiated by human beings could read as follows:

> "*The Provider warrants that it reviews and, if necessary, updates Credit Information every month.*"

Since the timeliness is a property of the provided *InformationGood*, we introduce *UpdatePeriod* as a subclass of *CPO:Attribute*. *UpdatePeriod* constraints the set of allowed *CPO:AttributeValues UpdatePeriodValue*. This is captured by the following axiom:

$$\begin{aligned} \mathit{UpdatePeriod} \sqsubseteq\ & \mathit{CPO{:}Attribute} \sqcap \\ & \exists \mathit{DnS{:}requisiteFor}.\mathit{InformationGood} \sqcap \\ & \forall \mathit{DnS{:}requisiteFor}.\mathit{InformationGood} \sqcap \\ & \exists \mathit{DnS{:}valuedBy}.\mathit{UpdatePeriodValue} \end{aligned} \tag{A12}$$

The *CPO:Attribute UpdatePeriod* is illustrated in Figure 3.

Use of information (§3) This category specifies how the customer may use the information. For example, consider licenses that typically regulate how certain information can be used. A agreed legal text could read as follows:

> *"The Provider grants the customer a non-transferable license to use the Credit Information delivered under the terms of this contract. The Customer may freely copy or forward Credit Information within its company. The Customer may not disclose or make the Credit Information otherwise available to third parties without prior consent of the Provider."*

The license specifies if the right to use a certain *InformationGood* is transferable, if the customer may disclose the *InformationGood* within the company ('Disclose within Company') or to external third parties ('Disclose to 3rd Party'). In order to facilitate contract monitoring, we model the right as an *Obligation* that specifies which alternatives are not allowed. This is realized by introducing an additional *CustomerObligation DisclosureObligation* (Axiom (A13)) with the *CompensationTask TransferInformation* (Axiom (A14)) and the *CPO:Attribute AdmissibleParty*. *AdmissibleParty* may take the values 'Not Transferable', 'Disclose within Company' and 'Disclose to 3rd Party' (Axiom (A15)). The following axioms capture this information. Note that the corresponding *Obligation* is omitted in Figure 3.

$$\begin{aligned} \mathit{DisclosureObligation} \sqsubseteq\ & \mathit{CustomerObligation} \sqcap \\ & \exists \mathit{DnS{:}defines}.\mathit{TransferInformation} \sqcap \\ & \exists \mathit{DnS{:}defines}.\mathit{AdmissibleParty} \end{aligned} \tag{A13}$$

$$\begin{aligned} \mathit{TransferInformation} \sqsubseteq\ & \mathit{CompensationTask} \sqcap \\ & \mathit{DnS{:}requisites}.\mathit{AdmissibleParty} \end{aligned} \tag{A14}$$

$$\begin{aligned} \mathit{AdmissibleParty} \sqsubseteq\ & \mathit{CPO{:}Attribute} \sqcap \\ & \exists \mathit{DnS{:}requisiteFor}.\mathit{TransferInformation} \sqcap \\ & \forall \mathit{DnS{:}requisiteFor}.\mathit{TransferInformation} \sqcap \\ & =_1 \mathit{DnS{:}valuedBy}.\{\text{'Not Transferable'}, \\ & \text{'Disclose within Company'}, \\ & \text{'Disclose to 3rd Party'}\} \end{aligned} \tag{A15}$$

Warranties and Liabilities (§4) In this category warranty and liability levels can be defined. In legal practice a wide range of different warranty and liability regulations are used. In this example, we consider a very simple approach, where automatically one level from a predefined set of warranty levels can be chosen. The predefined warranty levels are defined in the umbrella contract. In a natural language contract a level can be defined as follows:

"*The Provider warrants that the credit information is 100% accurate.*"

As shown in Figure 3, this can be realized by adding a *CPO:Attribute WarrantyLevel* to the *ProviderObligation* which is valued by a *DOLCE:Region* reflecting the three different warranty levels: 'No Warrenty', 'Uttermost Care', and 'Full Warranty'. Since the warranty can be considered as a fundamental property of a *InformationGood*, we model *WarrantyLevel* as a *CPO:Attribute* of *InformationGood*.

$$\begin{aligned} \mathit{WarrantyLevel} \sqsubseteq & \mathit{CPO{:}Attribute} \sqcap \\ & \exists \mathit{DnS{:}requisiteFor}.\mathit{InformationGood} \sqcap \\ & \forall \mathit{DnS{:}requisiteFor}.\mathit{InformationGood} \sqcap \\ & =_1 \mathit{DnS{:}valuedBy}.\{\mathit{NoWarrenty}', \text{`}\mathit{UttermostCare}', \\ & \text{`}\mathit{FullWarranty}'\} \end{aligned} \tag{A16}$$

Delivery Time (§5) In many applications delivery time is a crucial property that heavily influences the prices. It is also a property that often has to be customized dynamically, e.g., in order to adapt the contract to changing Web server load. A natural language clause could be formulated as follows:

"*The Provider shall deliver the Credit Information within five seconds after conclusion of the contract.*"

In the context of Web services, delivery time usually refers to the *response time*, in which the result is returned by the service. The *CPO:Attribute ResponseTime* specifies the period in which the *Service Task* has to be executed. Hence, it is modeled as a constraint of *ServiceTask* which is *DnS:valuedBy* an *CPO:AttributeValue ResponseTimeValue*. The approach is illustrated in Figure 3 and captured by the following axiom:

$$\begin{aligned} \mathit{ResponseTime} \sqsubseteq & \mathit{CPO{:}Attribute} \sqcap \exists \mathit{DnS{:}requisiteFor}.\mathit{ServiceTask} \sqcap \\ & \forall \mathit{DnS{:}requisiteFor}.\mathit{ServiceTask} \sqcap \\ & =_1 \mathit{DnS{:}valuedBy}.\mathit{ResponseTimeValue} \end{aligned} \tag{A17}$$

Prices and Payment Terms (§6) Usually the most important aspect regulated in a contract is the price that has to be paid by the customer for invoking the service. Prices of services may change frequently or are even determined dynamically in a negotiation or auction process. For example, a corresponding clause could be simply specified as follows:

"*The price for the provided credit information is EUR 15.*"

Due to this importance we have defined a *Customer* as a *ContractParty* that is obliged to execute a *CompensationTask* (Axiom (A9)). The nature of this compensation is left open and can be defined by constraining the allowed alternatives using policies. For the case where no compensation is required (e.g. service usage is free of charge) simply no policies are defined for *CompensationTask*. For the usual case where a certain amount of money has to be paid we have specialized *CompensationTask* to *MonetaryCompensation* which requires the specification of the *MonetaryUnits* that have to be transferred from the customer to the provider (Axiom (A18)). *MonetaryUnits* are *CompensationObjects* which specify a certain *Amount* of money in a given *Currency* (Axiom (A19)). The

CPO:Attribute Amount is valued by exactly one floating number (Axiom (A20)) and the *CPO:Attribute Currency* is valued by exactly one *CurrencyValue* (Axiom (A21)). Thus, *CurrencyValue* comprises Euro, Dollar, Yen, etc. This is formalized by the following axioms.

$$\begin{aligned} \mathit{MonetaryCompensation} \sqsubseteq \mathit{CompensationTask} \sqcap \quad & \text{(A18)} \\ \forall \mathit{DnS{:}anakasticDutyTowards}^{-}.\mathit{MonetaryUnit} \sqcap \\ \exists \mathit{DnS{:}anakasticDutyTowards}^{-}.\mathit{MonetaryUnit} \\ \mathit{MonetaryUnits} \sqsubseteq \mathit{CompensationObject} \sqcap \quad & \text{(A19)} \\ \exists \mathit{DnS{:}requisites}.\mathit{Amount} \sqcap \\ \exists \mathit{DnS{:}requisites}.\mathit{Currency} \\ \mathit{Amount} \sqsubseteq \mathit{CPO{:}Attribute} \sqcap \quad & \text{(A20)} \\ =_1 \mathit{DnS{:}valuedBy}.\mathit{XSD{:}Float} \\ \mathit{Currency} \sqsubseteq \mathit{CPO{:}Attribute} \sqcap \quad & \text{(A21)} \\ =_1 \mathit{DnS{:}valuedBy}.\mathit{CurrencyValue} \end{aligned}$$

Furthermore, a contract usually contains a *PaymentTerm* that specifies in which timeframe a *MonetaryCompensation* has to take place. We model the *PaymentTerms* as a *CPO:Attribute* constraining *MonetaryCompensation* as shown in Figure 3.

$$\begin{aligned} \mathit{PaymentTerm} \sqsubseteq \mathit{CPO{:}Attribute} \sqcap \quad & \text{(A22)} \\ \exists \mathit{DnS{:}requisiteFor}.\mathit{MonetaryCompensation} \sqcap \\ \forall \mathit{DnS{:}requisiteFor}.\mathit{CompensationTask} \sqcap \\ =_1 \mathit{DnS{:}valuedBy}.\mathit{DOLCE{:}Temporal\text{-}Region} \end{aligned}$$

All regulations specified above can be extended either by introducing new *CPO:Attributes* within an existing *CPO:PolicyDescription* or by adding further *Obligations* or *Permissions* to the *ContractDescription*.

6.3 Domain Ontology

In order to apply the contract ontology in a concrete application scenario, domain ontologies are required to introduce concepts and relations required for specializing *TradingObjects* as well as *COS:WebService* and *CompensationTasks*. Since in our credit information scenario we deal with information services, the functionality of a service can be specified by introducing the *Send Information* and *MonetaryCompensation* tasks, which concretize *ServiceTask* and *CompensationTask*, respectively. Furthermore, in order to define the functionality of a service specifying the service output is required, which can be done by means of a reference to a specific type of information. In the following we discuss a domain ontology dealing with *TradingObjects* in the credit information services example. As introduced above, there are five main categories of Credit Information. Ontologically, information mentioned above is represented by the concept *CSO:Data* that is *OIO:realized* by a *CSO:ComputationalObject* within the information system. Hence, we formally define different types of *CreditInformation* as follows:

$$\mathit{CreditInformation} \sqsubseteq \mathit{CSO{:}Data} \sqcap \exists \mathit{DOLCE{:}part}.\mathit{CompanyIdentifier} \sqcap \quad \text{(A23)}$$

$$\forall DOLCE{:}part.(BusinessBackgroundInformation \sqcup CreditScoreCalculationInformation \sqcup QualityOfCompanyInformation \sqcup CreditLimitCalculation \sqcup WarningInformation)$$

$$BusinessBackgroundInformation \sqsubseteq CSO{:}Data \sqcap \exists DOLCE{:}part.CompanyIdentifier \sqcap \forall DOLCE{:}part.(OwnershipInformation \sqcup History \sqcup Principles \sqcup Operations \sqcup Location) \quad \text{(A24)}$$

$$CompleteBusinessBackgroundInformation \sqsubseteq CSO{:}Data \sqcap \exists DOLCE{:}part.CompanyIdentifier \sqcap \exists DOLCE{:}part.Ownership \sqcap \exists DOLCE{:}part.History \sqcap \exists DOLCE{:}part.Principles \sqcap \exists DOLCE{:}part.Operations \sqcap \exists DOLCE{:}part.Location \quad \text{(A25)}$$

$$OwnershipInformation \sqsubseteq CSO{:}Data \sqcap \exists DOLCE{:}part.CompanyIdentifier \sqcap \exists DOLCE{:}part.Ownership \quad \text{(A26)}$$

The other types of *CreditInformation* are defined analogously. Note that this modeling approach enables a DL-reasoner to automatically infer a *CreditInformation* hierarchy, e.g. it is inferred that Ownership Information is a subclass of *BusinessBackgroundInformation* as well as *CreditInformation* or that *CompleteBusiness BackgroundInformation* is a subclass of *BusinessBackgroundInformation*. Such a hierarchy can be utilized to provide interoperability in the contracting and contract monitoring process since it enables improved matching of requested and provided information. Therefore, Rule (R3) defines a matching rule that enables to compare two *CreditInformation* concepts using a built-in *subsumes*, which implements a subsumption checking algorithm.

$$match(x, y) \leftarrow CreditInformation(x), CreditInformation(y), subsumes(y, x) \quad \text{(R3)}$$

A similar approach as presented here for the type of credit information provided by a service can be realized for all attributes of a Web service contract. While some might require complex descriptions of attribute values realized by ontologies, others might be valued by simple datatypes such as integers or strings. In this case, matching rules can be defined using simple datatype operators such as '=' or '<' instead of the *subsumes*-predicate. Of course, depending on the domain ontology other notions of match besides subsumption might be required, which are widely discussed in description logic literature (Noia et al., 2003; Li and Horrocks, 2003; Grimm et al., 2004). A more thorough discussion of using such customizable matching rules can be found in (Lamparter et al., 2007).

7 REPRESENTING MONITORING INFORMATION

In the last section, we presented contract information as a collection of *CPO:PolicyDescriptions* which are modeled by refining *DnS:SituationDescriptions*. In this section, we extend this approach in order to represent information about the execution of a contract. We call such information *monitoring information* and represent it by means of the *DnS:Situation MonitoringInformation* (Axiom (A27)). *MonitoringInformation* is modeled as specialization of *CPO:Configuration* and represents values of *CPO:Attributes* used in the Web service execution. A *CPO:Configuration* identifies the value of an *CPO:Attribute* belonging to an *DOLCE:Endurant* or *DOLCE:Perdurant*. Since we are dealing only with monitoring Web service invocations, we can specialize our modeling approach. The Core Software Ontology (CSO) and the Core Ontology of Service (COS) (Oberle, 2005) introduce the fundamental concepts required for describing software systems. According to (Oberle, 2005), the main entities living in the computational domain are *CSO:ComputationalObjects* and *CSO:ComputationalActivities*. *CSO:ComputationalObjects* can be regarded as concrete realization of *CSO:Software* or *CSO:Data*.[e] The execution of *CSO:Software* triggers *CSO:ComputationalActivities* and these *CSO:ComputationalActivities* may involve *CSO:Data*. Rule (R4) and (R5) capture this active and passive aspect by introducing the relations *executes* and *involvedIn*, respectively. Each *MonitoringInformation* instance has to contain at least one *CSO:ComputationalActivity* that is monitored (Axiom (A27)). As for *CPO:Configurations* in general, each *CSO:ComputationalActivity* and *CSO:ComputationalObject* may exhibit certain properties that are captured by *DOLCE:Qualities*.

$$\begin{aligned} \mathit{MonitoringInformation} \sqsubseteq\ & \mathit{CPO{:}Configuration} \sqcap \\ & \exists \mathit{DnS{:}settingFor}.\mathit{CSO{:}ComputationalActivity} \sqcap \\ & \forall \mathit{DnS{:}settingFor}.(\mathit{CSO{:}ComputationalActivity} \sqcup \\ & \mathit{CSO{:}ComputationalObject} \sqcup \\ & \mathit{DOLCE{:}Quality} \sqcup \mathit{DOLCE{:}Region}) \end{aligned} \tag{A27}$$

$$\begin{aligned} \mathit{executes}(x,y) \leftarrow\ & \mathit{CSO{:}Software}(x), \mathit{OIO{:}expresses}(x,z), \\ & \mathit{OoP{:}Plan}(z), \mathit{DnS{:}defines}(z,t), \mathit{CSO{:}ComputationalTask} \\ & \mathit{DnS{:}sequences}(t,y)\mathit{CSO{:}ComputationalActivity}(y) \end{aligned} \tag{R4}$$

$$\begin{aligned} \mathit{involvedIn}(x,y) \leftarrow\ & \mathit{CSO{:}Data}(x), \mathit{OIO{:}realizedBy}(x,z), \\ & \mathit{DOLCE{:}participantIn}(z,y), \\ & \mathit{CSO{:}ComputationalActivity}(y) \end{aligned} \tag{R5}$$

Providing information via a Web service leads to a *CSO:ComputationalActivity* where one party transfers a *CSO:ComputationalObject*, e.g. credit information, to another party. In executing this activity various types of monitoring information about the activity itself as well as about participating objects can be measured or perceived, which are represented as *DOLCE:Qualities* of the corresponding *CSO:ComputationalActivity* or *CSO:ComputationalObject*.

Example 1. *Figure 4 introduces a concrete example which represents information about a specific Web service invocation as an instance of MonitoringInformation. Consider the execution of a CSO:ComputationalActivity 'Send' carried out on February 27th, 2006*

[e] Note that *CSO:Software* can be seen as a special form of *CSO:Data*, viz., $\mathit{CSO{:}Software} \sqsubset \mathit{CSO{:}Data}$.

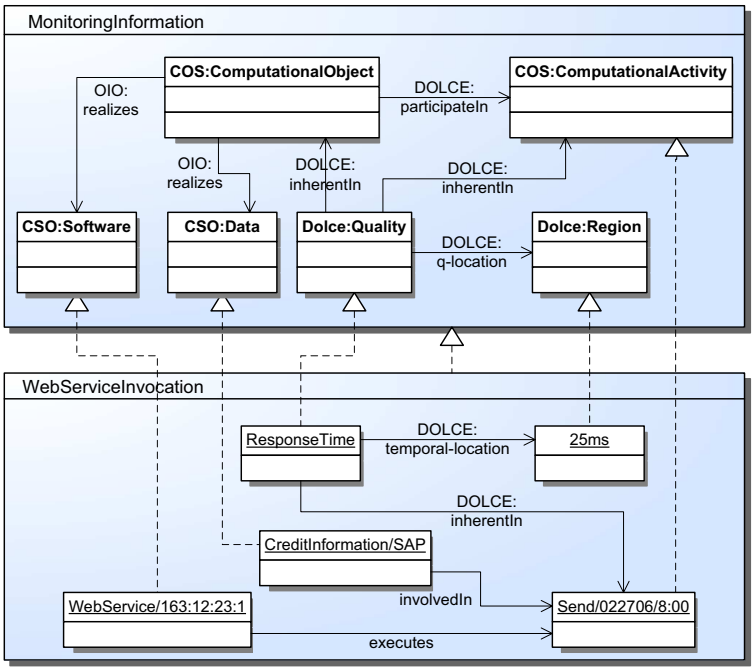

Figure 4 Representing *MonitoringInformation* as *DnS:Situation*. Note that plotting UML classes within a *DnS:Situation*-class illustrates a *DnS:settingFor*-relation between the *DnS:Situation* and the contained classes.

at 8am. The activity was executed by a Web service with the IP-address 163:12:23:1 and involved the digital representation of credit information of the company SAP. According to the Core Ontology of Services (Oberle, 2005), a COS:WebService is a specialization of CSO:Software and thus we model 'WebService/163:12:23:1' as an instance of the concept CSO:Software, while'Credit Information/SAP' is modeled as CSO:Data. Moreover, the DOLCE:Quality ResponseTime of the Send*-activity is measured and represented by the DOLCE:Region '25ms'. Of course, other DOLCE:Qualities of the CSO:ComputationalActivity as well as the CSO:ComputationalObject beyond 'ResponseTime' can be measured and represented in a similar way.*

8 AUTOMATED COMPLIANCE MONITORING

A major advantage of machine-interpretable, formal contracts is the fact that monitoring whether a Web service execution complies with the contract can be (at least partially) automated. This is particularly important in a scenario where many (different) contracts are closed and executed within a short time, which is usually the case for large-scale service-oriented architectures. Having introduced the *ContractDescription* in Section 6 and *MonitoringInformation* capturing information about contract execution in Section 7, we introduce in this section how the compliance of *MonitoringInformation* with respect to concrete *ContractDescription* can be verified.

Since terms within contracts are often context-dependent and require fuzzy interpretations the evaluation process requires additional interpretations. For example, certain

obligations have to be done “immediately” or “with utmost care”. Although such terms are interpreted in various different ways, lawyers typically have guidelines how to interpret such statements and expressions in a given context. Since these guidelines are not part of the contract, we have to add them to the knowledge base in a formalized way, which allows to include them in the contract evaluation process. For instance, if the term “immediately” is used to specify a timeframe in which a response of the service is expected, one could use the following rule of thumb: considering the current state of the art a response is received “immediately” only if it is received within 5 seconds after sending the request. Subsequently, we exemplify this approach using the a simple *ProviderObligation*.

Example 2. *The credit information service provider X has to provide a complete set of* Business Background Information *of company SAP to customer Y. This has to be done immediately after receiving the customer's request. Therefore, we derive the following formal definition of the* Provider Obligation*:*

$$
\begin{aligned}
&\mathit{Obligation}(\mathit{ProviderObligationX})\\
&\mathit{Provider}(X)\\
&\mathit{DnS{:}defines}(\mathit{ProviderObligationX}, X)\\
&\mathit{InformationGood}(\mathit{BBInformation}/Z)\\
&\mathit{DnS{:}defines}(\mathit{ProviderObligationX}, \mathit{BBInformation}/\mathit{SAP})\\
&\mathit{ServiceTask}(\mathit{Deliver})\\
&\mathit{DnS{:}defines}(\mathit{ProviderObligationX}, \mathit{Deliver})\\
&\mathit{ResponseTime}(\mathit{responseTimeX})\\
&\mathit{DnS{:}defines}(\mathit{ProviderObligationX}, \mathit{responseTimeX})\\
&\mathit{ResponseTimeValue}('\mathit{immediately}')\\
&\mathit{DnS{:}valuedBy}(\mathit{responseTimeX}, '\mathit{immediately}')\\
&\mathit{DnS{:}obligedTo}(X, \mathit{Transfer})\\
&\mathit{DnS{:}anakasticDutyTowards}(\mathit{BBInformation}/Z, \mathit{Transfer})\\
&\mathit{DnS{:}requisiteFor}(\mathit{responseTimeX}, \mathit{Transfer})
\end{aligned}
$$

Assume the requester monitored the execution of the contract above and observed the *MonitoringInformation* shown in Figure 4. Based on the *MonitoringInformation* the requester evaluates the *ProviderObligation*. In legal practise typically a scheme of questions is applied to determine the source of violation. The following questions exemplify this approach using the obligation defined in Example 2:

1. Was the requested trading object delivered? To answer this question we have to find out whether information is delivered by the provider at all and - in case it is - whether the delivered information is complete with respect to the agreement in the *ContractDescription*. We realize this by comparing the delivered *CSO:ComputationalObject* contained in the *MonitoringInformation* with the *InformationGood* agreed on in the contract. Assuming the hierarchy of credit information presented in Figure 1, the following *match*-predicate might be defined for credit information:

$$
\begin{aligned}
\mathit{match}(c_r, c_o) \leftarrow\ & \mathit{CreditInformation}(c_r), \mathit{CreditInformation}(c_o),\\
& \mathit{subsumes}(c_o, c_r)
\end{aligned}
\qquad \text{(R6)}
$$

We thus allow the provider to send more information than required, while making sure that at least the information agreed on is provided. Based on this *match*-predicate we determine whether a delivered information is correct as follows:

$$\begin{aligned} correctInformation(m,c) \leftarrow & MonitoringInformation(m), \\ & ProviderObligation(c) DnS{:}defines(c,t), \\ & InformationGood(t), DnS{:}playedBy(t,d1), \\ & CSO{:}Data(d1), DnS{:}settingFor(m,d2), \\ & CSO{:}Data(d2), match(d1,d2) \end{aligned} \tag{R7}$$

Alternatively, the Rule R7 could also be expressed by means of a SPARQL-query. This would be more appropriate, e.g., if the evaluation is only done once.

2. Was the correct service task executed? In a similar way we can also answer this question by formulating the *correctActivity*-rule. This time the executed *CSO:ComputationalActivity* stated in the *MonitoringInformation* is compared to the *CSO:ComputationalActivity* agreed upon in the *ContractDescription*. Again we use a *match*-predicate that relies on the *subsumes*-predicate. Thereby, we allow a general activity description in the contract to be fulfilled by a more specific activity. For example, a contract might specify that certain information has to be transferred. How this should be done is not specified exactly. Therefore, sending by mail or telling on the phone might be admissible since both are certain types for delivering information.

$$\begin{aligned} correctActivity(m,c) \leftarrow & MonitoringInformation(m), \\ & ProviderObligation(c), DnS{:}defines(c,t), \\ & ServiceTask(t), DnS{:}sequences(t,a1), \\ & CSO{:}ComputationalActivity(a1), DnS{:}settingFor(m,a2), \\ & CSO{:}ComputationalActivity(a2), match(a1,a2) \end{aligned} \tag{R8}$$

3. Was the task executed within the required timeframe? According to the *ProviderObligation* a *ServiceTask* has to be executed within a certain time, which is denoted by *ResponseTime*. This is verified by the rule below, which compares the monitored execution time with the *ResponseTime* in the *ContractDescription*.

$$\begin{aligned} activityInTime(m,c) \leftarrow & MonitoringInformation(m), \\ & ProviderObligation(c), DnS{:}defines(c,t), \\ & ServiceTask(t), ResponseTime(d), \\ & DnS{:}requisiteFor(d,t), interpretedRT(d,r1), \\ & DnS{:}settingFor(x,a), CSO{:}ComputationalActivity(a), \\ & DOCLE{:}inherentIn(r2,a), \\ & DOLCE{:}Region(r2), match(r1,r2) \end{aligned} \tag{R9}$$

However, since *ResponseTime* is expressed by a *XSD:String* rather than a *DOLCE:Temporal-Region* we need a conversion rule. Note that this interpretation of the term ”immediately” is not content of the contract but rather common sense knowledge

modeled by a company's lawyers. The interpretation may also change from time to time (e.g. due to new court decisions) and thus Rule (R10) has to be adapted.

$$\begin{aligned} interpretedRT(d,v) \leftarrow & DnS{:}valuedBy(d,v), ResponseTime(d), \\ & DnS{:}valuedBy(d,r1), ResponseTimeValue(r1), \\ & swrlb{:}equals(r1,'immediately'), \\ & DOLCE{:}Temporal\text{-}Region(v), assigns('<5s',v) \end{aligned} \tag{R10}$$

After the conversion both measures are expressed via *DOLCE:Temporal-Regions* and thus can be compared by a corresponding *match*-predicate as done in Rule R9.

In order to answer the question whether a *ContractDescription* is satisfied by a *MonitoringInformation* the following query has to be executed:

```
SELECT ?C, ?M                                                    (R11)
 WHERE {
    ?C rdf:type CCO:ContractDescription .
    ?M rdf:type CCO:MonitoringInformation .
    ?M correctInformation ?C .
    ?M correctActivity ?C .
    ?M activityInTime ?C .}
```

If the query returns a non-empty result set, the contract has been fulfilled correctly. Otherwise there is at least one obligation violated. However, the above query does not give evidence about the reason of the violation. In order to determine the legal consequences of a violation one might want to specify more fine-grained questions, which can be realized by issuing separate queries.

Of course, similar questions about other elements of the obligations can be formulated and expressed via rules or queries. Thereby, a company- and domain-specific evaluation process can be assembled that includes interpretation rules where necessary.

9 CONCLUSION AND OUTLOOK

In this paper we presented an ontology framework for Web service contracts as well as contract monitoring information and an algorithm to check whether a contract has been executed correctly. The framework relies on existing internet standards and thereby facilitates interoperability in a Web environment. The approach does not aim towards full automation, but rather enables semi-automatic contract management. In our opinion full automation is at least for the moment not feasible across organizational boundaries. Therefore, we suggest combining an umbrella contract covering static aspects with a formalized description of clauses that can be automatically negotiated, closed, monitored and executed. Automation reduces the manual management effort and thus reduces costs for operating an inter-organizational service oriented architecture. In addition, automation enables dynamic re-configuration of the system at runtime, e.g. to deal with service failures.

We have implemented a prototype that enables dynamic contracting of Web services during the runtime of a business process. The systems relies on a WS-BPEL engine that

is used to execute a business process. In doing this, Web services are selected and contracted at runtime. More details on the prototype can be found in Lamparter et al. (2007). Up to now only monitoring information that are directly provided by the WS-BPEL engine (e.g. response time of a Web service) can be used for monitoring the contract. To realize more sophisticated contract monitoring, we plan to integrate additional algorithms and protocols for measuring and predicting service quality. First work in this direction is presented in Casati et al. (2003), Ludwig et al. (2004), and Wang et al. (2005).

Acknowledgements

Research reported in this paper has been financed by the German Research Foundation (DFG) within the scope of the Graduate School for Information Management and Market Engineering (DFG grant no. GRK 895). In addition, we thank the reviewers and participants of the HICSS'07 conference and the International Journal of Service Sciences for their valuable comments and suggestions.

References

Angelov, S. and Grefen, P. (2003). The 4w framework for b2b e-contracting. *Int. Journal on Networking and Virtual Organisations*, 1(3).

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory Implemenation and Applications*. Cambridge University Press.

Brockmans, S., Volz, R., Eberhart, A., and Löffler, P. (2004). Visual modeling of OWL DL ontologies using UML. In McIlraith, S. A., Plexousakis, D., and van Harmelen, F., editors, *Proceedings of the Third International Semantic Web Conference (ISWC)*, volume 3298 of *Lecture Notes in Computer Science*, pages 198–213, Hiroshima, Japan. Springer.

Casati, F., Shan, E., Dayal, U., and Shan, M.-C. (2003). Business-oriented management of web services. *Commun. ACM*, 46(10):55–60.

Cole, J. and Milosevic, Z. (2001). Extending support for contracts in ebXML. In *ITVE '01: Proceedings of the workshop on Information technology for virtual enterprises*, pages 119–127. IEEE Computer Society.

Daskalopulu, A. and Sergot, M. (1997). The representation of legal contracts. *AI and Society*, 11(1/2):6–17.

Dumitru, R., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., and Fensel:, D. (2005). Web service modeling ontology. *Applied Ontology*, 1(1):77 – 106.

Gangemi, A., Sagri, M.-T., and Tiscornia, D. (2004). A Constructive Framework for Legal Ontologies. Deliverable d07, EU 6FP METOKIS Project, Deliverable. Available from `http://metokis.salzburgresearch.at`.

Global Grid Forum (2006). Grid Resource Allocation Agreement Protocol. Web Services Specification. Available from `http://www.ogf.org/Public_Comment_Docs/Documents/Oct-2006/WS-AgreementSpecificationDraftFinal\_sp\_tn\_jpver\_v2.pdf`.

Governatori, G. (2005). Representing business contracts in ruleml. *International Journal of Cooperative Information Systems*, 14:181–216.

Governatori, G., Milosevic, Z., and Sadiq, S. (2006). Compliance checking between business processes and business contracts. In *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, pages 221–232, Los Alamitos, CA, USA. IEEE Computer Society.

Griffel, F., Boger, M., Weinreich, H., Lamersdorf, W., and Merz, M. (1998). Electronic contracting with COSMOS - how to establish, negotiate and execute electronic contracts on the internet. In Kobryn, C., Atkinson, C., and Milosevic, Z., editors, *2nd Int. Enterprise Distributed Object Computing Workshop (EDOC '98)*, page 10.

Grimm, S., Motik, B., and Preist, C. (2004). Variance in e-business service discovery. In *Semantic Web Services: Preparing to Meet the World of Business Applications, workshop at ISWC 2004*.

Grosof, B. and Poon, T. (2003). Sweetdeal: Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proceedings of the 12th World Wide Web Conference*, pages 340–349, Budapest, Hungary.

Gruber, T. R. (1993). A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220.

Hage, J. (1996). A theory of legal reasoning and a logic to match. *Artificial Intelligence and Law*, 4:199–273.

Hoffner, Y. and Field, S. (2005). Transforming agreements into contracts. *International Journal of Cooperative Information Systems*, 14(2-3):217–244.

Hohfeld, W. (1913). Some fundamental legal conceptions as applied in judicial reasoning. *Yale Law Journal*, 23.

Horrocks, I. and Patel-Schneider, P. F. (2004). A Proposal for an OWL Rules Language. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 723–731, New York, NY, USA. ACM Press.

Horrocks, I., Patel-Schneider, P. F., and van Harmelen, F. (2003). From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26.

IBM Corporation (2003). Wsla language specification, version 1.0. `http://www.research.ibm.com/wsla`.

Jin, H. and Wu, H. (2005). Semantic-enabled specification for web services agreement. *International Journal of Web Services Practices*, 1(1–2):13–20.

Kangar, S. (1972). Law and logic. *Theoria*, 38:105–132.

Kolovski, V., Parsia, B., Katz, Y., and Hendler, J. A. (2005). Representing Web Service Policies in OWL-DL. In Gil, Y., Motta, E., Benjamins, V. R., and Musen, M. A., editors, *The Semantic Web - ISWC 2005, 4th International Semantic Web Conference (ISWC 2005)*, volume 3729 of *Lecture Notes in Computer Science*, pages 461–475. Springer.

Lamparter, S., Ankolekar, A., Grimm, S., and Studer, R. (2007). Preference-based selection of highly configurable web services. In *Proc. of the 16th Int. World Wide Web Conference (WWW'07)*, pages 1013–1022, Banff, Canada.

Lamparter, S., Ankolekar, A., Oberle, D., Studer, R., and Weinhardt, C. (2006). A policy framework for trading configurable goods and services in open electronic markets. In *Proceedings of the 8th Int. Conference on Electronic Commerce (ICEC'06)*, pages 162–173, New Brunswick, Frederiction, Canada.

Li, L. and Horrocks, I. (2003). A software framework for matchmaking based on semantic web technology. In *WWW '03: Proceedings of the twelfth international conference on World Wide Web*, pages 331–339, Budapest, Hungary. ACM Press.

Lindahl, L. (1977). Position and changea study in law and logic. *Synthese Library*, 112.

Ludwig, H., Dan, A., and Kearney, R. (2004). Cermona: an architecture and library for creation and monitoring of ws-agreents. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 65–74, New York, NY, USA. ACM Press.

Ludwig, H., Keller, A., Dan, A., King, R., and Franck, R. (2003). A service level agreement language for dynamic electronic services. *Electronic Commerce Research*, 3(1-2):43–59.

Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A., and Schneider, L. (2002). The WonderWeb library of foundational ontologies. WonderWeb Deliverable D17. Available from `http://wonderweb.semanticweb.org`.

Milosevic, Z. (1995). *Enterprise Aspects of Open Distributed Systems*. PhD thesis, Computer Science Dept. The University of Queensland.

Milosevic, Z. and Governatori, G. (2005). Special issue on contract architectures and languages – guest editors' introduction. *International Journal of Cooperative Information Systems*, 14(2-3):73–76.

Moses, T., Anderson, A., Proctor, S., and Godik, S. (2003). XACML profile for web services. available from `http://www.oasis-open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf`. Oasis Working Draft.

Motik, B., Sattler, U., and Studer, R. (2005). Query answering for OWL-DL with rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60.

Noia, T. D., Sciascio, E. D., Donini, F. M., and Mongiello, M. (2003). A system for principled matchmaking in an electronic marketplace. In *WWW '03: Proceedings of the twelfth international conference on World Wide Web*, pages 321–330, Budapest, Hungary. ACM Press.

OASIS ebXML Joint Committee (2004). Organization for the Advancement of Structured Information Standards, Enabling a global electronic market: ebXML. http://www.ebxml.org.

Oberle, D. (2005). *Semantic Management of Middleware*. PhD thesis, University of Karlsruhe(TH).

Oberle, D., Lamparter, S., Grimm, S., Vrandecic, D., Staab, S., and Gangemi, A. (2006). Towards ontologies for formalizing modularization and communication in large software systems. *Journal of Applied Ontology*, 2(2):163–202.

Oldham, N., Verma, K., Sheth, A., and Hakimpour, F. (2006). Semantic WS-Agreement partner selection. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 697–706, Edinburgh, Scotland. ACM Press.

Paschke, A. (2006). Verification, validation and integrity of distributed and interchanged rule based policies and contracts in the semantic web. In *International Semantic Web and Policy Workshop (SWPW'06) at ISWC'06*, Athens, Georgia, USA.

Paschke, A., Bichler, M., and Dietrich, J. (2005). Contractlog: An approach to rule based monitoring and execution of service level agreements. In *International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML 2005)*, Galway, Ireland.

Patil, A., Oundhakar, S., Sheth, A., and Verma, K. (2004). METEOR-S Web Service Annotation Framework. In *The 13th International World Wide Web Conference Proceedings*, pages 553–563. ACM.

Reinecke, J., Dessler, G., and Schoell, W. (1989). *Introduction to Business - A Contemporary View*. Allyn and Bacon, Boston.

Sergot, M. (2001). A computational theory of normative positions. *ACM Trans. Comput. Logic*, 2(4):581–622.

Skylogiannis, T., Antoniou, G., Bassiliades, N., and Governatori, G. (2005). DR-NEGOTIATE - a system for automated agent negotiation with defeasible logic-based strategies. In *EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, pages 44–49, Washington, DC, USA. IEEE Computer Society.

Sycara, K., Paolucci, M., Ankolekar, A., and Srinivasan, N. (2003). Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics*, 1(1).

Tan, Y.-H. and Thoen, W. (1998). A logical model of directed obligations and permissions to support electronic contracting. *Int. J. Electronic Commerce*, 3(2):87–104.

Uszok, A., Bradshaw, J. M., Johnson, M., Jeffers, R., Tate, A., Dalton, J., and Aitken, S. (2004). Kaos policy management for semantic web services. *IEEE Intelligent Systems*, 19(4):32–41.

W3C (2004). Web ontology language (OWL). http://www.w3.org/2004/OWL/.

W3C (2006). Web Services Policy Framework (WS-Policy). http://www.w3.org/Submission/WS-Policy/.

Wang, G., Wang, C., Chen, A., Wang, H., Fung, C., Uczekaj, S., Chen, Y.-L., Guthmiller, W. G., and Lee, J. (2005). Service level management using qos monitoring, diagnostics, and adaptation for networked enterprise systems. In *Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05)*, volume 0, pages 239–250, Los Alamitos, CA, USA. IEEE Computer Society.