To appear in International Journal of Electronic Commerce,

Accepted Journal Paper, revised version of Nov. 19, 2003

SweetDeal: Representing Agent Contracts with Exceptions using Semantic Web Rules, Ontologies, and Process Descriptions

Benjamin N. Grosof

MIT Sloan School of Management

50 Memorial Drive, Cambridge, MA 02142, USA; +01 617 253 8694

bgrosof@mit.edu ; http://ebusiness.mit.edu/bgrosof

Terrence C. Poon*

Oracle Corporation

500 Oracle Parkway, Redwood Shores, CA 94065, USA; +01 650 506 7000

tpoon@alum.mit.edu; * work done while student at MIT

Abstract:

We address the problem in automated knowledge-based e-contracts of how to represent *exception handling* provisions – which in many (if not most) natural language contracts constitute the majority of the contract's volume. We make three main novel contributions. First, we newly extend the previous SweetDeal e-contracting approach to incorporate ontology knowledge, specifically *ontologies about business processes,* and show how that is useful to represent provisions for exception handling. We give a detailed application scenario of provisions about late delivery in manufacturing supply chain management (SCM). SweetDeal is a rule-based approach to representation of business contracts that enables software agents to create, evaluate, negotiate, and execute contracts with substantial automation and modularity. Its rules are represented in the Situated Courteous Logic Programs knowledge representation encoded in RuleML, the leading approach to Semantic Web rules. The process ontologies are represented in Description Logic encoded in DAML+OIL, the close predecessor of W3C's OWL, the leading approach to Semantic Web ontologies.

Second, we give a new simple technical mechanism for the Semantic Web to integrate such rules "on top of" such ontologies. Rule predicates are defined by *reference* to classes or properties in ontology knowledge bases. This mechanism provides the basic technical approach to incorporate ontology knowledge into a SweetDeal e-contract.

Third, we newly show how to represent, in DAML+OIL/OWL, process ontology knowledge drawn in particular from the MIT Process Handbook. The Handbook is a large,

previously-existing repository used by practical industrial process designers. We formalize a fraction of the Handbook's knowledge and use it for our e-contracting application scenario.

More generally, our approach provides a foundation for representing and automating deals about services – in particular, about Semantic Web Services, so as to help search, select, and compose them. Our system is the first to combine emerging Semantic Web standards for knowledge representation of rules (RuleML) and of ontologies (DAML+OIL/OWL) with each other, and moreover for a practical e-business application domain. A prototype is running.[1][2]

Keywords: Electronic contracts, electronic commerce, XML, Semantic Web, Web Services, Semantic Web Services, knowledge representation, intelligent software agents,

[1] We intend to make the prototype publicly available.

[2] Earlier versions of this paper appeared as [28] [29].

rules, logic programs, ontologies, business process automation, process descriptions, process knowledge, RuleML, RDF, Description Logic, DAML+OIL, OWL, knowledge-based, declarative.

1. INTRODUCTION AND OVERVIEW

A key challenge in e-commerce is to specify the terms of the deal between buyers and sellers, e.g., pricing and description of goods/services. In previous work [27] [41], we have developed an approach that automates (parts or all of) such business contracts by representing and communicating them as modular sets of declarative logic-program rules. This approach enables software agents in an electronic marketplace to create, compose, modify, discover, evaluate, negotiate, execute, and monitor contracts with substantial automation and modularity. It enables a high degree of reuse of the contract description (i.e., of its declarative knowledge) for these multiple purposes in the overall process of contracting. That approach is now called *SweetDeal*. SweetDeal is fairly unique in its approach and capabilities; for related work on it, see [27] [41].

SweetDeal's rules are represented in the Situated Courteous Logic Programs (SCLP) knowledge representation encoded in RuleML [27] [7] [24] [22], the leading approach to Semantic Web rules. The SweetDeal prototype system is built upon the SweetRules toolkit [21] [24] [22] for rules inferencing and interoperability in SCLP RuleML.

In this paper, we address the problem in automated knowledge-based e-contracts of how to represent *exception handling* provisions.

Terminology: An *exception* is something that does not go as is normal or expected/usual. This is a concept that applies generally in specification of business processes. In the context of contracts, specifically, one important category of exception is: violation of a (contract) commitment. An *exception handler* is a business (sub-)process which specifies what behavior should be performed in case of an exception. In the extension of our SweetDeal approach, an exception handler may be specified as part of a contract. For example, a contract can first identify possible exceptions like late delivery or non-payment. Next, it can specify handlers to find or fix these exceptions, such as contingency payments, escrow services, and notifications.

The majority of the volume of many (if not most) existing contracts and business processes is devoted to specifying provisions for exception handling. A number of idioms and proverbs in the English language refer to this necessity, e.g.: "Murphy's Law", "Stuff Happens", and even "The course of true love never did run smooth.". Large business process automation software applications written by programmers tend to have lots of details about exception handling. So also do lengthy natural language contract documents written by lawyers.

More generally, exceptions and exception handlers are interesting because they are an important kind of relatively complex behavior -- i.e., contingent behavioral provisions about business processes -- that must be represented in contracting. They are particularly vital in contracts about services.

In this paper, we describe three main novel contributions.

1. We newly extend the previous SweetDeal e-contracting approach to incorporate ontology knowledge, specifically ontologies *about business processes*, and show how that is useful to represent provisions for exception handling. We give a detailed application scenario of provisions about late delivery in manufacturing supply chain management (SCM). The process ontologies are represented in Description Logic (DL) encoded in DAML+OIL [10]. DAML+OIL is the close predecessor of W3C's OWL [40], the leading approach to Semantic Web ontologies. We used DAML+OIL because when we began this work, OWL did not yet exist.

2. We give a new simple technical mechanism for the Semantic Web to integrate SCLP RuleML rules "on top of" Description Logic DAML+OIL/OWL ontologies. Predicates appearing in rules can be defined by *reference* to classes or properties specified in ontology knowledge bases. This mechanism provides the basic technical approach to incorporate ontology knowledge into a SweetDeal e-contract. Our system is the first to combine emerging Semantic Web standards for knowledge representation of rules (RuleML) and of ontologies

(DAML+OIL/OWL) with each other, and moreover for a practical e-business application domain.

3. We newly show how to represent, in DAML+OIL/OWL, process ontology knowledge (e.g., about exceptions and handlers) drawn specifically from the MIT Process Handbook (PH) [37]. The PH is a previously-existing repository of business process knowledge that is unique in its large content and frequent use by industry business process designers as well as researchers (soon available in open source version). The PH includes (as one part of its content) taxonomic/hierarchical aspects and knowledge about exceptions/exception-handling. We newly formalize a fraction of this part of the PH's knowledge and use it for our e-contracting application scenario. This is the first time that the PH has been automated using XML, or Semantic Web knowledge representation (KR), or DL logical knowledge representation, or LP logical knowledge representation. Our formalization of the PH knowledge newly enables practical deep inferencing with that knowledge using Semantic Web tools. Previously PH content was only relatively shallowly automated for inferencing.

More generally, our approach provides a foundation for representing and automating *deals about services* – including about electronic services, e.g., Web Services – so as to help create, compose, modify, discover, evaluate, negotiate, execute, and monitor them. It points the way to how and why to combine Semantic Web techniques [43] with Web Services techniques [48] to create *Semantic Web Services (SWS)* [44] [12] [14] [23]. The

Language requirements and architecture (version of Nov. 2003) of the Semantic Web Services Initiative [44] incorporates rules and contracting aspects that are largely based on our work reported here.

A prototype of our system is running. We intend to make the prototype publicly available.

2. OUTLINE OF THE REST OF THE PAPER

In section 3, we review the overall SweetDeal approach and its advantages. In section 4, we review RuleML, OWL, and Situated Courteous Logic Programs. In section 5, we newly give an approach to using a DAML+OIL ontology to define the predicates of RuleML rules. In section 6, we review the MIT Process Handbook (PH) [37] [36], and Klein *et al*'s extension of it to treat exception conditions in contracts [31]. In section 7, we newly show how to represent the Process Handbook's process ontology (including about exceptions) in DAML+OIL/OWL, giving some examples. In section 8, we describe our new development of an additional ontology specifically about contracts, again giving examples in DAML+OIL/OWL. This contract ontology extends and complements the PH process ontology. In section 9, we newly show how to use the DAML+OIL/OWL process ontology, including about contracts and exceptions, as the predicates etc. of RuleML rules, where a ruleset represents part or all of a (draft or final) contract with exceptions and exception handlers. We illustrate by giving an a detailed application scenario (long series of examples) of such a contract ruleset whose rule-based contingency provisions include detecting and penalizing late delivery exceptions, thus providing means to deter or adjudicate a late delivery. In section 10, we briefly describe the overall SweetDeal prototype system. In section 11, we give additional discussion including directions for current and future work.

3. REVIEW OF OVERALL SWEETDEAL APPROACH

In the overall SweetDeal approach to e-contracting:

- Contracts are represented using interoperable rules in XML. Declarative logic programs (LP) are the fundamental knowledge representation (KR) in which these rules are expressed. This KR actually spawned RuleML, via its predecessor BRML in IBM CommonRules. Underlying the choice of this KR is a requirements analysis for rule KR and syntax.

- Contracts may be: partial or complete, proposed or final.

- Rules represent part or all of an overall contract. Named importable rule modules represent portions of a contract, then are assembled by merging into a contract rulebase (that represents part or all of the overall contract).

- Modification of a contract rulebase can be performed modularly during negotiation or completion, e.g.: to add new contract provisions in a counterproposal; or to add price, quantity, and buyer after an auction is completed. This modularity of modification is enabled by using prioritized conflict handling in the rules representation. The Courteous expressive extension of LP, which is tractably compileable into Ordinary LP (i.e., LP with negation as failure), is employed to express such prioritized conflict handling.

- Procedural attachments in the rules are used to perform external actions or queries, as well as built-ins such as arithmetic operations or comparisons. The Situated expressive extension of LP is employed to express such procedural attachments; it provides a declarative abstraction of them. Thus, for example, a contract rulebase when executed may invoke surrounding external business processes.

- Thus, overall, the approach uses the Situated Courteous LP (SCLP) KR in RuleML.

- In addition, beyond the contract rulebase which is shared between contracting parties/agents, each agent also needs typically to do its own "*solo*" decision making / support. "Solo" here means using some information which is not shared with the other parties to the contract. Part or all of this solo decision making/support might be performed by an agent in a rule-based way, using solo rules in combination with the contract's rules. The SweetDeal work to date, however, focuses primarily on the shared contract rulebase and its uses, rather than upon the solo aspect.

- Previous work on the SweetDeal approach includes [27] which first gave the basic general approach, including the Courteous LP rules KR encoded in XML and implemented in the IBM CommonRules rules toolkit. Previous SweetDeal work additionally includes [41] which gave an approach to auctions negotiation and configuration, implemented in the ContractBot system which extended U. Michigan's AuctionBot auction server, a leading university (electronic) auction server. Previous SweetDeal prototypes with application pilots also included the EECOMS system (built in 1998-2000) for manufacturing supply chain collaboration, which performed negotiation [27]. EECOMS was a $ 29 Million (US) NIST ATP[6] project by the CIIMPLEX consortium including IBM (lead), Boeing, TRW, Baan, Vitria, and several universities and smaller software companies.

Next, we summarize the **advantages** of the SweetDeal approach. What are the **competing approaches** to representing contracts? One approach is no automation beyond text

---

[6] NIST = National Institute of Standards and Technology, a part of the US government's Dept. of Commerce. ATP = Advanced Technology Program, the aim of which is to speed research into commercialization within 3-5 years rather than the typical 10+ years.

document processing – this is the way most legal contracts are handled today. Another approach is to represent contracts in software using general code and data management techniques. A third approach is to use an XML interchange language for general-purpose e-commerce transactions and messaging; the leading (and representative) example of this approach is ebXML [15]. Relative to these three approaches, there are several advantages of SweetDeal's rule-based approach, based on Situated Courteous LP RuleML, to e-contracting. SweetDeal is fairly unique in being rule-based, and quite unique in using SCLP and in using XML. There is thus no closely related work (in the sense of other competing approaches) to it. The advantages of theSweetDeal approach include:

- Rules (vs. general code) provide a relatively high level of conceptual abstraction. The makes it easier for non-programmers to understand or to specify contract content.
- Rules are especially good for specifying contingent provisions of contracts
- One may automatically reason about the contract/proposal by performing inferencing in the rule KR, for which computational tractability guarantees are available [27]. E.g., to examine hypothetical ("what-if") scenarios, to perform testing, or to evaluate a proposed contract's value (in utility or money). This capability for reasoning is enabled by the declarativeness and expressiveness, together with the (average or worst-case) tractability, of the rule KR. (More about the expressiveness and complexity of the SCLP KR is discussed in section 4.)

- One may communicate automatically with deep shared semantics, cf. the "Semantic Web" vision: via RuleML which is interoperable with the same sanctioned inferences, between heterogeneous commercially important kinds of rule/database systems or rule-based applications ("agents") that use such systems.
- One may actually automatically execute the contract provisions by performing rule inferencing, including to invoke e-business actions via Situated (LP's) procedural attachments.
- One may modify modularly, and thus relatively easily, the contract provisions: using prioritized defaults and rule modules via Courteous (LP's capability for) overriding.
- Overall, there is a relatively high degree of automated reusability: arising from the KR's declarativeness, modularity, and interoperability.

Examples of contract provisions well-represented by rules in automated deal making include:

- Product descriptions, e.g., product catalogs in which properties are specified, often conditionally upon other properties. E.g., all women's sweaters are available in size extra-small.
- Pricing, e.g., dependent upon delivery-date, quantity, group memberships, or umbrella contract provisions.
- Terms & conditions (in the sense that phrase is used in contracts): e.g., refund or cancellation timelines or deposits, lateness or low-quality penalties, ordering lead time, shipping, credit-worthiness, business-partner qualification, service provisions.

- Trust: e.g., credit-worthiness, authorization, or required signatures.

These provisions appear not only in descriptions of seller offerings of specific products and services, but also often in buyer requirements (e.g., RFQ, RFP[7]) or seller capabilities (e.g., for sourcing or qualification during procurement/SCM).

4. REVIEW OF RULEML, OWL, AND SITUATED COURTEOUS LOGIC PROGRAMS

RuleML: Declarative logic programs in RuleML is the leading approach to Semantic Web rules, and is an emerging standard.[8] "RuleML" can be read as standing for either "Rule Markup Language" or "Rule Modeling Language". RuleML provides a Web-oriented abstract syntax and declarative knowledge representation semantics for rules. Its concrete syntaxes include an XML syntax and an RDF syntax. (RDF in turn has an XML-syntax encoding.) The RuleML standards effort is being pursued in collaboration with (1) the DARPA Agent Markup Language Program (DAML) [11]; and (2) the Joint US/EU ad hoc Agent Markup Language Committee (known simply as the "Joint Committee") [30] which designed DAML+OIL. In addition, RuleML is in informal cooperation with: (3) the W3C's Semantic Web Activity [43], which includes rules in its charter

---

[7] RFQ = Request For Quotation, i.e., a price quote with description. RFP = Request For Proposal.

[8] We (first author) co-founded and co-lead RuleML.

along with ontologies; (4) the Semantic Web Services Initiative (SWSI) [44]; and (5) the Oasis e-business standards body [39].

OWL: OWL [40] is the leading approach to Semantic Web ontologies, and is a nearly-completed standard from the World Wide Web Consortium (W3C). It is based very closely on DAML+OIL; their fundamental knowledge representation is Description Logic (DL). DL is an expressive fragment (i.e., subset) of First Order Logic. OWL, like DAML+OIL, encodes this syntactically in Resource Description Framework (RDF) [32]. RDF is a somewhat cleaner, simpler, and more expressive language for labeled directed graphs than basic XML, and itself in turn has an XML encoding (which is commonly used in practice). The approach we take in this paper to using DAML+OIL carries over with little or no modification to using OWL instead.

SCLP KR: The SCLP KR is an expressively powerful kind of declarative Logic Programs. The Courteous expressive extension of Logic Programs enables prioritized conflict handling and thereby facilitates modularity in specification, modification, merging, and updating. The Situated expressive extension of Logic Programs enables procedural attachments for "sensing" (testing rule antecedents) and "effecting" (performing actions triggered by conclusions).

Merging and modification is important specifically for automated ("agent") contracts, because contracts are often assembled from reusable provisions, from multiple organizational sources, and then tweaked. Updating is important because a contract is often treated as a template to be filled in. For example, before an on-line auction is held a contract template is provided for the good/service being auctioned. Then when the auction closes, the template is filled in with the winning price and the winner's name, address, and payment method. [41] shows how to use SCLP to represent contracts in this dynamically updated manner, for a real auction server – U. Michigan's AuctionBot – and the semi-realistic domain of a Trading Agent Competition about travel packages. More generally, the design of SCLP as a knowledge representation (KR) grew out of a detailed requirements analysis [27] for rules in automated contracts and business policies.

The SCLP KR has fairly attractive worst-case computational complexity [27][24][22]. SCLP inferencing is tractable (i.e., polynomial time) under the Datalog restriction (i.e., no logical functions) and two additional restrictions/assumptions. The first is that the number of logical variables per rule is bounded by a constant *v*. In practice, this is a quite reasonable assumption, with *v* often being roughly 5 or 10. The second is that the computational cost of a call to an attached procedure is worst-case polynomial time, again a quite reasonable assumption in practice. The complexity, and scaleability, of Datalog SCLP KR is not much worse than that of SQL databases (i.e., Datalog Horn LP KR) which in practice scale up very well. Our examples of SCLP in this paper (in section 9) abide by the Datalog restriction, for example. Going beyond the Datalog restriction, the complexity of SCLP KR is only a bit worse than that of Prolog – which in practice scales up quite well. By contrast, First Order Logic (FOL) and Description Logic under comparable restrictions (Datalog, and bounded number of variables per clause/axiom) are intractable. (SC)LP has one main expressive limitation, as compared to First Order Logic (FOL) or to Description Logic (DL): in LP one cannot conclude a disjunction (nor, directly, an existential). FOL and DL have two main expressive limitations, as compared to SCLP: they cannot express nonmonotonicity, default reasoning or conflict handling; and they cannot express procedural attachments, especially for effecting/actions.

5. INTEGRATING DAML+OIL/OWL ONTOLOGIES INTO RULEML RULES

In this section, we describe our new technical representational approach for the integration of the DAML+OIL ontologies into the RuleML rules, in which the RuleML rules are specified "on top of" the DAML+OIL ontology. **This same approach also applies to OWL** ontologies, if OWL is employed instead of DAML+OIL; no change is required. In section 9, we give examples of RuleML contract rules that make use of DAML+OIL process ontologies.

The high-level goal of rules "on top of" ontologies has been an important topic of discussion in the Semantic Web community, including in architecting standards, since at least 2000. Figure 1 shows the current (April 2002) version of the W3C's Semantic Web "stack" of standardization steps [2], annotated to indicate where RuleML and DAML+OIL/OWL fit in. However, before our work reported here, this Semantic Web goal of rules "on top of" ontologies had not been operationalized with a specific technical approach.

5.1 Concept and Syntactic Mechanism

The essence of our integration approach is that RuleML rules can ***reference*** OWL/DAML+OIL ontologies. In this subsection, we describe the conceptual approach and syntactic mechanism for doing this. We will give examples in section 9.

- A RuleML rulebase may import one or more OWL/DAML+OIL ontologies. I.e., the RuleML rulebase may optionally syntactically specify a list of imported ontological knowledge bases in OWL/DAML+OIL. E.g., in our examples in section 9, a contract rulebase imports two ontologies pr.daml and sd.daml.

- Syntactically, the names of predicates appearing in the RuleML rules may be URI's that reference (i.e., denote) classes and properties in a DAML+OIL/OWL ontology. This is accomplished quite simply by having the rule predicate use as its name the URI that is the name defined in the ontology for the class or property being referenced. RuleML permits a predicate (or an individual) to be a URI; we make heavy use of this capability since the names of DAML+OIL/OWL classes and properties (and individuals) are URI's. E.g., in our examples in section 9, a rule uses as a predicate the property sd:exceptionOccurred that is defined in the ontology sd.daml, and as another predicate the property pr:isHandledBy that is defined in the ontology pr.daml.

- In a similar spirit, the names of individuals appearing in RuleML rules may be the same URI's for individuals as used in a DAML+OIL/OWL ontology.[10]

- A DAML+OIL/OWL class is treated in RuleML as a unary (i.e., arity 1) predicate. A DAML+OIL/OWL property is treated in RuleML as a binary (i.e., arity 2) predicate.

- Assertions about instances in a class are treated as rule atoms in which the class predicate appears. Assertions about property links between class instances are treated as rule atoms in which the property predicate appears.

- Conceptually, the referenced DAML+OIL/OWL ontological knowledge base (formed by the union of the referenced/imported ontologies, if there are several such) is viewed as a background theory for the rulebase. In the next subsection, we give more details about the semantics.

5.2 Semantics

A natural question arises: how to define formally the semantics of such integration, i.e., of the *hybrid* KR formed by combining LP rules on top of DL ontologies (or, similarly, by combining Horn FOL rules on top of DL ontologies). To address this question, and

---

[10] Our examples in this paper's main application scenario (section 9) do not actually need to make use of this capability, hence we omit details. The examples focus rather on incorporating what in Description Logic terminology is called "T-box" ontological knowledge about the classes and properties.

motivated in large part by the work in this paper, a separate line of research has been developed that addresses this question: *Description Logic Programs (DLP)* [26] and the associated "DLP-fusion" approach to the semantics of combining DL knowledge with LP knowledge. Description Logic Programs is a KR that captures a large subset of the expressive intersection of Logic Programs and Description Logic. Defining/studying the expressive *intersection* of LP and DL is then a key to defining, and enabling automated inferencing in, a large subset of the expressive *union* of LP and DL. DLP can be viewed as an "ontology sublanguage" within the full LP KR. If the DL ontological knowledge base is in the DLP subset of DL, it can thus be merged (completely) into the LP KR – thereby avoiding any problems of incompleteness or inconsistency from the hybridizing.

What makes it challenging, in general, to define the semantics of hybridizing the LP and DL KR's is the potential for incompleteness or inconsistency. Defining the union of LP and DL is difficult for the fully general case of full LP (especially with nonmonotonicity and/or procedural attachments) combined with full DL; current research is exploring how to increase the expressiveness covered.

Figure 2 illustrates the relevant KR expressive classes and their overlaps, in the form of a Venn Diagram. Description Logic is (equivalent to) a subset of First Order Logic (FOL). The Logic Programs KR in its full generality of expressiveness includes two major features that are not expressible in FOL: negation-as-failure, which is logically nonmonotonic and underpins prioritized conflict handling as well; and procedural attachments. However, Horn Logic Programs (the Horn subset of LP) is (equivalent to) a subset of First Order Logic. Description Logic Programs are contained in the expressive intersection of Description Logic and Horn Logic Programs. DLP is thus a subset of FOL.

A somewhat similar hybrid KR to DLP-fusion is addressed in an approach by Antoniou [1], which was developed independently at the same time as this work and [26]. The approach in [1] is, however, quite limiting for practical purposes since it does not permit predicates defined in the DL ontology to appear in the heads of the LP rules. As we will see in the next section, it is quite natural and desirable to have such "ontological" predicates appear in rule heads. [26] gives more discussion about how the approach in [1] compares to the DLP approach.

5.3 Summary of Novel Contribution

The novel contribution in this paper of our approach to rules on top of ontologies is primarily the concept and the syntactic mechanism, along with a substantial business application scenario (in section 9) as use case, that together with the DLP semantics [26] show how and why to do this combination. The work reported here provided a prime motivation for developing the theory of DLP.

To our knowledge, ours is the first published description and example of such integration (by reference etc.) of DAML+OIL/OWL into RuleML. And this is one of the first two published descriptions and examples of combination of DAML+OIL (or OWL) with any *non-monotonic* rule KR — the other being [1] which was done independently.

6. REVIEW OF MIT PROCESS HANDBOOK (PH)

In this section, we review the MIT Process Handbook (PH) [37] [36], and Klein *et al*'s extension of it to treat exception conditions in contracts [31]. Our example scenario's process ontologies are drawn partly from the PH.[11]

6.1 Overview; Size and Significance

The PH is a previously-existing knowledge repository of business process descriptions. It is primarily textual and oriented to human-readability although with some useful automation for knowledge management using taxonomic structure. The PH has been under development at the MIT Center for Coordination Science (CCS) for over ten years, including the contributions of a diverse and highly distributed group of over 40 scientists, students and industrial sponsors [36] [35]. The goal of the PH project is to develop a repository and associated conceptual tools that help users effectively retrieve and exploit the process knowledge relevant to their current challenges. The PH is large: it contains, to date, descriptions of over 5000 processes, 12000 entities, and 38000 properties/values. The PH has been used by a number of practical industrial process designers from a variety of companies, partly via a dedicated commercial spinoff (Phios Corp.). It also has been used for a number of research projects, as well as for teaching.

The Handbook describes and classifies major business processes using the organizational concepts of *decomposition*, *dependencies*, and *specialization*. The Handbook models each

[11] The version of the PH we used was that of approximately March 2002.

process as a collection of activities that can be decomposed into sub-activities, which may themselves be processes. In turn, coordination is modeled as the management of dependencies that represent flows of control, data, or material between activities. Each dependency is managed by a coordination mechanism, which is the process that controls its resource flow.

The PH's process ontologies have been shown, at a research level, to be useful in a variety of domains such as business process reengineering [3] [36], e-contracting (this work), business process automation [4], software design [13], etc. This provides evidence that PH process ontologies could be used in future to help supply the semantic service descriptions for the key contracting and Semantic Web Services tasks discussed in section 1. The PH has been influential as well in the arena of industry standards for process representation/modeling: it spawned Process Interchange Format (PIF) [33] which in turn spawned Process Specification Language (PSL) [42], a draft NIST/ISO standard.

Among automated repositories of business process knowledge, the PH is unique (to our knowledge) in having a large amount of content and having been frequently used by a range of researchers and industry business process designers. An **open source** version of the PH is under development.[12]

---

[12] planned for release probably in 2004

6.2 Object-Oriented Taxonomic Aspects

The PH uses an Object-Oriented (OO) style of taxonomic hierarchy, as a tool to organize part of its content for retrieval and browsing. Processes (and many other important entities in the Handbook) are arranged into a generalization-specialization taxonomy, with generic processes at the top and increasingly specialized processes underneath. Each specialization automatically inherits the properties of its parents, except where it explicitly adds or changes a property. This is similar to taxonomic class hierarchies having default inheritance[13], such as in many Object-Oriented (OO) programming languages, knowledge representations (KR's), and information modeling systems. Note that the taxonomy is not a tree, as an entity may have multiple parents. In general, there thus is multiple inheritance. For example, BuyAsALargeBusiness is a subclass of both Buy and ManageEntity. Figure 3 shows a part of the taxonomy with some of the specializations for the "Sell" process. Note the first generation of children of "Sell" are questions; these are classes used as intermediate categories, analogous to virtual classes (or pure interfaces) in OO programming languages. Since there is multiple inheritance, it is easy to provide several such "cross-cutting" dimensions of categories along which to organize the hierarchy.

6.3 Previously Not Mapped to Semantic Web or XML

Previous to our work in SweetDeal, however, the PH's content had never been automated in XML (i.e., not Webized), nor had that content ever been represented in Description

---

[13] a.k.a. "inheritance with exceptions", a.k.a. "non-monotonic inheritance"

Logic KR, Logic Programs KR, or using Semantic Web techniques. The PH's content was previously only relatively shallowly automated for inferencing. A part of the PH's kind of knowledge had, however, been previously encoded in Process Interchange Format (PIF) [33], which maps straightforwardly to Knowledge Interchange Format (in classical logic) [9]; PIF export is a capability of the PH's software.

6.4 Exception Conditions

The terms of any contract establish a set of commitments between the parties involved for the execution of that contract. When a contract is executed, these commitments are sometimes violated. Often contracts, or the laws or automation upon which they rely, specify how such violation situations should be handled.

Building upon the PH, Klein *et al* [31] consider these violations to be coordination failures – called "exceptions" – and introduces into the PH the concept of exception handlers, which are processes that manage particular exceptions. We in turn build upon Klein *et al*'s approach. When an exception occurs during contract execution, an exception handler associated with that exception may be invoked. Figure 4 shows some (kinds of) exceptions that are currently in the PH, organized into a generalization-specialization hierarchy.

[14] a.k.a. "inheritance with exceptions", a.k.a. "non-monotonic inheritance"

For example, in a given contract (agreement), company A agrees to pay $50 per unit for 100 units of company B's product, and B agrees to deliver within 15 days (commitments). However, due to unforeseen circumstances, when the contract is actually performed, B only manages to deliver in 20 days (exception). As a result, B pays $1000 to A as compensation for the delay (exception handler).

There are four classes of exception handlers in [31]. For an exception that has not occurred yet, one can use:

- Exception <u>anticipation</u> processes, which identify situations where the exception is likely to occur.
- Exception <u>avoidance</u> processes, which decrease or eliminate the likelihood of the exception.

For an exception that has already occurred, one can use:

- Exception <u>detection</u> processes, which detect when the exception has actually occurred.
- Exception <u>resolution</u> processes, which resolve the exception once it has occurred.

[31] extends the PH with an exception taxonomy. Every process may be associated via *hasException* links to its potential exceptions (zero or more), which are the characteristic ways in which its commitments may be violated. *hasException* should be understood as "has potential exception". Similar to the process taxonomy, exceptions are arranged in a specialization hierarchy, with generic exceptions on top and more specialized exceptions

underneath. In turn, each exception is associated (via an *isHandledBy* link) to the processes (*exception handlers*) that can be used to deal with that exception. Since handlers are processes, they may have their own characteristic exceptions. Figure 5 shows some exception handlers in the Handbook, organized into a generalization-specialization hierarchy.

Following the general style of (multiple) inheritance in the PH, the exceptions associated with a process are inherited by the specializations of that process. Similarly, the handlers for an exception are inherited by the specializations of that exception.

7. REPRESENTING THE PH PROCESS ONTOLOGY IN DAML+OIL/OWL

7.1. Approach and Overview

In this section, we newly show how to represent the Process Handbook's process ontology (including about exceptions) in DAML+OIL, giving some examples. **This same approach also applies to OWL** ontologies, if OWL is employed instead of DAML+OIL; no change is required. Our approach has two basic aspects. The first is to represent the PH's ontological knowledge in terms of the fundamental Description Logic (DL) KR that underlies DAML+OIL (and OWL), i.e., to define classes, properties, and statements about these classes and properties that specify subclassof relationships, as well as domain, range, and other property-restriction information. The second is to encode this syntactically in DAML+OIL's syntax. The basic approach to represent this knowledge instead in OWL would be quite similar, it requires just a change in the syntactic encoding, since the KR features we use from DL are the same in OWL as in DAML+OIL.

The full PH's ontology is a quite large one, containing thousands of classes and properties, plus a very extensive body of associated textual descriptions. Our main concern in this paper is to show *how* to *represent* this ontology in DAML+OIL, and how/why to make *use* of that ontology for contracts with exception handling (and thus more generally for Semantic Web Services).

We have developed a PH process ontology in DAML+OIL, which we have given a URI[15] of http://ccs.mit.edu/sd/pr.daml, where "pr" stands for "process". So far we have represented in DAML+OIL only a few percent of the ontological knowledge in the full PH that could be represented with our approach. This pr.daml was created manually. Doing this manual work of representation is somewhat labor-intensive, but in terms of knowledge/skill requires only a moderate familiarity with the PH, along with basic familiarity with DAML+OIL. In current work, however, we (first author and additional collaborators) are developing a method to *automatically* create the DAML+OIL representation by automatically exporting ontological knowledge from the PH. In this section, we give a relatively small subset (which we call pr-subset.daml) of the PH process ontology we have so far represented.

[15] for purposes of this paper

7.2. DAML+OIL Ontological Axioms

We begin with some DAML+OIL headers that declare XML namespaces and that what follows is a DAML+OIL ontology (encoded, as usual, in RDF):

```
<?xml version="1.0" ?>

<rdf:RDF

xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

xmlns:daml="http://www.daml.org/2001/03/daml+oil#"

xmlns     ="pr:" >

<daml:Ontology rdf:about="">

  <daml:imports
rdf:resource="http://www.daml.org/2001/03/daml+oil"/>

</daml:Ontology>
```

Next we define some main concepts in the MIT Process Handbook as top-level classes.

For example:

```
<daml:Class rdf:ID="Process">

  <rdfs:comment>A process</rdfs:comment>
```

```
</daml:Class>

<daml:Class rdf:ID="Exception">

  <rdfs:comment>A violation of an inter-agent
commitment</rdfs:comment>

</daml:Class>

<daml:Class rdf:ID="ExceptionHandler">

  <rdfs:subClassOf rdf:resource="#Process"/>

  <rdfs:comment>A process that helps to manage a particular
exception</rdfs:comment>

</daml:Class>
```

Then we define the relations between concepts as object properties:

```
<daml:ObjectProperty rdf:ID="hasException">

  <rdfs:comment>Has a potential exception</rdfs:comment>

  <rdfs:domain rdf:resource="#Process" />

  <rdfs:range rdf:resource="#Exception" />

</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="isHandledBy">
```

```
  <rdfs:comment>Can potentially be handled, in some way or
aspect, by</rdfs:comment>

  <rdfs:domain rdf:resource="#Exception" />

  <rdfs:range rdf:resource="#ExceptionHandler" />

</daml:ObjectProperty>
```

The Handbook takes the approach – which we endorse – that it is typically desirable to treat a process repository as potentially extensible, i.e., open. It is often unrealistic to expect a repository to have an exhaustive listing of all handlers for a given exception.

Specializations are expressed as subclasses[16]:

```
<daml:Class rdf:ID="SystemCommitmentViolation">

  <rdfs:subClassOf rdf:resource="#Exception"/>

  <rdfs:comment> Violation of a commitment made by the system
operator to create an environment well-suited to the task at
hand. </rdfs:comment>

</daml:Class>
```

---

[16] In Figure 4 (in Section 6), SystemCommitmentViolation and AgentCommitmentViolation are shown as “Systemic” and “Agent”, respectively.

```
<daml:Class rdf:ID="AgentCommitmentViolation">

  <rdfs:subClassOf rdf:resource="#Exception"/>

  <rdfs:comment> Violation of a commitment that an agents makes
to other agents.

  </rdfs:comment>

</daml:Class>
```

The Process Handbook expects each specialization to inherit the properties of its parent. The DAML+OIL semantics provide this automatically since it entails monotonic (strict) inheritance of such properties.

7.3. ADL Concise Syntax

The syntax of DAML+OIL, while it is reasonably human-readable, is fairly verbose, since it is designed to facilitate automated processing as well (in this regard, it is similar to most XML or RDF syntaxes). For the sake of conciseness and easier human readability, as an alternative for exposition, we thus next define a simple ASCII syntax for a subset of DL that is sufficient for our purposes. We call this "*ADL*" syntax. Note that ADL is similarly a syntax for OWL too. ADL is defined as follows.

";" indicates the end of a (logical) statement. "/* ... */" encloses a comment (rdfs:comment) that is associated with the immediately preceding statement. Below, we let C, C1, C2, D, and R stand for classes, and P stand for a property.

```
C : ;
```

means simply that C is a class.

```
P :: ;
```

means simply that P is a property.

```
C1 : C2;
```

means C1 is a class that is a subclass of (class) C2.

```
P :: D -~ R;
```

means P is a property with domain (class) D and range (class) R.

```
(exists P . C)
```

means the DL restriction class (exists P . C), i.e., there exists a value of property P that is in class C.

```
(forall P . C)
```

means the DL restriction class (forall P . C), i.e., every value of property P is in class C.

```
(card-1 . P)
```

means the DL restriction that P has cardinality 1, i.e., it has exactly one value.

```
(mincard-1 . P)
```

means that DL restriction that P has minimum cardinality 1, i.e., it has at least one value.

7.4. ADL Version of Ontological Axioms

Next, we give the ADL version of pr-subset.daml above:

```
Process  ;

  /* A process */

Exception : ;

  /* A violation of an inter-agent commitment */

ExceptionHandler : Process;

  /* A process that helps to manage a particular exception */

hasException :: Process -~ Exception;

  /* Has a potential exception */

isHandledBy :: Exception -~ ExceptionHandler;

  /* Can potentially be handled, in some way or aspect, by */

SystemCommitmentViolation : Exception;

  /* Violation of a commitment made by the system operator to

       create an environment well-suited to the task at hand. */
```

```
AgentCommitmentViolation : Exception;

  /* Violation of a commitment that an agent makes to other
agents. */
```

In order to give a sense of kind and size of the knowledge in the current pr.daml, using a moderate amount of space, in **Appendix A** we give the ADL version of the **full pr.daml ontology**, omitting most comments.

8. CONTRACT ONTOLOGY

In this section, we describe our development of an additional process ontology specifically about contracting concepts and relations, again giving examples in DAML+OIL. This *contract ontology* extends and complements the PH process ontology. We give it the URI http://ccs.mit.edu/sd/sd.daml, where "sd" stands for "SweetDeal". Like pr.daml, sd.daml was created manually. In this section, we give a relatively small subset (which we call sd-subset.daml) of the contract ontology we have so far represented.

The ontology sd.daml again begins with some DAML+OIL header statements. These are similar to those at the beginning of pr.daml (given in the previous section), with one addition: sd.daml also imports the PH process ontology pr.daml.

```
<daml:Ontology rdf:about="">

  <daml:imports
rdf:resource="http://www.daml.org/2001/03/daml+oil"/>

  <daml:imports rdf:resource="http://ccs.mit.edu/sd/pr.daml"/>
```

```
</daml:Ontology>
```

We view a contract as a specification for one or more processes. Accordingly, we define the *Contract* class and a *specFor* relation that associates a contract to its process(es):

```
<daml:Class rdf:ID="Contract">
  <rdfs:subClassOf>
    <daml:Restriction daml:minCardinality="1">
      <daml:onProperty rdf:resource="#specFor"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
<daml:ObjectProperty rdf:ID="specFor">
  <rdfs:domain rdf:resource="#Contract" />
  <rdfs:range rdf:resource="pr:Process" />
</daml:ObjectProperty>
```

A contract represents the “terms and conditions” that the parties have agreed upon (typically) *before* performing the contract. E.g., they have come to agreement during a negotiation before their contract commitments actually come due. We define a separate concept, *ContractResult*, to represent the state of how the contract was actually carried out.

For example, *ContractResult* could describe the actual shipping date, the quality of the actually received goods, the amount of payment actually received, etc.

```
<daml:Class rdf:ID="ContractResult"/>

<daml:ObjectProperty rdf:ID="result">

  <rdfs:domain rdf:resource="#Contract" />

  <rdfs:range rdf:resource="#ContractResult" />

</daml:ObjectProperty>
```

The process ontology provides the *hasException* relation to indicate that a process *could* have a particular exception. How do we indicate that an exception has occurred during contract execution? We define the *exceptionOccurred* relation on *ContractResult* to denote that the exception happened as the contract was being carried out:

```
<daml:ObjectProperty rdf:ID="exceptionOccurred">

  <daml:domain rdf:resource="pr:ContractResult"/>

  <daml:range rdf:resource="pr:Exception"/>

</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="avoidsException">

  <daml:domain rdf:resource="pr:AvoidException"/>
```

```
  <daml:range
rdf:resource="http://www.daml.org/2001/03/daml+oil#Class"/>

</daml:ObjectProperty>

 <daml:ObjectProperty rdf:ID="resolvesException">

  <daml:domain rdf:resource="pr:ResolveException"/>

  <daml:range
rdf:resource="http://www.daml.org/2001/03/daml+oil#Class"/>

</daml:ObjectProperty>
```

ADL version of contract ontology axioms: Next, we give the ADL version of sd-subset.daml above:

```
sd:Contract : (mincard-1 . sd:specFor);

sd:specFor :: sd:Contract -~ pr:Process;

sd:ContractResult : ;

sd:result :: sd:Contract -~ sd:ContractResult;

sd:exceptionOccurred :: sd:ContractResult -~ pr:Exception;
```

There are a number of other concepts and ontological statements about contracts that we have developed in our SweetDeal Contract Ontology. In **Appendix B** we give the ADL version of the **full sd.daml ontology**, omitting most comments.

9. RULEML CONTRACTS WITH EXCEPTIONS USING THE PROCESS AND CONTRACT ONTOLOGIES

In this section, we newly show how to use the DAML+OIL process ontology, including about contracts and exceptions, as the definitions of the predicates of RuleML rules, where a ruleset represents part or all of a (draft or final) contract that has exceptions and exception handlers. RuleML rules thus refer to DAML+OIL process ontologies, partly drawn from PH content. This is one of the paper's three main contributions. It uses the other two: the technical approach to incorporating DAML+OIL/OWL ontologies into RuleML rules (section 5) and the PH ontologies in DAML+OIL (sections 7 and 8).

We illustrate by giving a detailed application scenario, including a long series of examples, of such a contract ruleset. The contract ruleset's rule-based contingency provisions include detecting and penalizing late delivery exceptions, thus providing means to deter or adjudicate a late delivery.

9.1. Outline of the Example/Scenario: Manufacturing Supply Chain Negotiation

Our example scenario refines a generic negotiation scenario that uses the SweetDeal approach. In the generic negotiation scenario, the contract ruleset is created and then modified during the course of negotiation between a buyer and seller. Contract rulebases are exchanged between the buyer and seller, as part of XML messages. The buyer has some business logic, a subset of which is specified and implemented in terms of rules, and some subset of those rules are exposed to be exchanged, and thus shared, with the seller. Likewise, the seller has business logic, a subset of which are rules, and a subset of those are exposed to be exchanged. The buyer and the seller may employ different rule systems to specify and implement their rule-based representation and inferencing.

In our particular example scenario, the above generic scenario is refined as follows. The buyer is a manufacturer and the seller is a supplier of parts. The buyer and seller applications/agents are each using a commercial rule system, but heterogeneous ones that belong to different major families of rule systems. The buyer's is a "production rule" system (i.e., descended from the OPS5 research system) that does forward-direction inferencing, e.g., Jess [19], while the seller's is a Prolog system that does backward-direction inferencing, e.g., XSB [47]. The buyer and the seller exchange negotiation messages. The early part of this sequence of exchanged messages includes a request for proposal. The last part of the sequence of exchanged messages includes finalization and acknowledgement of the agreement reached, including a purchase order. The middle part of the sequence of exchanged messages includes proposals and counter-proposals; this part is what we will focus on detailing below. Figures 6 and 7 illustrate.

To begin with:

- Buyer goes shopping (a.k.a. procurement).

The next three steps are more interesting.

- Seller sends a proposed contract.
- Buyer adds an exception handling provision for late delivery penalty, and sends the modified contract as a counter-proposal.
- Seller adds a replacement provision for late delivery risk premium (instead of a penalty), and sends the again-modified contract as a final offer. This provision illustrates that exception handling can itself be the subject of negotiation.

Later in this section, we will describe in detail the corresponding contract ruleset for each of these last three steps. In the examples below, DAML+OIL classes and properties, taken from the PH process ontology and contract (process) ontology, are used as predicate symbols.

One of the interesting capabilities enabled by our approach is that one can automatically do "what-if" (i.e., hypothetical-case) inferencing with the rules. Such what-if reasoning is a well-known desirable capability in contracting, and particularly in supply chain management. E.g., in the buyer's version of the contract:

- add facts about a hypothetical delivery date in the contract result
- ⇒ infer: the delivery is late,
- ⇒ infer: … which is an exception
- ⇒ infer: a late delivery penalty is owed.

One can also execute aspects of the contract via inferencing, e.g.:

- add facts of the actual contract result
- ⇒ infer: determine net payment owed.

9.2. Notation and Syntax:

RuleML, like most XML, is fairly verbose. For ease of human-readability, as well as save paper space, we give our RuleML examples in a Prolog-like syntax that maps straightforwardly to RuleML. More precisely, this syntax is IBM CommonRules V3.0 “SCLPfile” format, extended to support URI’s as logical predicate (and function) symbols and to support names for rule subsets (i.e., “modules”). Development is currently underway by the RuleML Initiative and the Joint Committee of a canonical RuleML human-oriented syntax – a concise ASCII string syntax to facilitate human reading and authoring – based in part on our SCLPfile syntax used here. Next, we detail that syntax.

“<-“ stands for implication, i.e., “if”. “;” ends a rule statement. The prefix “?” indicates a logical variable. “/*…*/” encloses a comment. “<…>” encloses a rule label (name) or rule module label. "{…}" encloses a rules module. Rule labels identify rules for editing and prioritized conflict handling, for example to facilitate the modular modification of contract provisions. Module labels are used to manage the merging of multiple rule modules to form a contract.

For example, the following fact in SCLPfile format

```
price(co123,50);
```

, which states that the price per unit in contract co123 is $50, has the following form in (SCLP) RuleML V0.8 XML syntax:

```
<fact><_head><_opr><rel>price</rel><_opr>
<tup><ind>co123</ind><ind>50</ind></tup>  </_head></fact>
```

The following rule in SCLPfile format

```
payment(?R,base,?Payment) <-

  sd:result(co123,?R) AND

  price(co123,?P) AND quantity(co123,?Q) AND

  multiply(?P,?Q,?Payment) ;
```

, which says that the base payment owed in the contract result is the price per unit multiplied by the quantity of units, has the following form in RuleML V0.8 XML syntax:

```
•<imp>
```

```
•  <_head> <atom> <_opr>
•    <rel> payment </rel> <_opr>
•      <tup> <var> R </var> <ind> base </ind> <var> Amount </var>
</tup>
•  </atom> </_head>
•  <_body> <andb> <atom> <_opr>
•    <rel href= "http//ccs.mit.edu/sd/sd.daml#result"/>
•         </_opr>
•           <tup> <ind> co123 </ind> <var> R </var>
•          </tup> </atom>
        …
   </andb>  </_body>  </imp>
```

We omitted showing explicitly, above, the body atoms past the first one; hence, the use of "…". Note that the predicate "http://ccs.mit.edu/sd/sd.daml#result" is a URI that references a property defined in our DAML+OIL contract ontology. In RuleML's XML syntax, we see the reference being made by use of the "rel" element's "href" attribute.

Namespace Qualification: Since repeatedly spelling out "http://ccs.mit.edu/sd/sd.daml#" is undesirably verbose, in our examples below, we instead use the prefix "**sd:**" to stand for that contract ontology name. In the XML syntactic mechanics, this just requires declaring a namespace in the RuleML document's header (similar to what we saw in the DAML+OIL header shown for the PH process ontology in section 5). Likewise, we use the prefix "**pr:**" to stand for "http://ccs.mit.edu/sd/pr.daml#"

9.3. Example 1: Supplier Proposal

Let's begin with an example draft contract `co123` where buyer Acme is purchasing 100 units of plastic product #425 from seller Plastics Etc. at $50 per unit. Acme requires Plastics Etc. to ship the product no later than three days after the order is placed [21] We specify this draft contract as the following rulebase (i.e., set of rules):

```
sd:Contract(co123);

sd:specFor(co123,co123_process);

sd:BuyWithBilateralNegotiation(co123_process);

sd:result(co123,co123_res);
```

[21] Here we use a relative date (e.g. 3) rather than an absolute date (e.g. 2002-04-02), for sake of simplicity and because the rule engine that we are using in our prototype (IBM CommonRules) does not (yet) provide convenient date arithmetic functions.

```
buyer(co123,acme);

seller(co123,plastics_etc);

product(co123,plastic425);

shippingDate(co123,3); /*i.e. 3 days after the order is placed*/

price(co123,50);

quantity(co123,100);

/* base payment = price * quantity  */

payment(?R,base,?Payment) <-

  sd:result(co123,?R) AND

  price(co123,?P) AND quantity(co123,?Q) AND

  multiply(?P,?Q,?Payment) ;
```

9.4. Example 2: Buyer's Counter-Proposal, with Late Delivery Penalty

Continuing our example, suppose the buyer wants to include a contract provision to penalize late delivery. (Alternatively, the seller might want to include such a provision in order to reassure the buyer.) This constitutes a counter-proposal during the negotiation. First, we add some rules to declare (i.e., specify) that this contract has an exception instance `e1` that is an instance of the LateDelivery class from the process ontology:

```
pr:hasException(co123_process,e1);

pr:LateDelivery(e1);
```

Note that the actual occurrence of an exception is associated with a contract result, as opposed to its potential occurrence which is associated with the contract (agreement)'s process. hasException specifies the potential occurrence. We will see below more about the actual occurrence.

Next, we give a rules module (i.e., a set of additional rules to include in the overall draft contract ruleset) that specifies a basic kind of exception handler process – to detect the late delivery.

In our approach, exception handler processes themselves may be rule-based (in part or totally), although in general they need not be rule-based at all. The exception handler detectLateDelivery is rule-based in this example. Below, the variable ?CO stands for a contract, ?R for a contract result, ?EI for an exception instance, ?PI for a process instance, ?COD for a promised contract shipping date, and ?RD for a contract result's actual shipping date.

```
<detectLateDelivery_module> {
```

```
/* detectLateDelivery is an instance of
DetectPrerequisiteViolation (and thus of DetectException,
ExceptionHandler, and Process) */

pr:DetectPrerequisiteViolation(detectLateDelivery) ;

/* detectLateDelivery is intended to detect exceptions of class
LateDelivery */

sd:detectsException(detectLateDelivery, pr:LateDelivery);

/* a rule defines the actual occurrence of a late delivery in a
contract result */

<detectLateDelivery_def> sd:exceptionOccurred(?R, ?EI) <-

  sd:specFor(?CO,?PI) AND

  pr:hasException(?PI,?EI) AND

  pr:LateDelivery(?EI) AND

  pr:isHandledBy(?EI, detectLateDelivery) AND

  sd:result(?CO,?R) AND

  shippingDate(?CO,?COD) AND shippingDate(?R,?RD) AND

  greaterThan(?RD,?COD) ;

}
```

Then we add the following rule to the contract to specify detectLateDelivery as a handler for e1:

```
<detectLateDeliveryHandlesIt(e1)>
pr:isHandledBy(e1,detectLateDelivery);
```

Merely detecting late delivery is not enough; the buyer also wants to get a penalty (partial refund) if late delivery occurs. Continuing our example, we next give a rules module to specify a penalty of $200 per day late, via an exception handler process lateDeliveryPenalty. Again, this handler is itself rule-based.

```
lateDeliveryPenalty_module {


/* lateDeliveryPenalty is an instance of PenalizeForContingency
(and thus of AvoidException, ExceptionHandler, and Process) */

pr:PenalizeForContingency(lateDeliveryPenalty) ;

/* lateDeliveryPenalty is intended to avoid exceptions of class
LateDelivery. */

sd:avoidsException(lateDeliveryPenalty, pr:LateDelivery);

/* penalty = - overdueDays * 200 ; (negative payment by buyer) */
```

```
<lateDeliveryPenalty_def> payment(?R,contingentPenalty, ?Penalty)
 <-
  sd:specFor(?CO,?PI) AND pr:hasException(?PI,?EI) AND
  pr:isHandledBy(?EI,lateDeliveryPenalty) AND
  sd:result(?CO,?R) AND sd:exceptionOccurred(?R,?EI) AND
  shippingDate(?CO,?CODate) AND shippingDate(?R,?RDate) AND
  subtract(?RDate,?CODate,?OverdueDays) AND
  multiply(?OverdueDays, 200, ?Res1) AND
  multiply(?Res1, -1, ?Penalty) ;
}
```

We add a rule to specify lateDeliveryPenalty as a handler for e1:

```
<lateDeliveryPenaltyHandlesIt(e1)>
  pr:isHandledBy(e1,lateDeliveryPenalty);
```

During contract execution, if Plastics Etc. ships its product 8 days after the order is placed (i.e. 5 days later than the agreed-upon date), then the rules about detectLateDelivery (i.e., in the detectLateDelivery module) will entail that late delivery exception has occurred, which will trigger the lateDeliveryPenalty handler to impose (i.e., entail) a penalty of $200 per day late, totaling $1000.

More precisely, suppose we represent the contract result as the ruleset formed by adding (to the above contract) the following "result" fact:

```
shippingdate(co123_res, 8) ;
```

Then the contract result ruleset entails various conclusions, in particular

```
sd:exceptionOccurred(co123_res,e1) ;

payment(co123_res, contingentPenalty, -1000) ;
```

Our SweetRules prototype system, which implements SCLP RuleML inferencing, can generate these conclusions automatically; thus so can our SweetDeal prototype system which employs SweetRules as a component.

9.5. Example 3: Seller's Counter-Counter-Proposal, with Late Delivery Risk Payment instead

Next, we continue our example further. In so doing, we (relatively briefly, due to space constraints) illustrate how to use prioritized conflict handling, enabled by the courteous feature of SCLP RuleML, to modularly modify the contract provisions, e.g., during bilateral negotiation. Suppose the seller wants to specify that the late delivery exception should be handled *instead* by the handler lateDeliveryRiskPayment, which imposes an up-front insurance-like discount to compensate for the risk of late delivery, basing risk upon a historical average probability distribution (defined separately) of delivery lateness. This constitutes a counter-counter-proposal, which is, say, the seller's final offer in the negotiation. The risk payment provision conflicts with the penalty provision. The prioritized conflict handling capability of the Courteous feature of SCLP enables the risk payment provision to specified simply by adding a new module, rather than having to modify any of the rules in the previous contract proposal. First, we define a rules module for the risk payment handler:

```
lateDeliveryRiskPayment_module {

/* lateDeliveryRiskPayment is an instance of AvoidException  (and
thus of ExceptionHandler, and Process) */

pr:AvoidException( lateDeliveryRiskPayment) ;

/* lateDeliveryRiskPayment is intended to avoid exceptions of
class LateDelivery. */

sd:avoidsException( lateDeliveryRiskPayment, sd:LateDelivery) ;

/* penalty = - expected_lateness * 200 (negative payment by
buyer) */

<lateDeliveryRiskPayment_def>

payment(?R, contingentRiskPayment, ?Penalty) <-

  sd:specFor(?CO,?PI) AND sd:hasException(?PI,?EI) AND

  pr:isHandledBy(?EI, lateDeliveryRiskPayment) AND

  sd:result(?CO,?R) AND
```

```
  historical_probabilistically_expected_lateness(?CO,
                                              ?EOverdueDays) AND
  multiply(?EOverdueDays, 200, ?Res1) AND
  multiply(?Res1, -1, ?Penalty);
}
```

Then we add a rule to specify lateDeliveryRiskPayment as a handler for e1:

```
<lateDeliveryRiskPaymentHandlesIt(e1)>
  pr:isHandledBy(e1, lateDeliveryRiskPayment);
```

Next, we give some rules that use prioritized conflict handling to specify that late deliveries should be avoided by lateDeliveryRiskPayment rather than by any other candidate avoid-type exception handlers for the late delivery exception (here, simply, lateDeliveryPenalty). We specify this using a combination of a MUTEX statement and an overrides statement that gives the lateDeliveryRiskPaymentHandlesIt(e1) rule higher priority than the lateDeliveryPenaltyHandlesIt(e1) rule.

```
/* There is at most one avoid handler for a given exception
instance. */
/* This is expressed as a MUTual EXclusion between two potential
conclusions, given certain other preconditions. */
```

```
/* The mutex is a consistency-type integrity constraint, which is
enforced by the courteous aspect of the semantics of the rule KR.
*/

MUTEX

  pr:isHandledBy(?EI, ?EHandler1) AND

  pr:isHandledBy(?EI, ?EHandler2)

GIVEN

  sd:AvoidException(?EHandler1) AND

  sd:AvoidException(?EHandler2) AND

  notEquals(?Ehandler1, ?EHandler2);

/* The rule lateDeliveryRiskPaymentHandlesIt(e1) has higher
priority than the rule lateDeliveryPenaltyHandlesIt(e1).  */

overrides(lateDeliveryRiskPaymentHandlesIt(e1),

          lateDeliveryPenaltyHandlesIt(e1) ) ;
```

Now suppose the probabilistically expected lateness of the delivery (before actual contract execution) is 3 days. I.e., suppose the contract also includes the following fact.

```
historical_probabilistically_expected_lateness(co123, 3) ;
```

If upon execution the modified-contract's result facts are as before – i.e., delivery is 5 days late – then the modified-contract's result entails as conclusions that the late delivery will be handled by the up-front risk payment of $600 = (3 days * $200).

```
payment(co123_res, contingentRiskPayment, -600) ;
```

The modified-contract's result does *not* entail that late delivery is handled by the penalty of $1000 – as it should not. The courteous aspect of the rules knowledge representation has properly taken care of the prioritized conflict handling to enforce that the new higher-priority contract provision about risk payment dominates the provision about penalty.

10. SWEETDEAL PROTOTYPE SYSTEM: OVERVIEW

Next we describe, briefly for reasons of focus and space, the current SweetDeal prototype system at MIT. The prototype includes capabilities for contract authoring, communication, and inferencing/execution. The inferencing capability is the one most pertinent to this paper. The prototype uses the SweetRules toolset for SCLP RuleML to perform such rule inferencing on the contract rulesets, e.g., the examples in section 9.

Contract Fragments and Repositories: SweetDeal also includes a simple form of queryable repositories of contract rule modules and process ontologies, as well as some relatively simple market agents that communicate, negotiate, evaluate, etc. the contracts. A software market agent can create contract proposals in a semi-automated manner (i.e., in support of a human user) by combining (1) reusable modular contract provisions, called *contract fragments*, from a queryable *contract repository*, along with (2) process knowledge from a queryable *process repository*. This market agent's capabilities supports the negotiation process in an overall interaction architecture for an agent marketplace with such rule-based contracts.

SweetRules: SweetDeal is part of our larger effort SWEET, acronym for "Semantic WEb Enabling Technology", and is prototyped on top of the SweetRules toolkit [21][22][24][25], implemented in Java and XSLT [52]. SweetRules implements SCLP RuleML inferencing and bi-directional translation of (SCLP) RuleML to and from multiple heterogeneous rule systems including: XSB, a popular and powerful open-source Prolog rule system [47]; Jess, a popular open-source forward production-rule system in Java [19]; the IBM CommonRules rule engine, a forward SCLP system [8]; Smodels, a forward logic-program rule engine [38]; and Common Logic (formerly known as Knowledge Interchange Format (KIF)), an emerging industry standard for knowledge interchange in classical logic [9].

Invocation: Rules representing contract provisions can invoke associated business processes – e.g., with side effects – via the action/effector procedural attachments capability and expressiveness provided by the Situated feature of SCLP. E.g., an exception handling rule can invoke an associated exception handler business process. (We did not use that capability in this paper's examples.)

Rule-based business processes: In general, these business processes could be any kind of procedure, e.g., a Java method or a Web Service. In particular, they may themselves be partly or completely specified via rules (executably).

11. CURRENT AND FUTURE WORK

In the second half of section 1, we described the main novel contributions of this work; that serves in lieu of a **Conclusions** section. Earlier at various points, we discussed some directions of current work – notably, other capabilities of the SweetDeal approach and system (section 10) and Description Logic Programs (section 5). Next, we further discuss interesting research directions of current and future work.

Theory and Standards for Combining LP/RuleML + DL/OWL: An important aspect of ontologies, mentioned earlier, is to develop the theory of combining rules on top of ontologies, including expressive union and intersection, semantics, proof theory, algorithms, and computational complexity. [26] gives our initial development of this theory, based on the Description Logic Programs KR; more is in progress. Objectives include to enable (1) completeness/consistency, and (2) efficiency in specification and inferencing. Particularly salient as an early step is to develop theory and techniques to handle existentially quantified property restrictions cf. DL, e.g., via skolemization. Note that our example ontologies (pr.daml and sd.daml) include such existentials.

The DLP-based theory of combination also provides a principled basis for unifying the *syntax* of the rules more closely with that of the ontologies, e.g., using RDF for both. Our current work on *Semantic Web Rules Language (SWRL)* [6], a simplified subset of RuleML tightly combined on top of OWL, is a step in this direction. Based largely on requirements defined by DAML [11] and the Joint Committee [30], its first version adds to OWL-DL the Datalog Horn rules expressive subset of LP/FOL so as to extend the expressive power of full DL. It uses essentially the same reference approach as we give in this paper. The resulting semantics can be viewed purely in terms of FOL. Since both DL and Horn are subsets of FOL, their union is as well. However, that expressive union is not well studied previously and needs theoretical, as well as practical, investigation.

Another technical issue is to design better module and import mechanisms for (official) RuleML/SWRL and OWL – currently, this aspect is rudimentary in both. This is particularly important since the semantics of any nonmonotonic KR requires a well-defined boundary to scope the set of premises. The relevant RuleML design aspects are currently under development.

SweetDeal Prototype Architecture: As we mentioned earlier (especially in section 10), the SweetDeal prototype architecture includes several other aspects. These provide directions for current and future work. One is the queryable repository for contract fragments, including new technical aspects for RuleML to better enable modules inclusion and naming. Another is architecture relationships of overall contracting software agents (i.e., that act as a buyer or seller or intermediary) to infrastructure specifically for contract *rules* inferencing/execution, translation, communication, authoring/editing, storage/retrieval, etc. Part of this is to develop more sophisticated techniques for invoking business processes, e.g., Web Services in particular, by invoking Situated LP procedural attachments. Another aspect is tying in to agent negotiation strategies.

More Example Scenarios: A direction for future work is to develop more and longer example scenarios and test them out by running them using SweetDeal/SweetRules together with tools for DAML+OIL/OWL and, later, also with tools for reasoning specifically about process knowledge.

Rule KR Technologies for Semantic Web Services: A major direction for our current and future work is relationships of our SweetDeal approach and its elements (rules, ontologies, process knowledge) to the area of Semantic Web Services (the convergence of Semantic Web and Web Services). One aspect of this is further developing rule KR technologies for use in Semantic Web Services, where services may use rules plus ontologies, and rules may call services.

Business Applications and Implications of Semantic Web Services (SWS): E-contracting is an important potential business application area for Semantic Web Services. In particular, a key direction of current work is to apply our SweetDeal approach to what we call the **deal layer of SWS**: i.e., contracts about Web Services, where the contractual agreement is described semantically. This is a crucial area for Web Services overall, which has been addressed at only a relatively shallow level previously, except for the SweetDeal approach. Our current work includes defining contracting aspects of requirements and architecture for SWSI [44]. Other business application areas for SWS, particularly that combine rules with ontologies, that we are investigating include: financial information integration, e.g., ECOIN [17] [18]; travel packages; trust management; and business policies.

Relating to emerging WS and SWS standards/pieces: There are a number of other relevant emerging WS and SWS standards/pieces. These include existing basic Web Services standards and techniques, e.g., WSDL invocations [51], SOAP messaging [45], WSBPEL (procedural) process models [49], and UDDI advertising/discovery [46]. These also include general efforts on emerging *Semantic* Web Services techniques and applications , e.g., DAML-Services [12], Web Services Modeling Framework (WSMF) [14], and the Semantic Web Services Initiative (SWSI) [44], as well as Web Services Choreography [50]. There are also existing and emerging standards for general-purpose e-business/agent communication, e.g., ebXML [15] and FIPA's Agent Communication Language [16].

Legal Aspects of Contracting: A direction for future work is how to incorporate more about legal aspects of contracting into our approach, including to connect to the Legal XML emerging standards effort [34], particularly LegalXML eContracts [20].

Process Ontologies, including from MIT Process Handbook: Other directions involve ontologies. One is to further develop the DAML+OIL/OWL ontology for business processes, e.g., our current work includes drawing on the Process Handbook. Another is to further develop the contract ontology. Currently, we are investigating how to formalize more deeply the relationship between a contract rulebase and a rule-based handler process. An open source version of the PH is planned. Building upon our work here we aim overall to enable the open source PH to be integrated with other Semantic Web knowledge sources for use in Semantic Web Services and other knowledge-based applications, including via automatic translation in and out of the PH.

Default Inheritance: Finally, there is the challenge of how to cope with the issue of *default* inheritance in regard to OWL/DL and also to the Process Handbook. Default inheritance (i.e., where inherited values may be overridden or cancelled) is used in most object-oriented/frame-based systems, including the Process Handbook. In current work (e.g., [5]), we are taking an approach to default inheritance using the prioritized conflict handling capability provided by the courteous feature of SCLP, and applying it to representing the Process Handbook in particular.

APPENDIX A. MORE PROCESS ONTOLOGY (pr.daml in ADL syntax)

```
hasException ::  Process -~ Exception;

isHandledBy ::  Exception -~ ExceptionHandler;

Process : ;

Exception : (exists isHandledBy . NotifyAboutException);

ExceptionHandler : Process;

Buy : Process;
```

```
Buy : (exists hasException . SubcontractorIsLate);

Buy : (exists hasException . LowQualityResult);

Buy : (exists hasException . SubcontractorDropsTask);

Buy : (exists hasException . SubcontractorChangesCost);

Buy : (exists hasException . ContractorDoesNotPay);

Buy : (exists hasException . ContractorCancelsTask);

BuyOverInternet : Buy;

BuyinElectronicStore : BuyOverInternet;

BuyInElectronicStoreUsingAuction : BuyinElectronicStore;

BuyInElectronicStoreUsingPostedPrices : BuyinElectronicStore;

SystemCommitmentViolation : Exception;

  /* Violations of commitments made by the system operator

     to create an environment well-suited to the task at hand. */

AgentCommitmentViolation : Exception;

  /* Violations of commitments agents make to each other. */

InfrastructureCommitmentViolation : Exception;

  /* Violations of commitments made by the infrastructure to
```

```
     provide dependable communication and computation. */

MatchmakerViolation : AgentCommitmentViolation;

ContractorMatchmakerCollusion : MatchmakerViolation;

  /* collusion between contractor (ex. buyer) and matchmaker

     (ex. auctioneer) */

ContractorViolation : AgentCommitmentViolation;

ContractorDoesNotPay : ContractorViolation;

ContractorDoesNotPay : (exists isHandledBy . CheckCreditLine);

ContractorDoesNotPay : (exists isHandledBy .
DetectPrerequisiteViolation);

ContractorDoesNotPay : (exists isHandledBy .
ProvideSafeExchangeProtocols);

FraudulentCreditPayment : ContractorDoesNotPay;

ContractorCancelsTask : ContractorViolation;

ContractorCancelsTask : (exists isHandledBy .
DetectViaNotification);

ContractorCancelsTask : (exists isHandledBy .
PenalizeForDecommitment);
```

```
SubcontractorViolation : AgentCommitmentViolation;

  /* subcontractor (seller) violates commitments */

SubcontractorIsLate : SubcontractorViolation;

SubcontractorIsLate : (exists isHandledBy .
DetectPrerequisiteViolation);

SubcontractorIsLate : (exists isHandledBy .
PenalizeForContingency);

SubcontractorIsLate : (exists isHandledBy .
PenalizeUsingRiskPayments);

LowQualityResult : SubcontractorViolation;

LowQualityResult : (exists isHandledBy .
DetectPrerequisiteViolation);

LowQualityResult : (exists isHandledBy . PenalizeForContingency);

LowQualityResult : (exists isHandledBy .
PenalizeUsingRiskPayments);

SubcontractorDropsTask : SubcontractorViolation;

SubcontractorDropsTask : (exists isHandledBy .
PenalizeForDecommitment);
```

```
SubcontractorDropsTask : (exists isHandledBy .
DetectViaNotification);

SubcontractorChangesCost : SubcontractorViolation;

FindException : ExceptionHandler;

FixException : ExceptionHandler;

AnticipateException : FindException;

MaintainReputationInformation : AnticipateException;

CheckCreditLine : AnticipateException;

DetectException : FindException;

DetectPrerequisiteViolation : DetectException;

DetectTimeout : DetectException;

MonitorUsingSentinels : DetectException;

DetectViaNotification : DetectException;

ResolveException : FixException;

NotifyAboutException : ResolveException;

NotifyAboutExceptionUsingEmail : NotifyAbouException;

NotifyAboutExceptionUsingPager : NotifyAbouException;
```

```
AvoidException : FixException;

ProvideSafeExchangeProtocols : AvoidException;

ProvideEscrowService : ProvideSafeExchangeProtocols;

ProvideEscrowServiceWithBuyerCertification :
ProvideEscrowService;

ProvideEscrowServiceWithThirdParty : ProvideEscrowService;

ProvideIncentives : AvoidException;

ProvidePenalties : ProvideIncentives;

PenalizePoorBehavior : ProvidePenalties;

PenalizeForDecommitment : PenalizePoorBehavior;

PenalizeForContingency : PenalizePoorBehavior;

PenalizeUsingRiskPayments : PenalizePoorBehavior;
```

APPENDIX B. MORE CONTRACT ONTOLOGY (sd.daml in ADL syntax)

```
sd:Contract : (mincard-1 . sd:specFor);

sd:specFor :: sd:Contract -~ pr:Process;

sd:ContractForOneProcess : sd:Contract;

sd:ContractForOneProcess : (card-1 sd:specFor);
```

```
sd:ContractResult : ;

sd:result :: sd:Contract -~ sd:ContractResult;

sd:exceptionOccurred :: sd:ContractResult -~ pr:Exception;

sd:exceptionLikely :: sd:ContractResult -~ pr:Exception;

sd:detectsClass :: pr:DetectException -~ daml:Class;

sd:anticipatesClass :: pr:AnticipateException -~ daml:Class;

sd:avoidsClass :: pr:AvoidException -~ daml:Class;

sd:resolvesClass :: pr:ResolveException -~ daml:Class;

sd:handles :: pr:ExceptionHandler -~ pr:Exception;

sd:detects :: pr:DetectException -~ pr:Exception;

sd:anticipates :: pr:AnticipateException -~ pr:Exception;

sd:avoids :: pr:AvoidException -~ pr:Exception;

sd:resolves :: pr:ResolveException -~ pr:Exception;
```

ACKNOWLEDGEMENTS

Thanks to our MIT Sloan colleagues Mark Klein, Chrysanthos Dellarocas, Thomas Malone, Peyman Faratin, and John Quimby, as well as Abraham Bernstein, for useful discussions about the Process Handbook and representing contract exceptions there. Thanks to anonymous reviewers for helpful comments. Funding support was provided by the Center for eBusiness @ MIT (Vision Fund award "Automatically Discovering, Selecting, and Combining Web Services Using Business Process Descriptions"), and by the DARPA Agent Markup Language (DAML) program (award "Tools for Supporting Intelligent Information Annotation, Sharing, and Retrieval").

REFERENCES

1. Antoniou, G. "Nonmonotonic Rule Systems using Ontologies". Proc. *Intl. Wksh. on Rule Markup Languages for Business Rules on the Semantic Web*, held at 1st Intl. Semantic Web Conf., 2002. http://tmitwww.tm.tue.nl/staff/gwagner/RuleML-BR-SW.html. Accessed on Nov. 17, 2003.

2. Berners-Lee, T. *et al*. Semantic Web Stack Diagram. http://www.w3.org/DesignIssues/diagrams/sw-stack-2002.png. Accessed on Nov. 17, 2003.

3. Bernstein, A. "The Product Workbench: An Environment for the Mass-Customization of Production-Processes," Proc. *8th Workshop on Information Technology and Systems (WITS '98)*, 1998. http://www.ifi.unizh.ch/ddis/bernstein.0.html. Accessed Nov. 17, 2003.

4. Bernstein, A. "How Can Cooperative Work Tools Support Dynamic Group Processes? Bridging the Specificity Frontier". Proc. *Conf. on Computer Supported Cooperative Work (CSCW-2000).* Philadelphia, PA: ACM Press, 2000.

5. Bernstein, A. and Grosof, B.N. (alphabetic order) "Beyond Monotonic Inheritance: Towards Semantic Web Process Ontologies". MIT Sloan Working Paper, Aug. 2003. Submitted. http://ebusiness.mit.edu/bgrosof. Accessed Nov. 17, 2003.

6. Boley, H.; Dean, M.; Grosof, B.N.; Horrocks, I.; Patel-Schneider, P.; and Tabet, S. (alphabetic order) "SWRL: A Semantic Web Rules Language Combining OWL and RuleML". Version of Nov. 14, 2003. http://www.daml.org/rules/proposal. Accessed on Nov. 17, 2003.

7. Boley, H.; Grosof, B.N.; Tabet, S.; *et al* (alphabetic order). RuleML Initiative. http://www.ruleml.org and http://ebusiness.mit.edu/bgrosof/#RuleML. Accessed on Nov. 17, 2003.

8. Chan, H.Y.; Grosof, B.N.; *et al* (alphabetic order). IBM CommonRules. http://www.alphaworks.ibm.com and http://www.research.ibm.com/rules. Accessed on Nov. 17, 2003.

9. Common Logic http://cl.tamu.edu/, formerly known as Knowledge Interchange Format http://logic.stanford.edu/kif and http://www.cs.umbc.edu/kif . Accessed on Nov. 17, 2003.

10. DAML+OIL Reference (Mar. 2001). http://www.w3.org/TR/daml+oil-reference. Accessed on Nov. 17, 2003.

11. DARPA Agent Markup Language Program. http://www.daml.org. This includes, in part, DAML-Rules http://www.daml.org/rules and DAML-Services http://www.daml.org/services. Accessed on Nov. 17. 2003.

12. DAML Services Coalition (alphabetically A. Ankolekar; M. Burstein; J. Hobbs, O. Lassila; D. Martin; S. McIlraith; S. Narayanan; M. Paolucci; T. Payne; K. Sycara; H. Zeng). ``DAML-S: Semantic Markup for Web Services", Proc. *International Semantic Web Working Symposium (SWWS)*, 2001; and more info at http://www.daml.org/services. Accessed on Nov. 17, 2003.

13. Dellarocas, C. "A Coordination Perspective on Software Architecture: Towards a design Handbook for Integrating Software Components," Ph.D. Thesis, Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA, 1996.

14. Fensel, D. and Bussler, C. "The Web Service Modeling Framework (WSMF)". White paper, 2002. http://informatik.uibk.ac.at/users/c70385/wese/publications.html. Accessed on Nov. 17, 2003.

15. ebXML (ebusiness XML) standards effort, http://www.oasis-open.org. Accessed on Nov. 17. 2003.

16. FIPA (Foundation for Intelligent Physical Agents), Agent Communication Language standards effort, http://www.fipa.org. Accessed on Nov. 17. 2003.

17. Firat, A.; Madnick, S.; and Grosof, B.N. "Knowledge Integration to Overcome Ontological Heterogeneity: Challenges from Financial Information Systems". Proc. *23rd Intl. Conf. on Information Systems (ICIS-2002)*, Dec. 2002. http://dsi.esade.es/icis2002. Accessed Nov. 17, 2003.

18. Firat, A.; Madnick, S.; and Grosof, B.N. "Financial Information Integration in the presence of Equational Ontological Conflicts". Proc. *12th Workshop on Information*

Technologies and Systems (WITS-2002), Dec. 2002.
http://ecweb.ecenter.smu.edu/wits02. Accessed Nov. 17. 2003.

19. Friedman-Hill, E. *et al*.Jess (Java Expert System Shell).
http://herzberg.ca.sandia.gov/jess. Accessed on Nov. 17. 2003.

20. Greenwood, D. *et* al. Oasis Legal XML eContracts Technical Committee.
http://www.oasis-open.org/committees/legalxml-econtracts. Accessed Nov. 17, 2003.

21. Grosof, B.N. "Representing E-Business Rules for Rules for the Semantic Web: Situated Courteous Logic Programs in RuleML". Proc. *11th Wksh. on Information Technology and Systems (WITS '01)*, 2001. http://www.busi.mun.ca/parsons/wits2001. Accessed on Nov. 17, 2003.

22. Grosof, B.N. "Representing e-commerce rules via Situated Courteous Logic Programs in RuleML". *Electronic Commerce Research and Applications* journal, (to appear, 2004).

23. Grosof, B.N. "A Semantic Web Services Primer." (White paper and portal.)
http://ebusiness.mit.edu/bgrosof/#SWSPrimer. Accessed on Nov. 17, 2003.

24. Grosof, B.N.; Gandhe, M.D.; and Finin, T.W. "SweetJess: Translating DamlRuleML to Jess". Proc. Intl. Wksh. on Rule Markup Languages for Business Rules on the Semantic Web, held at 1st Intl. Semantic Web Conf., 2002.
http://tmitwww.tm.tue.nl/staff/gwagner/RuleML-BR-SW.html. Accessed on Nov. 17, 2003.

25. Grosof, B.N.; Gandhe, M.D.; and Finin, T.W., "SweetJess: Inferencing in Situated Courteous RuleML Via Translation to and from Jess Rules". MIT Sloan Working Paper, version of May 2003. Submitted. http://ebusiness.mit.edu/bgrosof. Accessed on Nov. 17, 2003.

26. Grosof, B.N. and Horrocks, I. "Description Logic Programs: Combining Logic Programs with Description Logic". Proc. *12th Intl. Conf. on World Wide Web (WWW-2003)*, 2003. http://www.www2003.org/cdrom/index.html. Accessed on Nov. 17. 2003.

27. Grosof, B.N.; Labrou, Y.; and Chan, H.Y. "A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML". New York, NY: Proc. *1st ACM Conf. on Electronic Commerce (EC-99)*, ACM Press, 1999. http://www.research.ibm.com/iac/ec99. Accessed on Nov. 17, 2003.

28. Grosof, B.N. and Poon, T.C. "Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions". Proc. *Intl. Wksh. on Rule Markup Languages for Business Rules on the Semantic Web*, held in conjunction with the 1st Intl. Semantic Web Conf., Jun. 2002. http://tmitwww.tm.tue.nl/staff/gwagner/RuleML-BR-SW.html. Accessed on Nov. 17, 2003.

29. Grosof, B.N. and Poon, T.C. "SweetDeal: Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions". Proc. *12th Intl. Conf. on the World Wide Web (WWW-2003)*, May 2003. http://www.www2003.org/cdrom/index.html. Accessed on Nov. 17. 2003.

30. Joint US/EU ad hoc Agent Markup Language Committee. http://www.daml.org/committee. Accessed on Nov. 17, 2003.

31. Klein, M.; Dellarocas, C.; and Rodríguez-Aguilar, J.A. “A Knowledge-Based Methodology for Designing Robust Multi-Agent Systems.” Proc. *Intl. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2002. http://www.aamas-conference.org. Accessed on Nov. 17, 2002.

32. Lassila, O.; Swick, R; *et al.* (eds.) Resource Description Framework (RDF). http://www.w3.org/RDF. Accessed on Nov. 17, 2003.

33. Lee, J.; Gruninger, M.; Jin, Y.; Malone, T.; Tate, A.; and Yost, G. "The PIF Process Interchange Format and Framework Version 1.1," Proc. *Workshop on Ontological Engineering*, held at *European Conference on Artificial Intelligence (ECAI '96)*, 1996. Updated version 1.2 available at http://ccs.mit.edu/pif. Accessed on Nov. 19, 2003.

34. Legal XML, http://www.oasis-open.org. Accessed on Nov. 17, 2003.

35. Malone, T.W.; Crowston, K.; and Herman, G. (eds.) *Organizing Business Knowledge: The MIT Process Handbook*. Cambridge, MA: MIT Press, 2003.

36. Malone, T.W.; Crowston, K.; Lee, J.; Pentland, B.; Dellarocas, C.; Wyner, G.; Quimby, J.; Osborn, C.S.; Bernstein, A.; Herman, G.; Klein, M.; and O’Donnell, E. “Tools for inventing organizations: toward a handbook of organizational processes.” *Management Science*, 45, 3 (1999), 425-443.

37. MIT Process Handbook. http://ccs.mit.edu/ph. Accessed on Nov. 17, 2003.

38. Niemela, I. and Simons, P. Smodels (version 1). http://www.tcs.hut.fi/Software/smodels. Accessed on Nov. 17, 2003.

39. Oasis http://www.oasis-open.org. Accessed on Nov. 17, 2003.

40. OWL (Ontology Working Language), from W3C Web-Ontologies Working Group. http://www.w3.org/2001/sw/webont . Accessed on Nov. 17, 2003.

41. Reeves, D.M.; Wellman, M.P.; and Grosof, B.N. "Automated negotiation from declarative contract descriptions". *Computational Intelligence* 18, 4, special issue on Agent Technology for Electronic Commerce, (Nov. 2002), 482-500.

42. Schlenoff, C.; Gruninger, M.; Tissot, F., Valois, J., Lubell, J., and Lee, J. "The Process Specification Language (PSL): Overview and Version 1.0 Specification," NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD, 2000.

43. Semantic Web Activity of the World Wide Web Consortium. http://www.w3.org/2001/sw. Accessed on Nov. 17, 2003.

44. Semantic Web Services Initiative (SWSI). http://www.swsi.org. Accessed on Nov. 17, 2003.

45. SOAP, http://www.w3.org/2000/xp/Group and http://www.w3.org/2002/ws. Accessed on Nov. 17. 2003.

46. UDDI (Universal Description, Discovery, and Integration). http://www.uddi.org. Accessed on Nov. 17. 2003.

47. Warren, D. *et al*. XSB logic programming system. http://xsb.sourceforge.net. Accessed on Nov. 17, 2003.

48. Web Services Activity of the World Wide Web Consortium. http://www.w3.org/2002/ws. Accessed on Nov. 17, 2003.

49. Web Services Business Process Execution Language (WSBPEL). http://www.oasis-open.org. Accessed on Nov. 17. 2003.

50. Web Services Choreography Working Group. http://www.w3.org/2002/ws/chor. Accessed on Nov. 17, 2003.

51. WSDL (Web Service Definition Language), http://www.w3.org/2002/ws and www.w3.org/TR/wsdl. Accessed on Nov. 17. 2003.

52. XSLT (eXtensible Stylesheet Language Transformations). http://www.w3.org/Style/XSL and http://www.w3.org/TR/xslt. Accessed on Nov. 17. 2003.

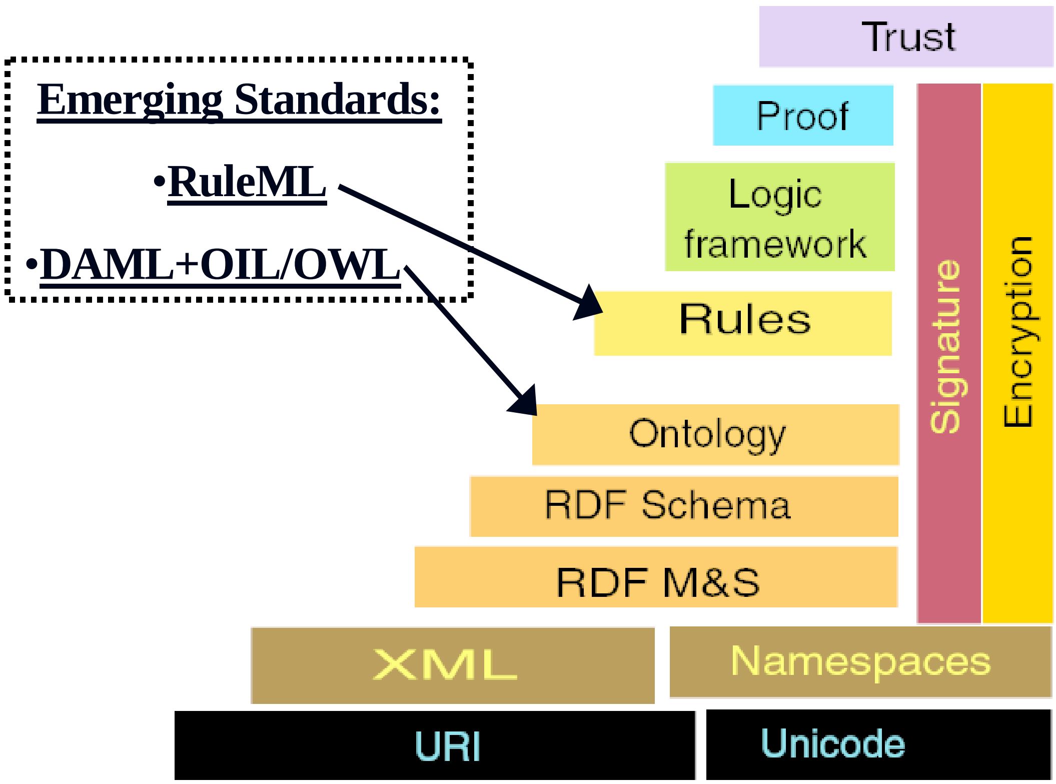

Figure 1: W3C Semantic Web "Stack": Standardization Steps

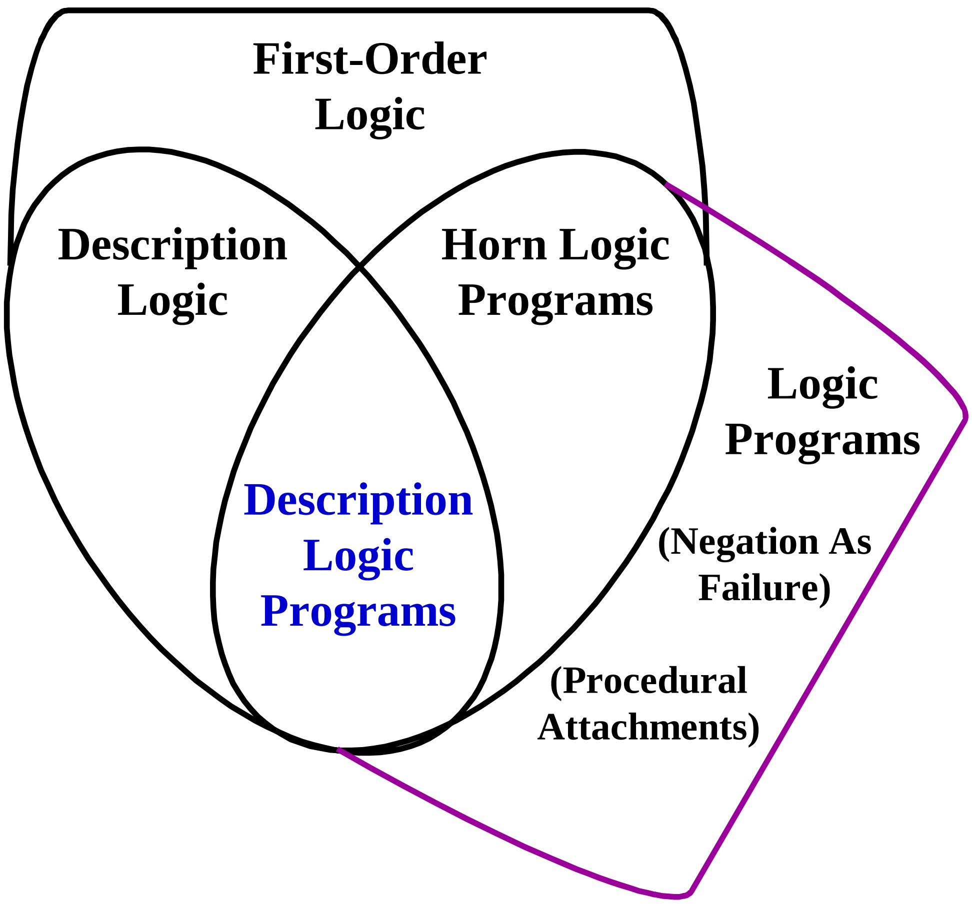

Figure 2: Venn Diagram of Expressive Overlaps between KR's

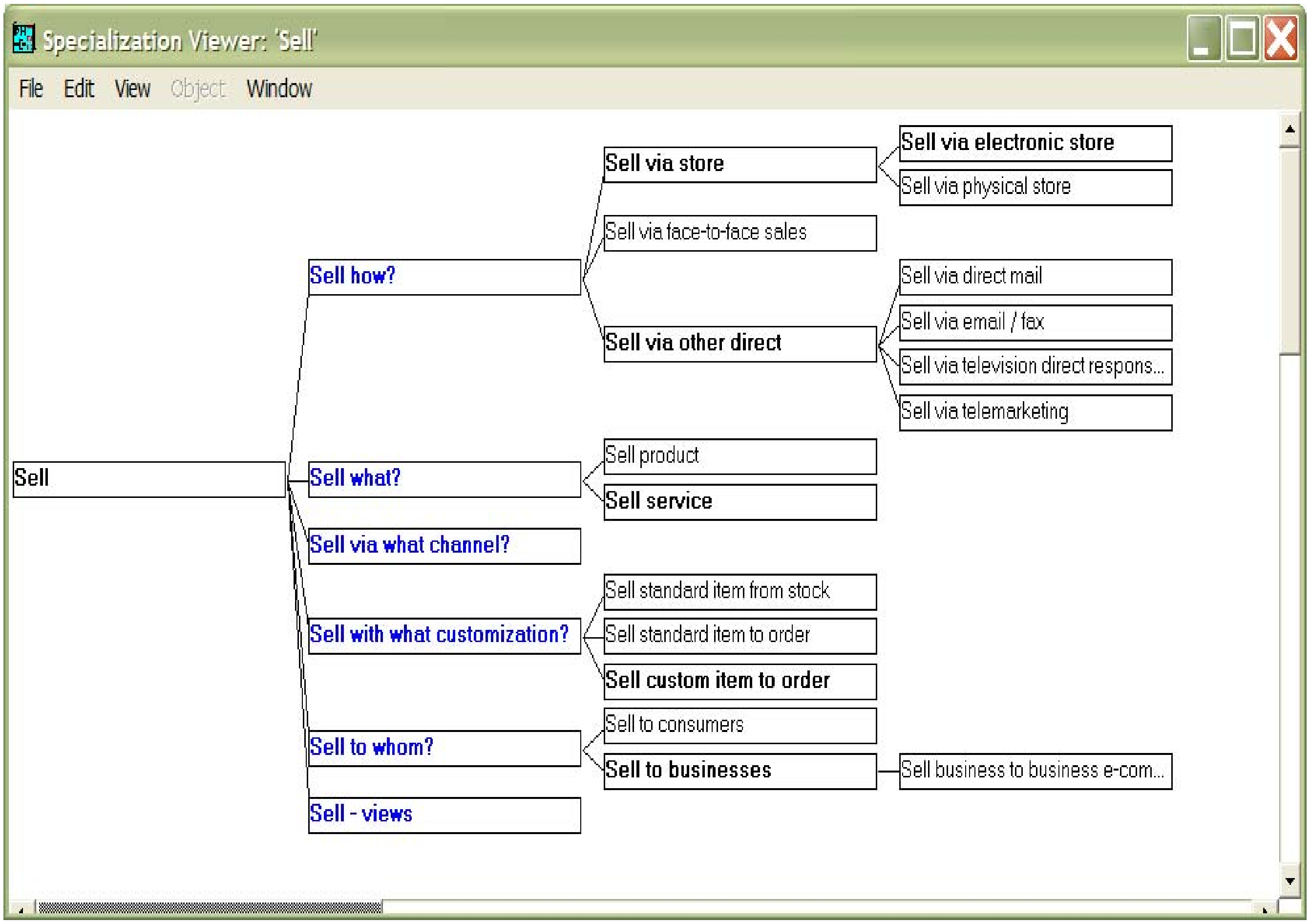

Figure 3: Some specializations of "Sell" in the MIT Process Handbook.

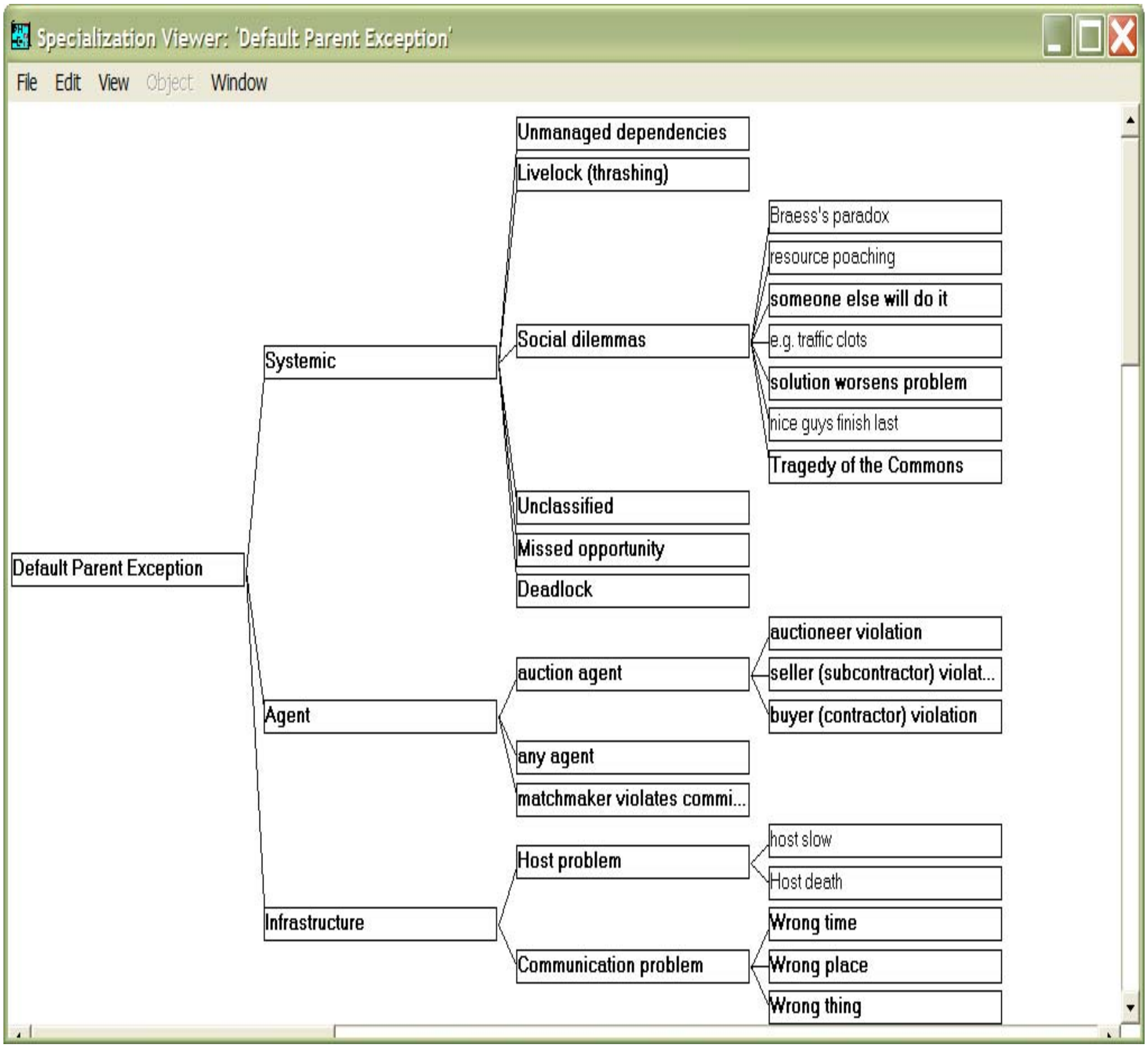

Figure 4: Some exceptions in the MIT Process Handbook.

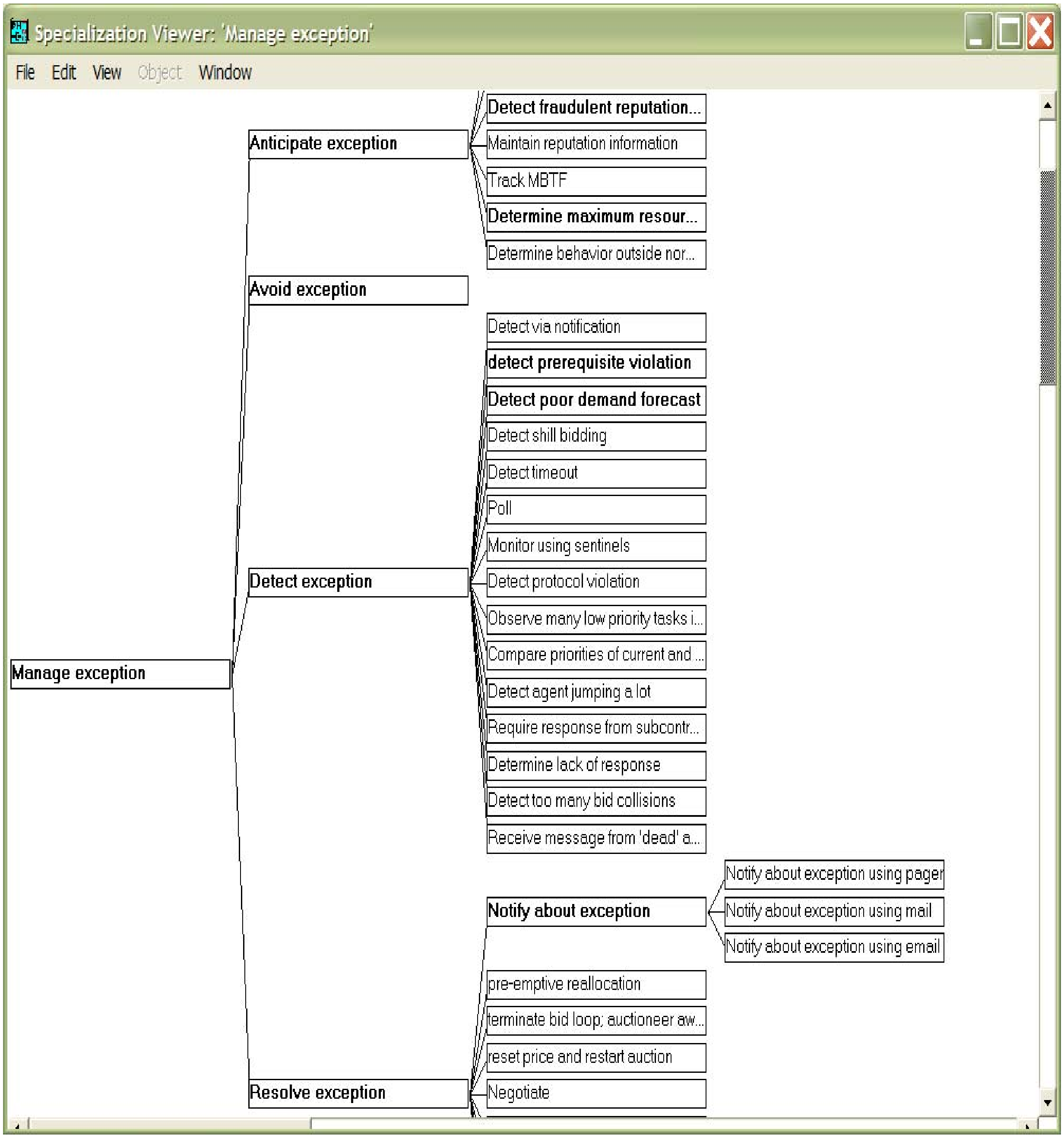

Figure 5: Some exception handlers in the MIT Process Handbook.[22]

[22] *Track MBTF* is a typo in the MIT Process Handbook. It should be *Track MTBF* (mean time between failures) instead.

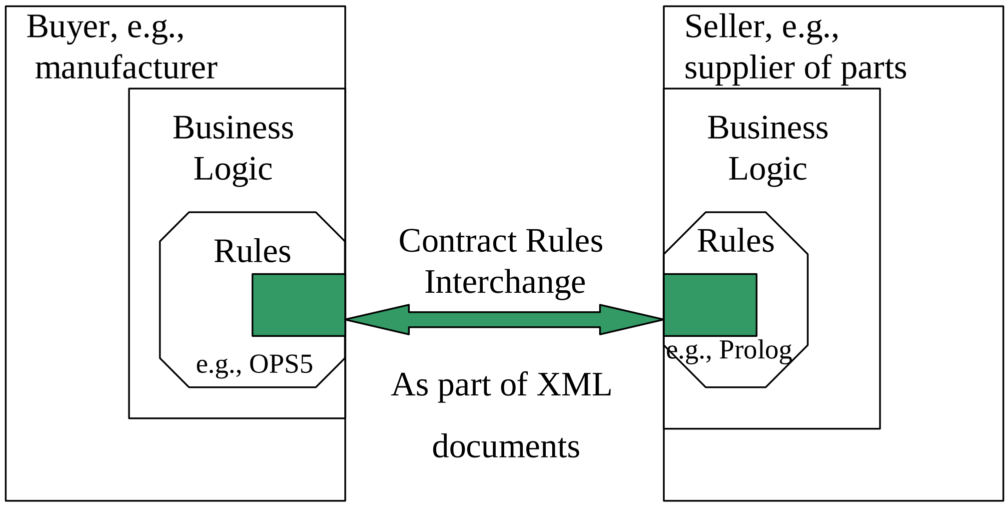

Figure 6: Contracting Parties NEGOTIATE Via Shared Rules.

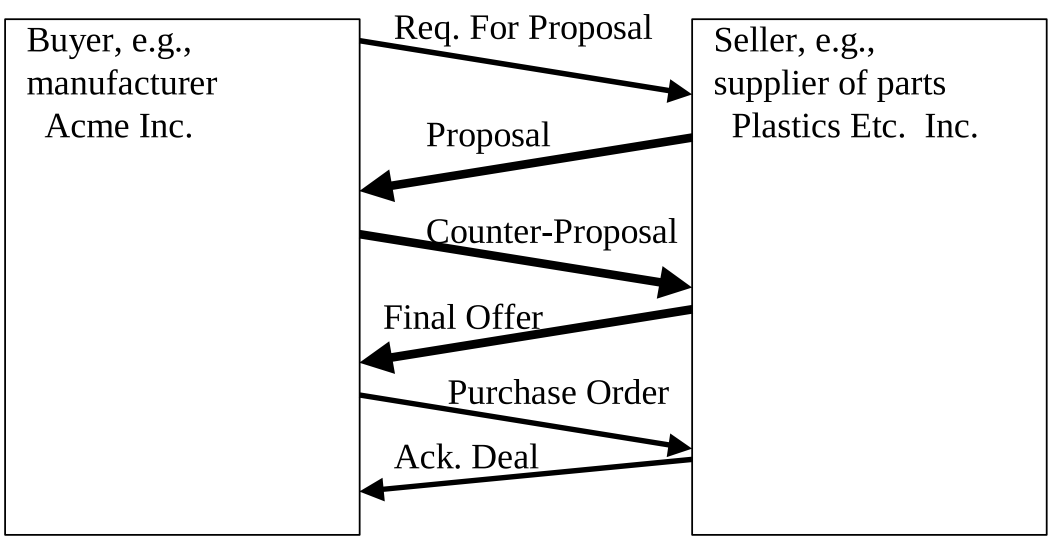

Figure 7: Exchange of Rules Content during Negotiation.