ISWC2005

International Semantic Web Conference
November 6-10 2005
Galway, Ireland

http://iswc2005.semanticweb.org

W9: The Semantic Web and Policy Workshop (SWPW)

Organisers: Lalana Kagal, Tim Finin, Jim Hendler

Monday, 7 November 2005

ISWC 2005 could not take place without the generous support of the following sponsors

Super Emerald Sponsors

Gold Sponsors

Silver Sponsors

ISWC 2005 Organising Committee

General Chair	Mark Musen, Stanford University
Research Track Co-Chair	Yolanda Gil, Information Sciences Institute
Research Track Co-Chair	Enrico Motta, The Open University
Industrial Track Chair	V Richard Benjamins, iSOCO, S.A.
Workshop Chair	Natasha F Noy, Stanford University
Tutorial Chair	R.V. Guha, Google
Poster & Demo Chair	Riichiro Mizoguchi, Osaka University
Semantic Web Challenge	Michel Klein, Vrjie Universiteit Amerdam
Semantic Web Challenge	Ubbo Visser, Universitat Bremen
Doctoral Symposium Co-Chair	Edward Curry, National University of Ireland, Galway
Doctoral Symposium Co-Chair	Enda Ridge, University of York
Meta-Data Chair	Eric Miller, W3C
Sponsorship Chair	Liam O'Móráin, DERI Galway
Local Organising Co-Chair	Christoph Bussler, DERI Galway
Local Organising Co-Chair	Stefan Decker, DERI Galway
Local Organiser	Brian Cummins, DERI Galway
Webmaster	Seaghan Moriarty, DERI Galway
Web Design	Johannes Breitfuss, DERI Innsbruck

Contents

Forward

The Semantic Web and Policy Workshop (SWPW) is a one day workshop held as part of the 4th International Semantic Web Conference 7 November 2005 in Galway, Ireland. This follows a similar workshop, Policy Management for the Web held at the 14th International World Wide Web Conference on 10 May 2005, in Chiba, Japan. SWPS is aimed at two different areas of research - (i) policy-based frameworks for the semantic web for security, privacy, trust, information filtering, accountability, etc.; and (ii) the applicability of semantic web technologies in policy frameworks for other application domains such as grid computing, networking, storage systems, and describing norms for multiagent systems. This workshop brings together not only researchers and developers but also users of policy systems in both these areas in an attempt to understand the scope of semantic web based policy frameworks and their usefulness.

From the submitted papers, fifteen were selected for inclusion in the printed proceedings and of these, nine for presentation and discussion during the workshop. The proceedings is available online from the SWPW web site at http://www.cs.umbc.edu/swpw/ and can be cited as

> Lalana Kagal, Tim Finin and James Hendler, Proceedings of the Semantic Web and Policy Workshop, 4th International Semantic Web Conference, 7 November, 2005, Galway, Ireland.

Ora Lassila of the Nokia Research Center in Burlington Massachusetts will give an invited talk entitled "Applying Semantic Web in Mobile and Ubiquitous Computing: Will Policy-Awareness Help?" and an accompanying paper is included in the proceedings.

At the end of the day a panel will be held to discuss the Web Policy Zeitgeist is as 2005 comes to a close. The panelists are Piero Bonatti, Wolfgang Nejdl (moderator), Karl Quinn, Norman Sadeh, and Kent Seamons. Each has been asked to respond to any or all of the following positions. Workshop participants are encouraged to think up new and provocative positions and to spring them on the panelists without warning and ask for a response.

- *Policies must be norms.* Policies must express norms for ideal behavior -- both positive and negative. Policies for real world applications will always be over constrained. What's interesting and challenging is how a poor agent plans around, trades off, and navigates through all of the conflicting constraints. Is it also important for policies to guide user in obtaining what they want?
- *Policies are not just about security or privacy.* Business rules are policies; Quality of service is regulated by policies. Policy specification languages should be able to express all of these shades of the notion of policy. What are the requirements for such a language?
- *Policies are not "islands".* They must interact with all sorts of software, data, and (why not?) knowledge. Decisions making needs different, specific kinds of information in each application. Adapting a policy framework to a specific application domain shall almost surely need some work. Our frameworks should minimize the effort.
- *Policies must be integrated with ontologies.* Or not. What is such integration expected to provide? Will this be useful only for the Semantic Web or are there advantages of using ontology based policies for other systems as well?
- *Policies must be X.* For some X. Articulate your own position and argue for it.

The SWPW organizers wish thank the excellent program committee for their hard work in reviewing the submitted papers and providing constructive feedback to the authors. We also thank the workshop authors for submitting good papers, responding to the reviewers' comments and keeping to our production schedule. Finally we thank the ISWC organizers for their help and especially Dr. Natalya F. Noy who, the ISWC 2005 workshop chair.

Lalana Kagal (chair), MIT CSAIL Laboratory
Tim Finin, UMBC Ebiquity Laboratory
Jim Hendler, UMD Mindswap Laboratory

Schedule

8.45 – 9.00 Welcome from the SWPW Chair

9.00 - 10.00 Invited talk

Applying Semantic Web in Mobile and Ubiquitous Computing: Will Policy-Awareness Help? Ora Lassila, Nokia Research Center

10.00 - 10.30 Information Filtering

The TriQL.P Browser: Filtering Information using Context- , Content- and Rating-Based Trust Policies, Christian Bizer, Richard Cyganiak, Tobias Gauss and Oliver Maresch

10.30 - 11.00 Coffee Break

11.00 - 12.30 Authorization Policies

The REWERSE View on Policies, Grigoris Antoniou, Matteo Baldoni, Cristina Baroglio, Piero A. Bonatti, Claudiu Duma, Norbert E. Fuchs, Alberto Martelli, Wolfgang Nejdl, Daniel Olmedilla, Viviana Patti, Joachim Peer and Nahid Shahmehri (22 mins)

Design and Application of Rule Based Access Control Policies, Huiying Li, Xiang Zhang, Honghan Wu and Yuzhong Qu (22 mins)

Rule-based and Ontology-based Policies: Toward a Hybrid Approach to Control Agents in Pervasive Environments, Alessandra Toninelli, Jeffrey Bradshaw, Lalana Kagal and Rebecca Montanari (22 mins)

Policy Based Dynamic Negotiation for Grid Services Authorization, Ionut Constandache, Wolfgang Nejdl and Daniel Olmedilla (22 mins)

12.30 - 14.00 Lunch

14.00 – 16.00 Policy and Trust Frameworks

A Logic Based SLA Management Framework, Adrian Paschke, Jens Dietrich and Karsten Kuhla

Real-world trust policies, Vinicius da Silva Almendra and Daniel Schwabe

Specification of Policies for Automatic Negotiations of Web Services, Steffen Lamparter and Sudhir Agarwal

Towards a Policy-Aware Web, Vladimir Kolovski, Yarden Katz, James Hendler, Daniel Weitzner and Tim Berners-Lee

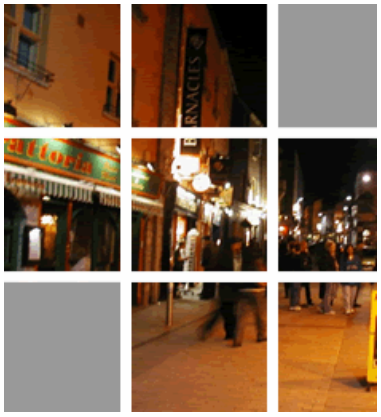
16.00 - 16.30 Coffee Break

16.30 - 17.30 Panel

2005 Web Policy Zeitgeist, Piero Bonatti, Wolfgang Nejdl, Karl Quinn Norman Sadeh and Kent Seamons

17.30 - 17.45 Wrap up

SWPW



Semantic Web and Policy Workshop

4th International Semantic Web Conference
7 November 2005, Galway, Ireland

This workshop is aimed at two different areas of research - (i) policy-based frameworks for the semantic web for security, privacy, trust, information filtering, accountability, etc. and (ii) the applicability of semantic web technologies in policy frameworks for other application domains such as grid computing, networking, storage systems, and describing norms for multiagent systems. This workshop will bring together not only researchers and developers but also users of policy systems in both these areas in an attempt to understand the scope of semantic web based policy frameworks and their usefulness.

- Policy specification, implementation, and enforcement
- Static and dynamic conflict resolution
- Dynamic policy modification
- Formal models for policy verification
- Applicability of XML, RDF and OWL for policy specification
- Using SW rule languages (e.g. RuleML, SWRL, N3) for policies
- Policies for access control, privacy, and accountability
- Obligation management
- Business contracts and rules
- Decidability and tractability issues
- Policy engineering and user-oriented policy authoring systems
- Case studies for policy management using semantic web technologies, including web site access, network routing, storage management, grid computing, pervasive computing, information filtering , digital rights management, collaboration

Format and venue. SWPW will be a one day workshop consisting of invited talks, presentations of submitted papers, a panel and ample time for discussion. The workshop will be held as part of ISWC 2005 in Galway, Ireland.

Submission details. We seek two kinds of papers: research papers that report on the results of original research and short papers that articulate a position, describe an application or demonstrate a working language or system. Both research papers and short papers will be included in the workshop proceedings. Research papers should describe original research not published elsewhere and should not exceed eight pages in length. Short papers are expected to be four to six pages. Short position papers should provide insight into the requirements for, or challenges of, developing or applying policies for web-based information systems. Short application papers should describe an implemented novel use of policies in a web-based environment. Short demonstration papers should document an implemented system or language that uses policies. Each submission should indicate the type of paper being submitted: research, position, application or demonstration.

Deadlines. Papers must be submitted electronically through the SWPW web site by 25 July 2005. Decisions will be announced on 5 September and final camera ready copy must be submitted by 30 September.

Chairs

Lalana Kagal, MIT
Jim Hendler, University of Maryland
Tim Finin, UMBC

Program Committee

Anne Anderson, Sun Microsystems
Vijay Atluri, Rutgers University
Jeremy Carroll, HP labs
Dan Connolly, W3C
Lorrie Cranor, CMU
Naranker Dulay, Imperial College
Tim Finin, UMBC
Benjamin Grosof, MIT
Jim Hendler, UMCP
Lalana Kagal, MIT
Jonathan Moffett, U. of York
Rebecca Montanari, U. of Bologna
Wolfgang Nejdl, L3S and U. Hannover
Daniel Olmedilla, L3S
Bijan Parsia, UMCP
Eric Prud'hommeaux, W3C
Norman Sadeh, CMU
Babak Sadighi, SICS
Stefan Poslad, Queen Mary, U. London
Kent Seamons, BYU
Anna Squicciarini, Purdue
William Winsborough, GMU
Mahesh Tripunitara, Purdue

http://cs.umbc.edu/swpw/

Applying Semantic Web in Mobile and Ubiquitous Computing: Will Policy-Awareness Help?

Ora Lassila

Nokia Research Center
5 Wayside Road
Burlington MA, USA

Abstract. The Semantic Web can be seen as a means of improving the interoperability between systems, applications, and information sources. Emerging personal computing paradigms such as mobile and ubiquitous computing will benefit from better interoperability, as this is an enabler for a higher degree of automation of many tasks that would otherwise require the end-users' attention. In this paper we present one possible view of mobile and ubiquitous computing enhanced with the application of Semantic Web technologies, and explore the various benefits of policy-awareness to this application domain.

1 Introduction

The emergence of *smartphones* – mobile phones capable of functions typically associated with personal digital assistants (PDAs) or even personal computers – has made *mobile information access* an everyday reality, and *mobile computing* the new (emerging) paradigm of personal computing and communications. With wide-area, local-area and *proximity* networking technologies now available on these mobile devices, we are rapidly transitioning towards Mark Weiser's vision of *ubiquitous computing* [1], where a single (personal) device is no longer the focal point of user's attention and where computation is effectively distributed to the environment surrounding the user.

Mobile and ubiquitous computing, while offering new opportunities, also pose several technological challenges not necessarily present in (or critical to) the current paradigm of personal computing based on the *desktop metaphor*. In the short term, progress is being made to better enable many typical personal computing tasks on these devices (for example, good progress has been made to enable normal "Web experience" on smartphones [2]), but ultimately the form-factor of handheld devices will force us to radically rethink user interfaces and user interaction, rather than merely suggesting a "miniaturization" of the prevalent graphical desktop interface. Even though we could eventually overcome the physical limitations inherent in mobile information retrieval, the real limitations have more to do with the usage situations of mobile devices: Information access often (if not predominantly) takes place in situations where the user is "attention-constrained"; in other words, the user is primarily paying attention to *something else* (say, driving a car) and cannot expend full attention to the process of finding and retrieving information (or to any other "personal computing" task for that matter).

In addition, *mobility* in itself introduces a more challenging environment with regard to connectivity, security, privacy, service discovery, etc., while at the same time making completely new applications possible that simply would not make sense in a more stationary form of computing. A device such

as a mobile phone, by virtue of being a constant companion to the user, can also be trusted with considerable amount of useful personal and private information; this, in turn, will force us to ensure that issues of information access (security, privacy) are properly dealt with.

We believe that mobile and ubiquitous computing devices would become more useful if they could undertake tasks *on behalf of* the user, rather than – as is currently the case – forcing the user to do essentially everything herself. The transition from mobile devices as *tools* to mobile *assistants* will require the application of not only technologies for implementing *autonomous operation*, but also sophisticated ontological techniques for representing information about the mobile devices, their functionality, users, environments, etc. [3, 4]

In general, this paper discusses the possible application of Semantic Web [5] technologies to mobile and ubiquitous computing. This application is motivated by the need for better *automation of user's tasks* (as a means of making the user's life easier); we will adopt the view that automation is best enabled by improving the *interoperability* between systems, applications, and information. In particular, we hope to demonstrate the need for *policy-awareness* as the ultimate enabler of the next generation personal information systems, and will offer the observation the not only are Semantic Web technologies are particularly well suited to rich, flexible representation of various policies, but that without the policy-awareness the application of Semantic Web technologies to mobile and ubiquitous computing may be hampered.

2 Ubiquitous Computing as an "Interoperability Nightmare"

Although much of ubiquitous computing research has focused on various aspects of user interaction [6], we can argue that a key characteristic of the paradigm – and one that makes ubiquitous computing distinctly different from the current personal computing paradigm(s) – is the *proliferation of devices that need to be connected*. Today's user connects his PC to a handful of other devices (printers, network gateways, etc.) and these connections are fairly static. Distinctly different from today's situation, ubiquitous computing scenarios are anticipated to involve dozens, if not hundreds of devices (sensors, external input and output devices, remotely controlled appliances, etc.). We therefore observe that ubiquitous computing, in its full-blown manifestations, represents the ultimate "interoperability nightmare". Furthermore, with the advent of mobility and associated proximity networking, the set of connected devices will constantly change as the usage context changes and as devices come into and leave the range of the user's ubiquitous computing device(s). Because of the dynamic nature of the new paradigm, technologies that improve interoperability will be crucial.

Given the need to dynamically connect to a large ever-changing set of devices and services, devices in a ubiquitous computing environment should be capable of sophisticated discovery and *device coalition formation*: the goal should be to accomplish discovery and configuration of new devices without "a human in the loop." In other words, the objective is the discovery and utilization of services offered by other automated systems without human guidance or intervention, thus enabling the automatic formation of device coalitions through this mechanism. Ultimately, one of the most important components of the realization of the Semantic Web (and also that of mobile and ubiquitous computing) is "serendipitous interoperability", the ability of software systems to discover and utilize services they have not seen before, and that were not considered when the systems

were designed [7]. To realize this, qualitatively stronger means of representing service semantics are required, enabling fully automated discovery and invocation, and complete removal of unnecessary interaction with human users.

Semantic Web techniques have proven useful in providing richer descriptions for Web resources, and consequently they can also applied to describing *functionality*. Semantic Web *Services*[1] appear to be an appropriate paradigm to be applied in representing the functionality of ubiquitous computing devices. Virtual and physical functions can be abstracted as services, providing a uniform view of all different kinds of functionality [3, 9]. Realization of this is contingent on the continuing emergence of suitable ontologies for modeling ubiquitous computing environments [10].

Avoiding *a priori* commitments about how devices are to interact with one another will improve interoperability and will thus make dynamic, unchoreographed ubiquitous computing scenarios more realistic. With reference to the aforementioned *serendipitous interoperability*, the true fulfillment of the vision for ubiquitous computing has a promise of serendipity in it that cannot be realized without discovery mechanisms that are *qualitatively stronger* than the current practice.

Semantic Web technologies represent a potential for this qualitatively stronger interoperability as compared to the traditional standards-based approach (where one essentially has to anticipate all future scenarios). With the Semantic Web approach it is possible for agents to "learn" new vocabularies and – via reasoning – make meaningful use of them. Furthermore, in addition to current notions of device and application interoperability, the Semantic Web represents interoperability at *the level of the information itself*.

3 Role of Context-Awareness in Ubiquitous Computing

As a primary means of addressing many of the issues in mobile and ubiquitous computing, *context-awareness* [11] offers a way to adapt a device's behavior to each usage situation, location, environment, user goal, etc. More generally, determining the user's *context* serves as a convenient means of limiting the search space along multiple dimensions of the system's operation:

Information retrieval: *Which information is interesting, relevant and applicable?* Given that her attention is focused elsewhere, the mobile user may merely "have questions" and will need very specific (and thus potentially terse) answers; using context information to limit the scope will make it easier to provide high-quality answers. Generally, having access to information in "raw" form (i.e., without any forethought as to how the information is to be presented or formatted), combined with the representation and reasoning capabilities enabled by Semantic Web technologies, will be helpful, because then *what* information gets presented (and *how* it gets formatted) can be a context-based decision.

User Interfaces: *Which user interface choices and configurations are appropriate?* Mobile devices suffer from various limitations such as small display size, awkward keyboard, etc. Taking *context* very broadly – covering just about everything that is known about the user, her task, the current environment, and the *device* she is using to access information – allows us to customize

[1] *Semantic Web Services* are generally defined as the augmentation of Web Service descriptions through semantic annotations, to facilitate a higher degree of automation of service discovery, composition, invocation and monitoring in an open, unregulated and often chaotic environment [8].

user interfaces; it may, in fact, be possible to go well beyond contemporary *content repurposing* approaches (such as [12]). We have been able to demonstrate that context information, combined with rich semantic models of the user, her environment and tasks, can be helpful in providing good default values and more generally tailoring the user interfaces to be ergonomically efficient for a particular user, task or purpose [13].

Service Discovery: *Which services to consider as relevant and/or applicable?* Service discovery in completely open-ended situations may suffer from the difficulty of having to pick services with little guidance and/or no limitations wrt. the search space; context here serves as a means of providing tighter boundaries for queries. Furthermore, characteristic to mobile and ubiquitous computing environments is that services are often tied to a particular location (or other notion of context); being able to choose or substitute appropriate functionality will enable flexible and robust operation [14].

Security & Privacy: *How to appropriately restrict access to information and services?* Access is sometimes dependent on the user's context. For example, a context may include *associative* aspects of the user's situation (e.g., a particular context may depend on who the other people are in the user's immediate vicinity), which, in turn, may have ramifications to privacy-related services and functionality.

Automation & Autonomy: *How to decide which operations to automate, and how?* Autonomous operation (which can be considered the extreme form of automation) is typically implemented using automated planning technologies (e.g., in agent-based systems); context can, again, serve as a means of limiting the search space when performing planning.

The process of determining context benefits from access to as many sources of information as possible, related to the user, her task, the environment, etc. [15]; we will subsequently argue that without a proper solution for security and privacy, efforts to implement context-awareness may be hampered.

4 Benefits of Policy-Awareness

In a general sense, a *policy* is a prescriptive means of limiting a system's behavior in some future situations. Typically policies are expressed using various *deontic modalities* [16] such as rights and obligations. With regard to information *access*, the use of an open, *policy-aware* infrastructure has been proposed for the World Wide Web [17]; we argue that this is a potentially applicable approach in mobile and ubiquitous computing as well. Semantic Web techniques, again, are well suited to describing, reasoning about, and exchanging *policies* [18, 19].

Policy-awareness may benefit mobile and ubiquitous computing in several ways:

Access: By providing access to maximal amount of information, policy-awareness enables higher-quality context derivation than would otherwise be possible. With the proper application of security and privacy policies it may be possible to base context derivation on data whose other uses would be prohibited.

Autonomy: The view of a policy as a prescriptive set of instructions about how a system should behave in some particular situation supports the use of policies as part of the realization to

increase personal computing systems' overall capability of autonomous behavior. Mobile and ubiquitous computing require intelligent response to changing conditions in a system's operating environment (e.g., changes in communication bandwidth); the use of policies may enable correct yet opportunistic exploitation of these changes without constant user intervention.

Contracts: Since devices in mobile and ubiquitous computing environments are always dependent on external services and functionality, mechanisms for *contracting* between systems will be necessary. As a general notion, contracting between agents has been and remains a standard technique in distributed problem solving [20]; more recently, the idea has been extended to allow contract representation using standard Semantic Web formalisms and to make it possible to reason about contracts [21, 22].

5 Conclusions

Semantic Web technologies offer several benefits to new computing paradigms such as mobile and ubiquitous computing. Not only do Semantic Web technologies lend themselves well to representation, reasoning and exchange of many different kinds of information (about functionality, contexts, policies, contracts, user models, etc.), but generally these technologies are a qualitatively stronger approach to *interoperability* than contemporary standards-based approaches. With sophisticated ontological representations we can realize effortless access to heterogeneous information sources and services, independent of the access device or the user's context; furthermore, we can finally untap the serendipitous potential that exists in unchoreographed encounters of automated and autonomous systems in cyberspace.

We do believe, however, that a pervasive framework for expressing and *enforcing* policies described using rich knowledge representation formalisms, is necessary for the full realization of this vision. For mobile and ubiquitous computing, the application of policy-awareness enables several key characteristics such as better access to information, higher degree of autonomy, and the ability to use and enforce contracts when employing external services and functionality.

References

1. Weiser, M.: The computer for the twenty-first century. Scientific American **265** (1991) 94–104
2. Nokia: Nokia develops a new browser for Series 60 by using open source software. Nokia press release, 2005-06-13 (2005)
3. Lassila, O., Adler, M.: Semantic Gadgets: Ubiquitous Computing Meets the Semantic Web. In Fensel, D., Hendler, J., Wahlster, W., Lieberman, H., eds.: Spinning the Semantic Web. MIT Press (2003) 363–376
4. McGrath, R.E., Ranganathan, A., Campbell, R.H., Mickunas, M.D.: Use of Ontologies in Pervasive Computing Environments. Technical report, University of Illinois at Urbana-Champaign (2003)
5. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American **284** (2001) 34–43
6. Abowd, G.D., Mynatt, E.D.: Charting past, present, and future research in ubiquitous computing. ACM Transactions on Computer-Human Interaction **7** (2000) 29–58
7. Lassila, O.: Serendipitous Interoperability. In Eero Hyvönen, ed.: The Semantic Web Kick-off in Finland – Vision, Technologies, Research, and Applications. HIIT Publications 2002-001. University of Helsinki (2002)

8. Payne, T., Lassila, O.: Semantic Web Services (guest editors' introduction). IEEE Intelligent Systems **19** (2004) 14–15
9. Masuoka, R., Parsia, B., Labrou, Y.: Task Computing – the Semantic Web meets Pervasive Computing. In Fensel, D., Sycara, K., Mylopoulos, J., eds.: The SemanticWeb - ISWC 2003. Volume 2870 of Lecture Notes in Computer Science., Springer-Verlag (2003)
10. Chen, H., Perich, F., Finin, T., Joshi, A.: SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In: International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous), Boston, MA (2004)
11. Dey, A., Abowd, G., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human-Computer Interaction **16** (2001) 97–166
12. Nokia: Nokia acquires Eizel to enhance mobile enterprise portfolio. Nokia press release, 2003-04-22 (2003)
13. Khushraj, D., Lassila, O.: Ontological Approach to Generating Personalized User Interfaces for Web Services. 4th International Semantic Web Conference, to appear (2005)
14. Lassila, O., Dixit, S.: Interleaving Discovery and Composition for Simple Workflows. In: Semantic Web Services, AAAI Spring Symposium Series, AAAI (2004)
15. Lassila, O., Khushraj, D.: Contextualizing Applications via Semantic Middleware. In: The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous), IEEE Computer Society (2005)
16. Mally, E.: Grundgesetze des Sollens: Elemente der Logik des Willens. Graz: Leuschner und Lubensky, Universitäts-Buchhandlung (1926)
17. Weitzner, D.J., Hendler, J., Berners-Lee, T., Connolly, D.: Creating a Policy-Aware Web: Dicretionary, Rule-based Access for the World Wide Web. In Ferrari, E., Thuraisingham, B., eds.: Web and Information Security. Idea Group, Inc., Hershey, PA (2005)
18. Kagal, L., Parker, J., Chen, H., Joshi, A., Finin, T.: Security and Privacy Aspects. In: Security, Privacy and Trust in Mobile Computing Environments. CRC Press (2003)
19. Kagal, L.: A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments. PhD thesis, University of Maryland Baltimore County, Baltimore MD (2004)
20. Davis, R., Smith, R.: Negotiation as a metaphor for distributed problem solving. Artificial Intelligence **20** (1983) 63–109
21. Grosof, B.N., Poon, T.C.: SweetDeal: representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In: WWW '03: Proceedings of the 12th international conference on World Wide Web, New York, NY, USA, ACM Press (2003) 340–349
22. Uszok, A., Bradshaw, J.M., Jeffers, R., Johnson, M., Tate, A., Dalton, J., Aitken, S.: Policy and Contract Management for Semantic Web Services. In: Semantic Web Services, AAAI Spring Symposium Series, AAAI (2004)

Acknowledgements

The author's own research cited in this paper was supported in part by the Nokia Technology Platforms and the Nokia Research Center. The author would also like to thank the anonymous reviewers of the workshop who provided many constructive comments.

The TriQL.P Browser: Filtering Information using Context-, Content- and Rating-Based Trust Policies

Christian Bizer[1], Richard Cyganiak[1], Tobias Gauss[1], and Oliver Maresch[2]

[1] Freie Universität Berlin, Germany
chris@bizer.de
richard@cyganiak.de
tobias.gauss@web.de
[2] Technische Universität Berlin, Germany
Oliver-Maresch@gmx.de

Abstract. The TriQL.P browser is a general purpose RDF browser that supports users in exploring RDF datasets containing information from multiple information sources. Information can be filtered using a wide range of user-definable trust policies. Policies can be based on information context, information content, information or information source ratings, and on the presence or absence of digital signatures. In order to help users understand the filtering decisions, the browser can explain why a piece of information fulfils the selected trust policy.

1 Trust Policies for the Semantic Web

The Semantic Web is an open, dynamic network of independent information providers all having different views of the world, different levels of knowledge, and different intentions. Thus, information found on the Semantic Web has to be seen as claims rather than as facts. Before using these claims, the information consumer has to evaluate their trustworthiness and determine the subset which he wants to trust for his specific task.

In everyday life, we use a wide range of trust assessment policies for evaluating the trustworthiness of information: We might trust Andy on restaurants but not on computers, trust professors on their research field, believe foreign news only when it is reported by several independent sources and buy only from sellers on eBay who have more than 100 positive ratings.

Which policy is chosen depends on the specific task, our subjective preferences, our past experiences and the trust relevant information available. For tasks which are economically relevant to the information consumer he might require a very strict trust policy, involving for example recommendations by people he knows. For other tasks, a looser policy like 'Accept all information that has been asserted by at least two independent information providers, no matter who they are.' might be acceptable.

The future Semantic Web is supposed to be a dense mesh of interrelated information, similar to the information perception situation we face in the offline

world. Thus, we argue, a trust policy framework for the Semantic Web can and should support a similarly wide range of trust policies as used offline [4].

Every trust policy employs one or more trust assessment methods. These methods can be classified into three categories:

Rating-Based Methods include rating systems like the one used by eBay and Web-Of-Trust mechanisms. Most trust architectures proposed for the Semantic Web so far fall into this category [1][11]. The general problem with these approaches is that they require explicit and topic-specific trust ratings. For many application domains, providing such ratings and keeping them up-to-date puts an unrealistically heavy burden on information consumers.

Context-Based Methods use meta-data about the circumstances in which information has been claimed, e.g. who said what, when and why. They include role-based trust methods, using the author's role or his membership in a specific group, for trust decisions. Example policies from this category are: 'Prefer product descriptions published by the manufacturer over descriptions published by a vendor' or 'Distrust everything a vendor says about its competitor.' Context-based trust mechanisms do not require explicit ratings, but rely on the availability of background information. Within many Semantic Web application areas, such background information might be available.

Content-Based Methods do not use meta-data about information, but rules and axioms together with the information content itself and related information about the same topic published by other authors. Example policies following this approach are: 'Believe information which has been stated by at least 5 independent sources.' or 'Distrust product prices that are more than 50% below the average price.'

2 The TriQL.P Browser

The TriQL.P browser is a general purpose RDF browser which shows how Semantic Web content can be filtered using a wide range of trust policies, combining methods from all three categories described above.

The TriQL.P browser is based on the Piggy Bank extension for the Firefox browser [10]. Piggy Bank extracts Semantic Web content from Web pages as users browse the Web. On websites where Semantic Web content is not available, Piggy Bank can invoke screen-scrapers to re-structure content into Semantic Web format. The extracted information can be browsed, sorted and searched using a comfortable user-interface, and saved into a local repository for future reference and aggregation.

In addition to the functionality provided by Piggy Bank, the TriQL.P browser gives users the ability to:

- collect provenance meta-data together with information from the Web;
- import information aggregated from multiple sources by a third party into the local repository using the RDF/XML, TriX [7] and TriG [3] syntaxes;
- load trust policy suites containing sets of policies;

– filter information in the local repository using these policies;
– explain on demand why displayed information fulfils a selected policy.

Figure 1 shows the user interface of the TriQL.P browser. Information items from the local repository are displayed on the left-hand side. The policy selection box on the right side allows users to select a policy from the policy suite currently loaded. After selecting a policy, the left-hand view updates to show only information matching this policy. There is a 'Oh, yeah?'-button [2] next to each piece of information. Pressing this buttons opens a new window with an explanation why the piece of information fulfils the selected trust policy.

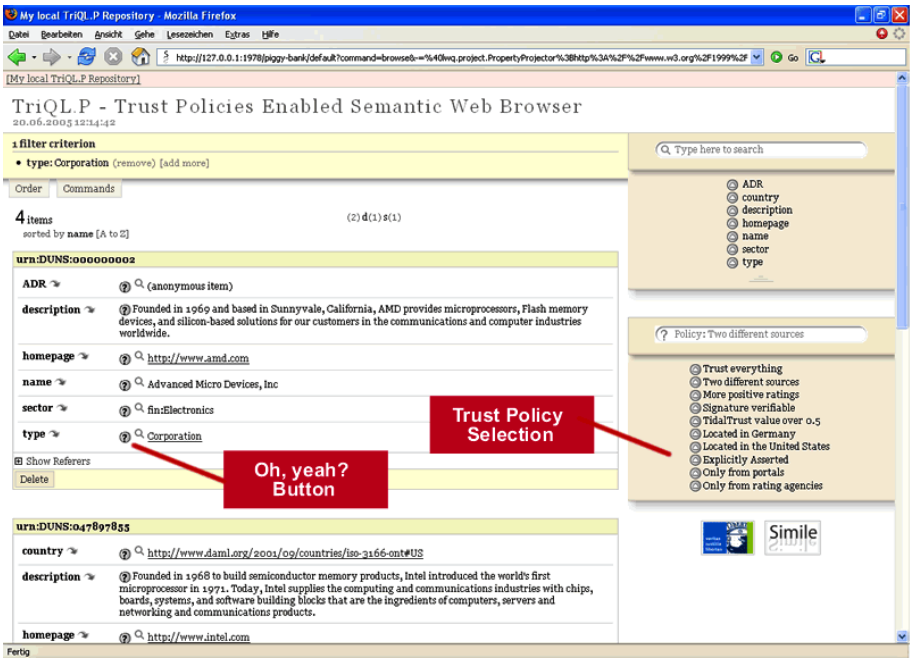


Fig. 1. The TriQL.P user interface. The user selects a trust policy from the right-hand box. The left-hand view updates to show only matching information. The 'Oh, yeah?' buttons open new windows with explanations why a piece of information fulfils the selected policy.

Figure 2 shows an explanation why information information about Peter Smith's email address fulfils the policy 'Trust only information that has been asserted by at least two different sources.'

Figure 3 shows an explanation why a news article fulfils the policy 'Trust only information from information providers who have a Tidal Trust score above 0.5'. The Tidal Trust metric calculates trust scores by determining shortest paths between individuals in a social network of weighted trust statements and calcu-

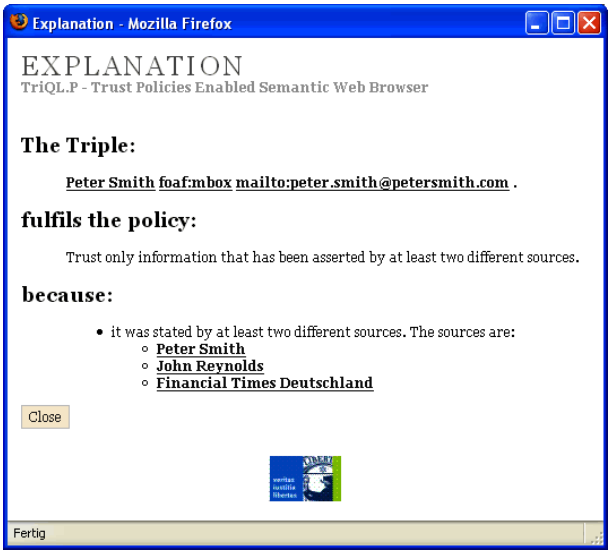


Fig. 2. The explanation window. This explanation establishes why information about Peter Smith's email address fulfils the policy 'Asserted by at least two different sources'.

lating its weighted average [11]. The explanation generated for this trust metric shows the calculation result and contains details about the calculation steps and the information sources used, allowing an information consumer to comprehend the calculations at different levels of detail.

The TriQL.P browser is pretty flexible in rendering explanations for different policies. An explanation for the policy 'Trust only information providers which are working for at least two projects about a specific topic' would contain the list of projects for each information provider. An explanation for the policy 'Trust only information that has been signed by the information providers' would contain details about the signature verification process[3].

The following sections explain how information collected from different sources is represented within the TriQL.P browser, how trust policies are expressed and applied, and how explanations are generated.

3 Representing Information

The TriQL.P browser uses Named Graphs [5] as internal data model. Named Graphs are a slight extension of the RDF abstract syntax and provide well-defined semantics for the attachment of provenance information and other meta-data to RDF graphs.

[3] Various policies and corresponding explanations are found at http://www.wiwiss.fu-berlin.de/suhl/bizer/TriQLP/browser/

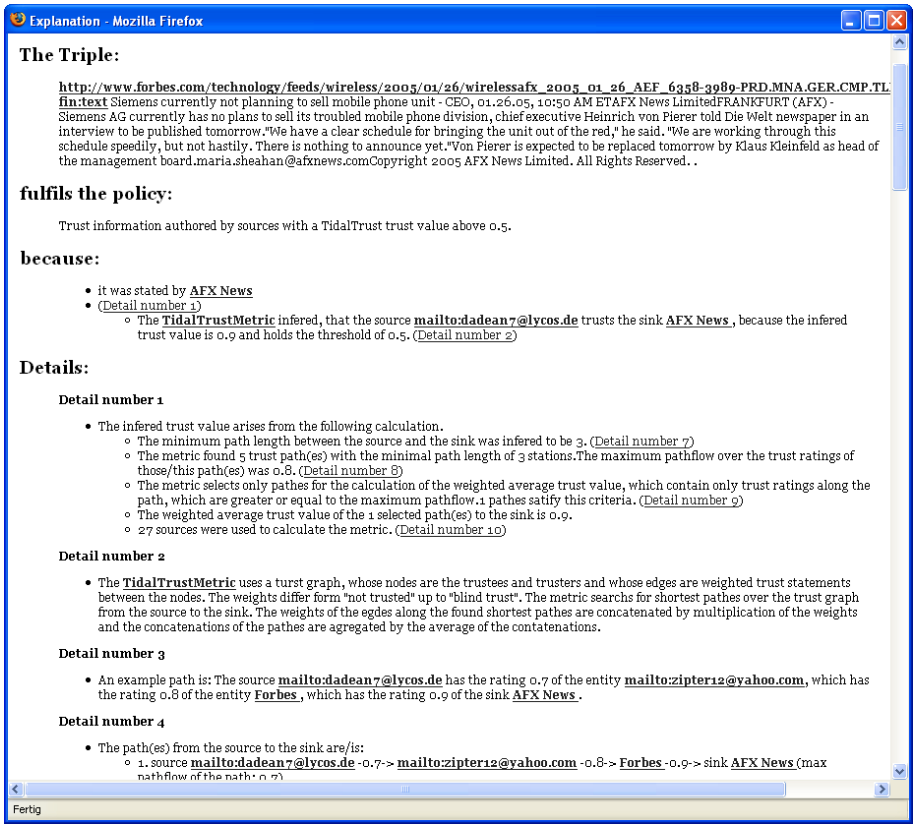


Fig. 3. Explanation for the policy 'Trust only information from information providers who have a Tidal Trust score above 0.5'

Such provenance information can be expressed with the Semantic Web Publishing Vocabulary (SWP) [6]. SWP also provides terms to indicate whether a graph is asserted or quoted and to attach digital signatures to it.

Whenever the browser saves information from a webpage into the local repository, it creates a new named graph for this visit of the page and stores the current timestamp, the URL of the page and the authority (website URL) together with the actual information. The following example shows the browser's internal representation of information about Peter Smith collected from the URL `http://www.bizer.de/myFriends.htm`, together with the recorded provenance information. The example uses the TriG syntax [3].

```
<urn:uuid:8c845860-dce7-11d9-b9c0-00112ff60c7f> {
    ex:PeterSmith a foaf:Person ;
        foaf:name "Peter Smith" ;
        foaf:mbox <mailto:peter.smith@petersmith.com> .
```

```
<urn:uuid:8c845860-dce7-11d9-b9c0-00112ff60c7f>
    swp:assertedBy
       <urn:uuid:8c845860-dce7-11d9-b9c0-00112ff60c7f> ;
    swp:authority <http://www.bizer.de> ;
    dc:date "2005-06-14T17:18:10+02:00" ;
    swp:savedFrom <http://www.bizer.de/myFriends.htm> . }
```

4 Expressing and Applying Policies

The TriQL.P browser displays information from a virtual trusted graph which contains a subset of the triples from all named graphs stored in the local repository. The browser's trust evaluation layer uses trust policies to determine which triples from the untrusted named graphs are promoted into the virtual trusted graph. The decision is made on a triple-by-triple basis, although many policies are written to accept or reject entire graphs.

The heart of every trust policy is a TriQL.P query. TriQL.P is a query language similar to but predating SPARQL [14]. In addition to basic graph pattern matching, TriQL.P offers two language constructs which are especially useful for expressing trust policies: COUNT() for formulating quantity conditions and METRIC() as an open interface to different rating metrics.

An example TriQL.P query is shown below. When executed against the untrusted repository, it selects all triples (bound to the variables `?SUBJ`, `?PRED` and `?OBJ`) which are asserted by at least two authorities.

```
SELECT ?SUBJ, ?PRED, ?OBJ
WHERE ?GRAPH (?SUBJ, ?PRED, ?OBJ)
              (?GRAPH swp:assertedBy ?warrant .
               ?warrant swp:authority ?authority)
AND COUNT(?authority) >= 2
```

In order to associate explanation templates with individual graph patterns and to provide additional metadata about a policy, TriQL.P queries are divided into single graph patterns and constraints and are recombined using the TPL - Trust Policy Language. The example below shows a TPL policy built from the query above:

```
:Policy6 rdf:type tpl:TrustPolicy ;
     tpl:policyName "Asserted by at least two sources" ;
     tpl:policyDescription "Trust only information that has
              been asserted by at least two different sources." ;
     tpl:textExplanation "it was stated by at least two
              different sources. The sources are:" ;
     tpl:graphPattern [
         tpl:pattern "(?GRAPH swp:assertedBy ?warrant .
                       ?warrant swp:authority ?authority)";
```

```
        tpl:textExplanation "@@?authority@@" ; ] ;
    tpl:constraint "COUNT(?authority) >= 2" .
```

The display layer retrieves information from the trust evaluation layer using *find* queries (queries for all triples matching a triple pattern where subject, predicate and object may be wildcards) against the virtual trusted graph.

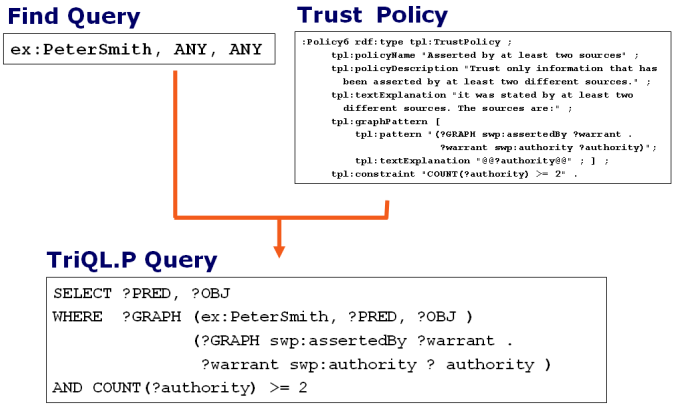


Fig. 4. A find query asking for all information about ex:PeterSmith is combined with a trust policy resulting into a complete TriQL.P query.

To display all information about the resource `ex:PeterSmith`, the following steps are performed:

1. The display layer sends a *find* query and a policy URI to the trust evaluation layer.
2. The engine combines the *find* query with the graph patterns and constraints contained within the policy into a complete TriQL.P query. The variable `?SUBJ` is pre-bound to the value `ex:PeterSmith`.
3. The TriQL.P query is executed against the untrusted repository.
4. RDF triples are created from the variables `?SUBJ`, `?PRED` and `?OBJ` of every query solution, and sent back to the display layer.

The trust evaluation layer caches the values bound to all other query variables, like `?warrant` and `?authority` in the example. They may be used later to generate explanations.

TriQL.P offers an open interface for rating metrics that cannot be expressed as graph patterns. The following example shows how the TidalTrust metric is used as a constraint within a policy:

```
tpl:constraint "METRIC(tpl:TidalTrustMetric, ?USER, ?author, 0.5)".
```

Metrics are implemented as Java plug-ins into the TriQL.P query engine. There are currently four metric plug-ins available: eBay, TidalTrust [11], Appleseed [15] and PageRank [13]. The first three metrics require explicit ratings. The PageRank metric avoids the nessesarity of explicit ratings but allows common RDF predicates like foaf:knows or rdf:seeAlso to be used as links for ranking.

5 Explaining Filtering Decisions

Each 'Oh, yeah?' button in the browser's user interface corresponds to one RDF triple of the virtual trusted graph. The following steps are executed to generate an explanation why a triple fulfils a selected policy:

1. The engine retrieves the previously cached set of variable bindings that was used to produce the triple.
2. The explanation templates associated with the trust policy are instantiated using the variable bindings.
3. The resulting text snippets are grouped into a tree which is rendered into HTML.

In addition to this basic explanation mechanism, `METRIC()` plug- ins generate their own explanations about their calculation process and information used within the process.

6 Conclusions

We have argued that trust frameworks for the Semantic Web should not rely solely on explicit ratings but also facilitate information context and information content for trust assessments. We have shown a flexible way to express trust policies and to explain filtering decisions based on these policies, and we have described how to integrate our policy framework into a general-purpose Semantic Web browser.

Our approach of expressing policies as query templates instead of expressing them as rules [8] might suit users familiar with query languages like SPARQL. The information provenance explanations generated by the browser are similar to the provenance traces used within TRELLIS [9]. Compared with TRELLIS, our explanations are more flexible as they don't assume a single provenance ontology. The explanations generated by Inference Web [12] are complementary to our work. Inference Web focuses on explaining distributed reasoning paths, while we are focusing on explaining information provenance, background knowledge used in the assessments and metric calculations.

We hope that our prototype facilitates further thinking about pragmatic ways to incorporate trust policy frameworks into Semantic Web applications, as trust is an essential topic for the Semantic Web but is often ignored by current applications.

The TriQL.P browser is available under BSD license. More information about the browser, example RDF datasets and example policy suites are found at: `http://www.wiwiss.fu-berlin.de/suhl/bizer/TriQLP/browser/`

References

1. R. Agrawal, P. Domingos, and M. Richardson. Trust Management for the Semantic Web. In *2nd International Semantic Web Conference*, 2003.
2. T. Berners-Lee. Cleaning up the user interface, section - the "oh, yeah?"-button, 1997. http://www.w3.org/DesignIssues/UI.html.
3. C. Bizer. The trig syntax. http://www.wiwiss.fu-berlin.de/suhl/bizer/TriG/.
4. C. Bizer and R. Oldakowski. Using context- and content-based trust policies on the semantic web. In *13th World Wide Web Conference (Poster)*, 2004.
5. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs website. http://www.w3.org/2004/03/trix/.
6. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *14th International World Wide Web Conference*, 2005.
7. J. Carroll and P. Stickler. TriX: RDF Triples in XML. In *Proceedings of Extreme Markup Languages*, 2004.
8. T. Gianluca. Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder. In *2nd International Semantic Web Conference*, 2003.
9. Y. Gil and V. Ratnakar. Trusting information sources one citizen at a time. In *1st International Semantic Web Concerence*, 2002.
10. D. Huynh, S. Mazzocchi, and D. Karger. Piggy bank: Experience the semantic web inside your web browser. Submitted to the International Semantic Web Conference 2005.
11. Jennifer Golbeck. *Computing and Applying Trust in Web-based Social Networks*. PhD thesis, 2005. http://trust.mindswap.org/papers/GolbeckDissertation.pdf.
12. D. L. McGuinness and P. P. da Silva. Infrastructure for web explanations. In *2nd International Semantic Web Conference*, 2003.
13. L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.
14. E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217/, 2005.
15. C.-N. Ziegler and G. Lausen. Spreading activation models for trust propagation. In *IEEE International Conference on e-Technology, e-Commerce, and e-Service*, 2004.

The REWERSE View on Policies

P.A. Bonatti, G. Antoniou, M. Baldoni, C. Baroglio, C. Duma, N. Fuchs, A. Martelli, W. Nejdl, D. Olmedilla, J. Peer, V. Patti, and N. Shamheri

REWERSE (REasoning on the WEb with Rules and SEmantics)
WG-I2: *Policy specification, composition, and conformance*

Abstract. In this position paper we outline the vision adopted by the working group on policies of the EU FP6 Network of Excellence REWERSE, IST-2004-506779.

Keywords: Integrated heterogeneous policies, Cooperative policy enforcement, Lightweight trust, Trust management, Natural language interfaces, Explanation mechanisms.

1 Introduction

REWERSE (REasoning on the WEb with Rules and SEmantics) is one of the two european networks of excellence on the semantic web funded by the European Union within the 6th framework program.[1] The focus of REWERSE is on lightweight knowledge representation and reasoning, based as much as possible on rule-based languages because of their low computational complexity.

One of REWERSE's working groups, WG I2, is expressly devoted to *policy specification, composition, and conformance*. The members of WG I2 have identified in policies one of the most interesting areas for applying semantic web ideas.

Policies are pervasive in web applications. They play crucial roles in enhancing security, privacy, but also service usability. They may determine the success of a web service (or its failure). A user will not be able to benefit of the protection mechanisms of its system until she understands and is able to personalize the policies applied by the system. Simlarly, the facilities of a web service will not be fully available to its customers unless they understand the policies applied by the system (access control policies, privacy policies, and business rules, at least).

The vision of WG I2 can be summarized by the following list of strategic goals and lines of research:

– We adopt a *broad notion of policy*, encompassing not only access control policies, but also privacy policies, business rules, quality of service, etc. We believe that all these different kinds of policies should eventually be integrated into a single framework.

[1] REWERSE has 27 academic and industrial participants distributed across 14 european countries. The network officially started on March 1, 2004. More information on `http://www.rewerse.net`.

- *Strong and lightweight evidence*: Policies make decisions based on properties of the peers interacting with the system. These properties may be strongly certified by—say—cryptographic techniques, or be reliable to an intermediate degree, where evidence gathering and validation are easier (lightweight evidence). A flexible policy framework shall merge the two forms of evidence to meet the efficiency and usability requirements of web applications.
- The above two points imply that trust negotiation, reputation models, business rules, and action specification languages should be integrated into a single framework, to some extent. It is crucial to find the right tradeoff between generality and efficiency. *So far, no framework has tried to merge all these aspects together into a coherent system.* This is one of the hard challenges of WG I2.
- *Automated trust negotiation* (ATN)—adapted to other forms of negotiation—is one of the main ingredients that we use to make heterogeneous peers effectively interoperate. Therefore we are actively contributing to the advances in the area of *trust management*.
- By *lightweight knowledge representation and reasoning* we do not only refer to computational complexity; we mean also reducing the effort to specialize our general frameworks to specific application domains; and we mean that our tools should be easy to learn and use for common users, with no particular training in computers or logic. We regard these properties as crucial for the success of a semantic web framework.
- The last issue cannot be tackled simply by adopting a rule language. We are working at a *controlled natural language syntax for policy rules*, to be translated by a parser into the internal logical format.
- *Cooperative policy enforcement*: A secure cooperative system should (almost) never say *no*. Web applications need to help new users in obtaining the services that the application provides—potential customers should not be discouraged. When the prerequisites for accessing a service are not met, the web application should better explain what is missing and help the user in obtaining the required permissions.
- As part of cooperative enforcement, advanced *explanation mechanisms* should be developed to help users in understanding policy decisions and obtaining the permission to access the desired service.

In the rest of this paper we expand on the above issues and point out what we regard as interesting research directions.

2 A broad notion of policy

Policies are pervasive in all web-related contexts. Access control policies are needed to protect any system open to the internet. Privacy policies are needed to assist users while they are browsing the web and interacting with web services. Business rules (that in the view of WG I2 are just another kind of policy) specify which conditions apply to each customer of a web service. Other policies specify

constraints related to Quality of Service (QoS). In E-government applications, visas and other documents are released according to specific eligibility policies. Of course this list is not exhaustive, and is limited only by the class of applications that can be deployed in the world wide web.

Note that most of these policies make their decisions based on similar pieces of information—essentially, properties of the peers involved in the transaction. For example, age, nationality, customer profile, identity, and reputation may all be considered both in access control decisions, and in determining which discounts are applicable (as well as other eligibility criteria). Then it is appealing to integrate these kinds of policies into a coherent framework, so that (i) a common infrastructure can be used to support interoperability and decision making, and (ii) the policies themselves can be harmonized and synchronized.

In the general perspective depicted above, policies may also establish that some events must be logged (audit policies), that user profiles must be updated, and that when a transaction fails, the user should be told how to obtain missing permissions. In other words, policies may specify *actions* whose execution may be interleaved with the decision process. Such policies are called *provisional policies*.

Then, in our view, *policies act both as decision support systems and as declarative behavior specifications*. An effectively user-friendly approach to policy specification would give common users (with no training in computer science or logic) a better control on the behavior of their own system (see the discussion in Section 5).

Of course, the extent to which this goal can be actually achieved depends on the policy's ability of *interoperating* with legacy software and data—or more generally, with the rest of the system. Then a policy specification language should support suitable primitives for interacting with external packages and data in a flexible way.

The main challenges raised by the above discussion are the following:

- Harmonizing security and privacy policies with business rules, provisional policies, and other kinds of policy is difficult because their standard formalizations are based on different derivation strategies, and even different reasoning mechanisms, sometimes (cf. Section 4.3). Deduction, abduction, and event-condition-action rule semantics need to be integrated into a coherent framework, trying to minimize subtleties and technical intricacies (otherwise the framework would not be widely accessible to common users).
- The interactions between a rule-based theory and "external" software and data has been extensively investigated in the framework of logic-based mediation and logic-based agent programming [11, 10]. However, there are novel issues related to implementing high-level policy rules with low-level mechanisms such as firewalls, web server and DBMS security mechanisms, operating system features etc., that are typically faster and more difficult to bypass than rule interpreters [8]. A convincing realization of this approach might boost the application of the rich and flexible languages developed by the security community.

3 Strong and lightweight evidence

There exist currently two different major approaches for managing trust: policy-based and reputation-based trust management. The two approaches have been developed within the context of different environments and targeting different requirements. On the one hand, policy-based trust relies on objective "strong security" mechanisms such as signed certificates and trusted certification authorities (CA hereafter) in order to regulate the access of users to services. Moreover, the access decision is usually based on mechanisms with well defined semantics (e.g., logic programming) providing strong verification and analysis support. The result of such a policy-based trust management approach usually consists of a binary decision according to which the requester is trusted or not, and thus the service (or resource) is allowed or denied. On the other hand, reputation-based trust relies on a "soft computational" approach to the problem of trust. In this case, trust is typically computed from local experiences together with the feedback given by other entities in the network. For instance, in eBay buyers and sellers rate each other after each transaction. The ratings pertaining to a certain seller (or buyer) are aggregated by the eBay's reputation system into a number reflecting seller (or buyer) trustworthiness as seen by the eBay community. The reputation-based approach has been favored for environments, such as Peer-to-Peer or Semantic Web, where the existence of certifying authorities could not be always assumed but where a large pool of individual user ratings is often available.

Yet another approach—very common in today's applications—is based on forcing users to commit to contracts or copyrights by having users click an "accept" button on a pop-up window. This is perhaps the lightest approach to trust, that can be generalized by having users utter *declarations* (on their e-mail address, on their preferences, etc.) e.g. by filling an HTML form.

Real life scenarios often require to make decisions based on a combination of the above approaches. Transaction policies must handle expenses of all magnitudes, from micropayments (e.g. a few cents for a song downloaded to your iPod) to credit card payments of a thousand euros (e.g. for a plane ticket) or even more. The cost of the traded goods or services typically contributes to determining the risk associated to the transaction and hence the trust needed for performing it.

Strong evidence is generally harder to gather and verify than lightweight evidence. Sometimes, a "soft" reputation measure or a declaration (in the sense outlined above) is all one can obtain in a given scenario. We strongly believe that the success of a trust management framework can be determined by the ability of *balancing trust levels and risk levels* for each particular task supported by the application. So we add the following items to the list of interesting research directions:

– How should the different forms of trust be integrated? A first proposal can be found in these proceedings (see the paper by Bonatti, Duma, Nejdl, Olmedilla, and Shahmehri). However, new reputation models keep on being introduced, and there is a large number of open research issues in the

reputation area (e.g., vulnerability to coalitions). Today, it is not clear which of the current approaches will be successful and how the open problems will be solved (this is why our current proposal aims at maximal modularity in the integration of numerical and logical trust).
– How many different forms of evidence can be conceived? In principle, properties of (and statements about) an individual can be extracted from any—possibly unstructured—web resource. Supporting such a variety of information in policy decisions is a typical semantic web issue—and an intriguing one. However, such general policies are not even vaguely as close to become real as the policies based on more "traditional" forms of evidence (see the discussion in the next section).

4 Trust management

4.1 Some history

During the past few years, some of the most innovative ideas on security policies arose in the area of *automated trust negotiation* [1, 2, 5, 6, 12–15]. That branch of research envisaged peers that automatically negotiate credentials according to their own declarative, rule-based policies. Rules specify for each resource or credential request which properties should be satisfied by the subjects and objects involved. Then, at each negotiation step, the next credential request is formulated essentially by *reasoning* with the policy, e.g. by inferring implications or computing abductions.

Since year 2000 there exist frameworks where credential requests are formulated by exchanging *sets of rules* [2, 5]. Requests were formulated *intensionally* in order to express compactly and simultaneously all the possible ways in which a resource can be accessed—thereby shortening negotiations and improving privacy protection (because peers can choose the best option from the point of view of sensitivity). Intuitively, it is not appealing to request "*an ID and a credit card*" by enumerating all possible pairs of ID credentials and credit card credentials; it seems much better to *define* what IDs and credit cards are and send the definition itself. Another peer may use it to check whether some subset of its own credentials fulfills the request. This boils down to gathering the relevant concept definitions in the policy (so-called *abbreviation rules*) and sending them to the other peer that reasons with those rules locally.

In other words, in [2, 5] *peers communicate by sharing their ontologies*. Interestingly, typical policies require peers to have a common a priori understanding only of the predicate representing credentials and arithmetic predicates, because any other predicate can be understood simply by sharing its definition. Therefore, the only nontrivial knowledge to be shared is the X.509 standard credential format. In this framework, interoperability based on ontology sharing is already at reach! This is one of the aspects that make policies and automated trust negotiation a most attractive application for semantic web ideas.

Another interesting proposal of [5] is the notion of *declaration*, that has already been discussed in Section 3. This was the first step towards a more flexible

and lightweight approach to policy enforcement, aiming at a better tradeoff between protection efforts and risks.

This framework was chosen as the starting point for the work of WG I2, because according to [9] it was still one of the most complete trust negotiation systems in 2002. The major limitation was the lack of distributed negotiations and credential discovery, which are now supported as specified in [3, 2]. As we already pointed out, a first approach at integrating crisp and soft notions of trust is described in [3] and in these proceedings.

4.2 Negotiations

In response to a resource request, a web server may ask for some credentials, proving that the client can access the resource. However, the credentials themselves are sensitive resources, in general. So the two peers are in a completely symmetrical situation: the client, in turn, may ask the server for credentials (say, proving that it participates into the Better Business Bureau program) before sending off the required credentials. Each peer decides how to react to incoming requests according to a local policy, which is typically a set of rules written in some logic programming dialect. As we already pointed out, requests are formulated by selecting some rules from the policies.

This basic schema has been refined along the years taking several factors into account [1, 2, 5, 6, 12–15].

First, policy rules may possibly inspect a *local state* (such as a legacy database) that typically is not accessible by the other peers. In that case, in order to make rules intelligible to the recepient, they are first partially evaluated w.r.t the current state.

Second, *policies themselves are sensitive resources*, therefore not all relevant rules are shown immediately to the peer. They are first filtered according to policy release rules; the same schema may be applied to policy release rules themselves for an arbitrary but finite numer of levels.

As a consequence, some negotiations that might succeed, in fact fail just because the peers do not tell each other what they want. The study of methodologies and properties that guarantee negotiation success (when appropriate) is an interesting open research issue.

Moreover, *credentials are not necessarily on the peer's host*. It may be necessary to locate them on the network [7]. As part of the automated support to *cooperative enforcement*, peers may give each other hints on where a credential can be found [16].

There are further complications related to actions (cf. Section 4.3). In order to tune the negotiation strategy to handle all these aspects optimally, PROTUNE—the core policy language of REWERSE—supports a *metapolicy language* [3, 2] that specifies which predicates are sensitive, which are associated to actions, which peer is responsible for each action, where credentials can be searched for, etc., thereby guiding negotiation in a declarative fashion and making it more cooperative and interoperable. Moreover, the metapolicy language can be used

to instantiate the framework in different application domains and link predicates to the ontologies where they are defined.

4.3 Provisional policies

Policies may state that certain requests or decisions have to be logged, that the system itself should search for certain credentials, etc. In other words, policy languages should be able to specify *actions*. Event-condition-action (ECA) rules constitute one possible approach. Another approach, supported by the current core policy language of REWERSE, consists in labelling some predicates as *provisional*, and associating them to actions that (if successful) make the predicate true [3, 2]. It may also be specified that an action should be executed by some other peer; this results in a request.

A cooperative peer tries to execute the actions under its responsibility whenever this helps in making negotiations succeed. For example, provisional predicates may be used to encode business rules. The next rule (formulated in PROTUNE's language) enables discounts on low_selling articles in a specific session:

$$\begin{aligned}&\texttt{allow}(Srv) \leftarrow \ldots, \texttt{session}(ID),\\ &\quad \texttt{in}(X, \texttt{sql:query}(\texttt{'select} * \texttt{from low\_selling'}),\\ &\quad \texttt{enabled}(\texttt{discount}(X), ID)\,.\end{aligned}$$

Intuitively, if $\texttt{enabled}(\texttt{discount}(X), ID)$ is not yet true but the other conditions are verified, then the negotiator may execute the action associated to `enabled` and the rule becomes applicable (if $\texttt{enabled}(\texttt{discount}(X), ID)$ is already true, no action is executed). The (application dependent) action can be defined and associated to `enabled` through the metapolicy language of PROTUNE. With the metalanguage one can also specify when an action is to be executed.

Some actions would be more naturally expressed as ECA rules. However, it is not obvious how the natural bottom-up evaluation schema of ECA rules should be integrated with the top-down evaluation adopted by the current core language. The latter fits more naturally the abductive nature of negotiation steps. The integration of ECA rules in the core policy language is one of the open issues in REWERSE's agenda.

4.4 Stateful vs. stateless negotiations

The negotiations described above are in general stateful, because (i) they may refer to a local state—including legacy software and data—and (ii) the sequence of requests and counter requests may become more efficient if credentials and declarations are not submitted again and again, but are rather kept in a local negotiation state.

However, negotiations are not *necessarily* stateful:

- the server may refuse to answer counter-requests, or—alternatively—the credentials and declarations disclosed during the transaction may be included in every message and need not be cached locally;

– the policy does not necessarily refer to external packages.

In other words, stateless protocols are just special cases of the frameworks introduced so far. Whether a stateless protocol is really more efficient depends on the application. Moreover, efficiency at all costs might imply less cooperative systems.

The question is: *are stateful protocols related to scalability issues?* We do not think so. The web started as a stateless protocol, but soon a number of techniques have been implemented to simulate stateful protocols and transactions in a number of real world applications and systems, capable of answering a huge number of requests per time unit. We must observe that if the support for stateful negotiations had been cast into http, then probably many of the intrinsic vulnerabilities of simulated solutions (like cookies) might have been avoided.

So we think that policy languages and frameworks for the web should support both stateful and stateless protocols to face the variety of different needs of web applications.

4.5 What's new?

The existing approaches to trust management and trust negotiation already tackle the need for flexible, knowledge based interoperability, and take into account the main idiosyncrasies of the web—because ATN frameworks have been designed with exactly that scenario in mind. Today, to make a real contribution (even in the context of a policy-aware web), one should work on the open issues of trust management, that include at least the following topics:

– Negotiation success: how can we guarantee that negotiations succeed despite all the difficulties that may interfere? For example: rules not disclosed because of lack of trust; credentials not found because their repository is unknown. What kind of properties of the policy protection policy and of the *hints* (see Section 4.2) guarantee a successful termination when the policy "theoretically" permits access to a resource?
– Optimal negotiations: which strategies optimize information disclosure during negotiation? Do any reasonable preconditions prevent unnecessary information disclosure?
– A related problem is: In the presence of multiple ways of fulfilling a request, how should the client choose a response? One needs both a language for expressing preferences, and efficient algorithms for solving the corresponding optimization problem. While this negotiation step is more or less explicitly assumed by most works on trust negotiation, there is no concrete proposal so far.

Moreover, the integration of abductive semantics and ECA semantics is an open issue, as we have pointed out in a previous section. Of course this list is not exhaustive.

5 Cooperative policy enforcement

Cooperative enforcement involves both machine-to-machine and human-machine aspects. The former is handled by negotiation mechanisms: published policies, provisional actions, hints, and other metalevel information (see Section 4.2) can be interpreted by the client to identify automatically what information is needed to access a resource, and how to obtain that information. The human-machine interaction aspect deserves some discussion.

One of the causes of the enormous number of computer security violations on the Internet is the users' lack of technical expertise. In particular, users are typically not aware of the security policies applied by their system, not to speak about how those policies can be changed and how they might be improved by tailoring them to specific needs. As a consequence, most users ignore their computer's vulnerabilities and the corresponding countermeasures, so the system's protection facilities cannot be effectively exploited.

For example, it is well known that the default, generic policies that come with system installations—biased toward functionality rather than protection—are significantly less secure than a policy specialized to a specific context, but very few users know how to tune or replace the default policy. Moreover, users frequently do not understand what the policy is really checking up on, and hence they are unaware of the risks involved in many common operations.

Similar problems affect privacy protection. In trust negotiation, credential release policies are meant to achieve a satisfactory tradeoff between privacy and functionality (many interesting services cannot be obtained without releasing some information about the user). However, one cannot expect such techniques to be effective unless users are able to understand and possibly personalize the privacy policy enforced by their system.

Additionally, a better understanding of a web service's policy makes it easier for a first-time user to interact with the service. If a denied access results simply in a "*no*", then the user has no clue on how he or she can possibly acquire the permission to get the desired service (e.g., by completing a registration procedure, by supplying more credentials, by filling in some form, etc.) This is why we are advocating a form of *cooperative policy enforcement*, where negative responses are enriched with suggestions and other explanations whenever such information does not violate confidentiality (sometimes, part of the policy itself is sensitive).

For these reasons, WG I2 selected as one of its main objectives *greater user awareness and control on policies*. We are making policies easier to understand and formulate to the common user in the following ways:

- We adopt a *rule-based policy specification language*, because such languages are very flexible and at the same time structurally similar to the natural way in which policies are expressed by nontechnical users.
- We are making the policy specification language more friendly by developing a *controlled natural language* front-end to translate natural language text into executable rules (see next section).

– We are developing *advanced explanation mechanisms* to help the user understand what policies prescribe and control.

We have just published a deliverable on such explanation mechanisms [4]. It contains a requirements analysis for explanations in the context of automated trust negotiation (ATN). Moreover, we define explanation mechanisms for *why, why-not, how-to*, and *what-if* queries. There are several novel aspects in our approach:

– We adopt a *tabled explanation structure* as opposed to more traditional approaches based on single derivations or proof trees. The tabled approach makes it possible to describe infinite failures (which is essential for *why not* queries).
– Our explanations show simultaneously different possible proof attempts and allow users to see both local and global proof details at the same time. Such combination of local and global (intra-proof and inter-proof) information is expected to facilitate navigation across the explanation structures.
– We introduce suitable heuristics for focussing explanations by removing irrelevant parts of the proof attempts. Anyway, we provide a second level of explanations where all the missing details can be recovered, if desired.
– Our heuristics are *generic*, i.e. domain independent. This means that they require no manual configuration.
– The combination of tabling techniques and heuristics yields a completely novel method for explaining failure. In the past, the problem has been ignored or formulated differently.

Moreover, we make our explanation mechanisms *lightweight* and *scalable* in the sense that (i) they do not require any major effort when the general framework is instantiated in a specific application domain, and (ii) most of the computational effort can be delegated to the clients.

Queries are answered using the same policy specifications used for negotiation. Query answering is conceived for the following categories of users:

– Users who are trying to understand how to obtain access permissions;
– Users who are monitoring and verifying their own privacy policy;
– Policy managers who are verifying and monitoring their policies.

Currently, advanced queries comprise *why/why not, how-to*, and *what-if* queries.

Why/why not queries can be used by security managers to understand why some specific request has been accepted or rejected, which may be useful for debugging purposes. Moreover, why-not queries may help a user to understand what needs to be done in order to obtain the required permissions (a process that in general may include a combination of automated and manual actions). Such features are absolutely essential to enforce security requirements without discouraging users that try to connect to a web service for the first time. How-to queries have a similar role, and differ from why-not queries mainly because the former do not assume a previous query as a context, while the latter do.

What-if queries are hypothetical queries that allow to predict the behavior of a policy before credentials are actually searched for and before a request is actually submitted. What-if queries are good both for validation purposes and for helping users in obtaining permissions.

Among the technical challanges related to explanations, we mention:

– Finding the right tradeoff between explanation quality and the effort for instantiating the framework in new application domains. Second order explanation systems prescribe a sequence of expensive steps, including the creation of an independent domain knowledge base expressly for communicating with the user. This would be a serious obstacle to the applicability of the framework.

5.1 Natural language policies

Policies should be written by and understandable to the final users, to let them keep the behavior of their system under control. Otherwise the risk that users keep on adopting generic (hence ineffective) built-in policies, and remain unaware of which controls are actually made by the system would be extremely high—and this would significantly reduce the benefits of a flexible policy framework.

Of course, most users have no specific training in programming nor in formal logics. Fortunately, they spontaneously tend to formulate policies as rules; still, logical languages may be intimidating.

For this reason, we are designing front-ends based on graphical formalisms as well as *natural language interfaces*. We would like policy rules to be formulated like: "*Academic users can download the files in folder historical_data whenever their creation date precedes 1942*".

Clearly, the inherent ambiguity of natural language is incompatible with the precision needed by security and privacy specifications. For this reason we adopt a *controlled* fragment of English where a few simple rules determine a unique meaning for each sentence. This approach is complemented with a suitable interface that clarifies what the machine understands.

Some working group members have long standing expertise in controlled natural language, and a natural language front-end based on the ATTEMPTO system (`http://www.ifi.unizh.ch/attempto/`) is being progressively adapted to the need of policy languages, including negation as failure and deontic constructs.

6 Conclusions and perspectives

In our vision policies are really knowledge bases: a single body of declarative rules used in many possible ways, e.g., for negotiations, query answering, and other forms of system behavior control.

As far as trust negotiation is concerned, transparent interoperation based on ontology sharing can potentially become "everyday technology" in a short time (cf. Section 4.1). As such, trust negotiation may become a success story for semantic web ideas and techniques.

We are currently implementing the features described in this paper by extending the PeerTrust automated negotiation system (`http://www.learninglab.de/english/projects/peertrust.html`) and the parser of the Attempto system. Our current prototypes support real credential verification and real distributed computations, as well as the grammar structure and the anaphora mechanism underlying the sample natural language rule illustrated above.

Recently, the Semantic Web community declared interest in a notion of policy very similar to REWERSE's approach (see the paper co-authored by Hendler and Berners-Lee in these proceedings). While the examples outlined there are just special cases of what can be done in the old framework introduced in [5], there is evidence of interest in more advanced and powerful developments, in line with the curret goals of the research in ATN. There seems to be an emphasis on stateless interactions (i.e. degenerate negotiations consisting of one step only) defended by scalability arguments. However, we do not believe in such a priori restrictions. After all, the web started as a stateless protocol, but soon a number of techniques have been implemented to simulate stateful protocols and transactions in a number of real world applications.

We insist on the importance of *cooperative policy enforcement and trust management*, that give common users better understanding and control on the policies that govern their systems and the services they interact with. The closer we get to this objective, the higher the impact of our techniques and ideas will be.

References

1. M. Blaze, J. Feigenbaum, and M. Strauss. Compliance Checking in the PolicyMaker Trust Management System. In *Financial Cryptography*, British West Indies, February 1998.
2. P.A. Bonatti and D. Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *IEEE 6th Intl. Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, pages 14–23. IEEE Computer Soc., 2005.
3. P.A. Bonatti and D. Olmedilla. Policy specification language. Technical Report I2-D2, REWERSE, Feb 2005. http://www.rewerse.net.
4. P.A. Bonatti, D. Olmedilla, and J. Peer. Advanced policy queries. Technical Report I2-D4, REWERSE, Aug 2005. http://www.rewerse.net.
5. P.A. Bonatti and P. Samarati. A uniform framework for regulating service access and information release on the web. *Journal of Computer Security*, 10(3):241–272, 2002. Short version in the Proc. of the Conference on Computer and Communications Security (CCS'00), Athens, 2000.
6. Rita Gavriloaie, Wolfgang Nejdl, Daniel Olmedilla, Kent E. Seamons, and Marianne Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *1st European Semantic Web Symposium (ESWS 2004)*, volume 3053 of *Lecture Notes in Computer Science*, pages 342–356, Heraklion, Crete, Greece, may 2004. Springer.
7. N. Li, W. Winsborough, and J.C. Mitchell. Distributed Credential Chain Discovery in Trust Management (Extended Abstract). In *ACM Conference on Computer and Communications Security*, Philadelphia, Pennsylvania, November 2001.

8. Arnon Rosenthal and Marianne Winslett. Security of shared data in large systems: State of the art and research directions. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 962–964. ACM, 2004.
9. K. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobsen, H. Mills, and L. Yu. Requirements for Policy Languages for Trust Negotiation. In *3rd International Workshop on Policies for Distributed Systems and Networks*, Monterey, CA, June 2002.
10. V. S. Subrahmanian, Piero A. Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Ozcan, and Robert Ross. *Heterogenous Active Agents*. MIT Press, 2000.
11. V.S. Subrahmanian, S. Adali, A. Brink, R. Emery, J.J. Lu, A. Rajput, T.J. Rogers, R. Ross, and C. Ward. Hermes: Heterogeneous reasoning and mediator system. http://www.cs.umd.edu/projects/publications/ abstracts/hermes. html.
12. W. Winsborough, K. Seamons, and V. Jones. Negotiating Disclosure of Sensitive Credentials. In *Second Conference on Security in Communication Networks*, Amalfi, Italy, September 1999.
13. W. Winsborough, K. Seamons, and V. Jones. Automated Trust Negotiation. In *DARPA Information Survivability Conference and Exposition*, Hilton Head Island, SC, January 2000.
14. Marianne Winslett, Ting Yu, Kent E. Seamons, Adam Hess, Jared Jacobson, Ryan Jarvis, Bryan Smith, and Lina Yu. Negotiating trust on the web. *IEEE Internet Computing*, 6(6):30–37, 2002.
15. Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Trans. Inf. Syst. Secur.*, 6(1):1–42, 2003.
16. C. Zhang, P.A. Bonatti, and M. Winslett. Peeraccess: A logic for distributed authorization. In *12th ACM Conference on Computer and Communication Security (CCS 2005)*. ACM. To appear.

Design and Application of Rule Based Access Control Policies

Huiying Li, Xiang Zhang, Honghan Wu, Yuzhong Qu

Department of Computer Science and Engineering, Southeast University,
Nanjing 210096, P.R.China
{huiyingli, xzhang, hhwu, yzqu} @seu.edu.cn

Abstract. Access control is an important issue among the security problems of resources in distributed systems. In order to enable entities in distributed systems to understand and interpret policies correctly, common concern is drawn to the problem of expressing access control policies with semantic information. In this paper, we introduce how to express access control policies based on OWL and SWRL. It includes a definition of OWL ontology to describe the terms and a declaration of SWRL rules to explicit the relationship between properties. Finally some use cases are given to explain how to express policies in form of rule using the terms defined beforehand.

1 Introduction

The dissemination and manipulation of information resources across large-scale networks of computers is increasingly prevalent. As a result, common concern is drawn to the security of resources in distributed systems, which deals with many aspects, such as identification and authentication of users, encryption of resources and access control issue. Instead of dealing with all the aspects, this paper introduces how to design rule-based access control policies using semantic language-OWL (Web Ontology Language)[10] and rule language-SWRL (Semantic Web Rule Language)[3].

Policy language has been studied for a long time, there are also some languages can express access control policy, such as XACML (eXtensible Access Control Markup Language)[12], X-RBAC (XML Role-Based Access Control)[6], KAoS(Knowledge-able Agent-oriented System)[5], Rei[8] and Ponder[9]. Among them, XACML is an OASIS standard specification now. It defines a general policy language based on XML used to protect resources as well as an access decision language. X-RBAC is based on an extension of the role-based access control model, it provides a framework for specifying mediation policies in a multidomain environment and allows specification of RBAC policies and facilitates specification of timing constraints on roles and access requirements as well [6]. Both XACML and X-RBAC are based on XML, and they have had broad

applications in some enterprise environments. However it is difficult for them to deal with the inter-operation at the level of semantics, because the entities and relationships defined by XML are lack of formal meaning. In order to enable entities in distributed systems to understand and interpret polices correctly, it will be better to represent policy language in semantic language such as RDF-S, DAML+OIL or OWL[7]. KAoS and Rei both have accomplished some work in this regard. Providing an open distributed architecture for software agents is the initial purpose to develop KAoS[5], it could also express the agents action control policy relying on the DAML-based ontology in Web/Grid Service environment. Represented in OWL-Lite, Rei is a policy language based on deontic concepts and includes constructs for rights, prohibitions, obligations and dispensations [8]. Another policy language is Ponder, which is a declarative, object-oriented language that can be used to specify both security and management policies [9]. Different from the policy languages discussed above, we design the access control policies based on OWL and SWRL in the form of rules using entities defined in advance.

As mentioned above, policy language expressed in ontology language has the advantage of dealing with inter-operation at semantics level. We define ontology to help express access control policies using OWL, which added considerable expressive ability. But OWL still has the expressive limitations, particularly with respect to what can be said about properties [1]. For enhancing the expressive and deducible ability, we also define some rules to explain the relationship between properties using SWRL, a Horn clause rules extension to OWL.

The paper is structured as follows: a brief introduction to SWRL is given in Section 2. Following this, the ontology and rules designed by us are presented in Section 3. In Section 4, we talk about the advantage of our method to express policy using some use cases. Finally, the summary of our work and future research directions are discussed in Section 5.

2 Semantic Web Rule Language

SWRL is a Horn clause rules extension to OWL proposed as a W3C member submission in 2004. It extends OWL DL by adding a simple form of Horn-style rules in a syntactically and semantically coherent manner for the purpose of enhancing expressive ability.

The main axiom added to OWL DL is Horn clause rules, which are of the form of an implication between an antecedent (body) and consequent (head). The informal meaning of a rule can be read as: whenever (and however) the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

A rule written in an informal human readable syntax has the form below:

$$\text{antecedent} \rightarrow \text{consequent}.$$

Both the antecedent and consequent of a rule consist of zero or more atoms. Atoms can be of the form C(x), P(x, y), Q(x, z), sameAs(x, y) or differentFrom(x, y), where C is an

OWL DL description which may be a class name or a complex description using Boolean combination, restrictions etc. P is an OWL DL individual-valued Property, Q is an OWL DL data-valued Property, and x, y are either variables or OWL individuals, z is either a variable or an OWL data value. Informally, an atom C(x) holds if x is an instance of the class description C, an atom P(x, y) (or Q(x, z)) holds if x is related to y(z) by property P(Q), an atom sameAs(x, y) holds if x is interpreted as the same object as y, and an atom differentFrom(x, y) holds if x and y are interpreted as different objects.

Having the rules discussed above, it will be easy to assert more complex relationships between properties, such as the composition of two properties. A typical example is showed as follows:

$$\text{parent}(?a, ?b) \wedge \text{brother}(?b, ?c) \rightarrow \text{uncle}(?a, ?c).$$

Hereinto, variables are indicated using the standard convention of prefixing them with a question mark, parent and brother are both OWL properties. Then the rule above asserts that the composition of parent and brother properties implies the uncle property. In other words, if John has Mary as a parent and Mary has Bill as a brother, then this rule requires that John has Bill as an uncle.

Using SWRL, we define some rules similar with the one presented above to express relationships between the properties in our OWL ontology. It makes the meaning of the property more clear. We also allow users to assert their own access control policies in the form of rules using entities defined in our ontology.

3 Design of Rule Based Access Control Policies

In overview, we design access control policies in the form of rules using entities defined in our OWL ontology. The ontology includes concepts about agent and resource and properties between them such as **isPermittedDoWith** and **isProhibittedDoWith**. We also express some rules using SWRL to describe the relationships between the properties defined in the ontology. Currently, our application domain is the project of WonderSpace, which is a Semantic Web application developed by our Lab (http://xobjects.seu.edu.cn/). Its potential users are researcher groups. The project aims at speeding up their research and study, and bringing high efficiency in collaborations by providing an environment for resource and knowledge sharing.

3.1 The ontology

The purpose of designing our ontology is to help users express their access control policies in WonderSpace, the core classes and properties are defined as Fig. 1.

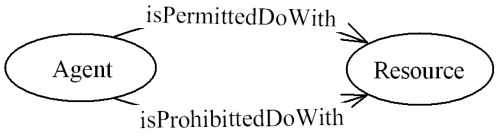


Fig. 1. Graph model of the core classes

Using this model, user can assert that what kinds of agents are permitted/prohibited to access to what kinds of resources. The access control policy is associated with the properties of the agent instead of identities. For example, a research group leader published a resource and specified the access control policy like this: only PhD students in my group are permitted to download this resource. Such policy can be expressed in the form of rules using classes and properties defined in our ontology. For the reason of providing more properties for user to describe agent and resource, we design the Agent class and Resource class showed in Fig. 2 and Fig.3. We also specify many subproperties of **isPermittedDoWith** and **isProhibittedDoWith** to express more actions that can/cannot take to the resource, such as **isPermittedPublishWith**, **isPermittedDeleteWith**, **isPermittedDownloadWith**, **isPermittedReadWith**, **isPermittedUpdateWith**, which are all the subproperties of **isPermittedDoWith**, while **isProhibittedPublishWith**, **isProhibittedDeleteWith**, **isProhibittedDownloadWith**, **isProhibittedReadWith**, **isProhibittedUpdateWith** are all subproperties of **isProhibittedDoWith**.

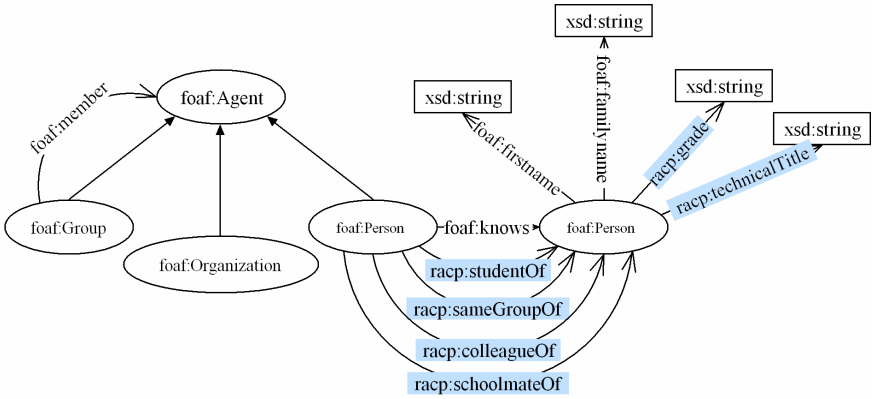


Fig. 2. Agent ontology

Fig. 2 shows the main properties and subclasses of Agent class, the entities are borrowed from the ontology of Friend of a Friend (FOAF) project. FOAF project is about creating a Web of machine-readable homepages describing people, the links between them and the things they create and do (http://www.foaf-project.org/). FOAF has already done a good work about developing ontology about Agent, so we use the basic classes and

properties in the FOAF ontology for reference. For the purpose of expressing domain information about agent, we also extend the ontology by adding some properties, some of them are DatatypeProperty such as **grade**, **technicalTitle**, some are ObjectProperty to describe the relationship between peoples such as **studentOf**, **sameGroupOf, colleagueOf**, **schoolmateOf**. We define these four objectproperties as the subproperty of **knows**, and some of them had subproperties likewise, for example, the **phdStudentOf**, **masterStudentOf**, **collegeStudentOf** are the subproperty of **studentOf.**

The resource ontology showed in Fig. 3 is the one used in the WonderSpace now. It denotes the classification of resources and figures out some attributes of resources.

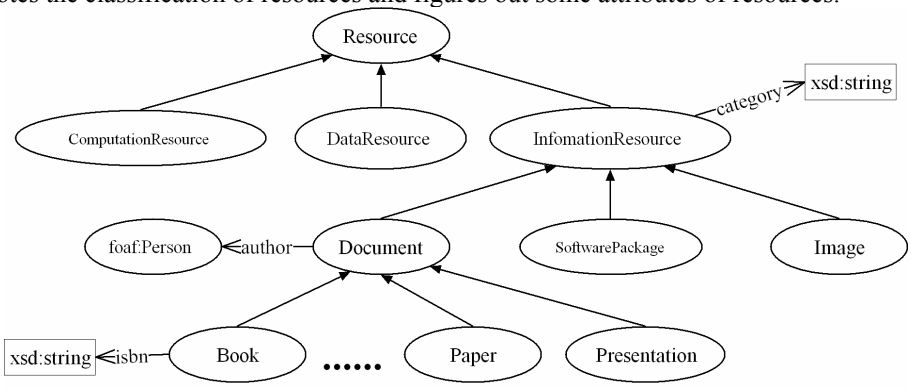


Fig. 3. Resource ontology

Currently, the ontology discussed above is relatively rough, and some entities defined in it are domain dependent. One of our future researches is to define a more general and domain independent ontology.

3.2 The rules

Besides the ontology, we give more explicit meaning to the properties by defining some rules, such as:

$$member(?z, ?x) \wedge member(?z, ?y) \wedge Person(?x) \wedge Person(?y) \rightarrow sameGroupOf(?x, ?y).$$

It means that if ?x and ?y are persons and group ?z has the member ?x and ?y, then ?x and ?y have the relationship of **sameGroupOf**. In fact, this rule gives a more clear meaning to **sameGroupOf** property. And a group may declare all its members, it can be deduced that any two of these members has the relationship of **sameGroupOf** then.

Another rule different from the one above looks like this:

$$phdSchoolmateOf(?x, ?y) \wedge phdStudentOf(?y, ?z) \rightarrow phdStudentOf(?x, ?z).$$

That is to say, if ?y is a PhD student of ?z, and ?x and ?y have the relationship of **phdSchoolmateOf** (note: the meaning of this property in the context is that they are both PhD students of one superviser), we can deduce that ?x is also a PhD student of ?z. It

means that although there do not have the explicit declaration to denote ?x is a PhD student of ?z, it can be reasoned using the exiting information and the rules.

By adding such rules, the expressive ability and reasoning ability are both enhanced commendably.

4 Case Study

Now, the application domain of policy language is WonderSpace, a Semantic Web application developed by our Lab. The basic scenario in WonderSpace is that user can upload the resources s/he wants to share with others to the server and specify access control policy about the resources using entities defined in our ontology. Whenever a user requests to access to one resource, the server will decide to accept or reject the request by using the policy and FOAF information of the users stored in server.

Here, we propose some use cases to denote how to express access control policies.

Example 1. Jack published a paper and asserted that the members of WonderSpace group could read this paper, while the members of DrSNP group could download it.

$$\text{member}(\text{wonderspace}, ?x) \rightarrow \text{isPermittedReadWith}(?x, \text{paper1}),$$
$$\text{member}(\text{drsnp}, ?y) \rightarrow \text{isPermittedDownloadWith}(?y, \text{paper1}).$$

Example 2. Colin published a presentation and asserted the access control policy like this: only my PhD students in the same group with me will be allowed to download this presentation. This policy can be expressed as follows:

$$\text{sameGroupOf}(?x, \text{colin}) \wedge \text{phdStudentOf}(?x, \text{colin}) \rightarrow \text{isPermittedDownloadWith}(?x, \text{presentation1}).$$

Example 3. One of the system policies may be announced like this: only the students of a professor can publish a book. Though it looks a little unreasonable (in some cases it will be useful), it may be represented as follows:

$$\text{technicalTitle}(?z, \text{“professor”}) \wedge \text{studentOf}(?x, ?z) \wedge \text{Book}(?y) \rightarrow \text{isPermittedPublishWith}(?x, ?y).$$

As **phdStudentOf** is the subproperty of **studentOf**, considering this rule, it can be deduced that a PhD student of a professor can also publish book.

Example 4. A student Mary uploaded a note and denoted that: my teachers and their colleagues were prohibited to read this note,

$$\text{studentOf}(\text{Mary}, ?x) \wedge \text{colleagueOf}(?x, ?z) \rightarrow \text{isProhibittedReadWith}(?x, \text{note1}) \wedge \text{isProhibittedReadWith}(?z, \text{note1}).$$

Our policy engine is under development, it is undoubtedly a hard work because of the complex reason over policies, but it is also proved an interesting work. For simplifying the problem, we suppose that both resources and related policies and the FOAF information of users are all creditable and stored in the server. When a user proposes a request to access (such as download) to one resource, the policy engine will search all related policy about the resource and verify one by one whether the user satisfy the conditions specified in the policy by using the FOAF information. Take example 2 as an

instance, if Bill requests to download presentation1, this policy will be found and the engine will look into Bill's FOAF file and try to find if there have the **sameGroupOf** and **phdStudentOf** properties related to Colin. If succeed, Bill will be allowed to download presentation1, if not (one may possibly not declare all relationship in his FOAF file), the engine will look through Colin's FOAF file and try to deduce the relationships between Bill and Colin. At the same time, because there has a rule about **sameGroupOf** defined in advance, the engine will also investigate whether the group Colin belongs has announced that both Bill and Colin are all its members. These are only basic ideas about the policy engine, it still have a lot of work for us to do.

5 Conclusion and Future Work

We have introduced an approach to design access control policies based on OWL and SWRL. Our work includes the definition of OWL ontology and some policy rules, it help users to specify who are permitted/prohibited to take what action to what resources. The most obvious feature of our work different from early work is that we use rules to help users express their access control policies. We give more formal meaning to the property defined in OWL ontology and help the engine deduce more information by adding some rules. And we also allow the users to assert their access control policies in the form of rule, which is more nature to the users, and they will express the policy more accurately because of the powerful expressive ability of rules.

As discussed earlier, the policy ontology and rules is still domain dependent in some ways. One of our future researches is to develop a more general ontology. The reasoning engine is also an important work under consideration. And we are planning to develop a full access control policy language and apply it to the no central controlled environment: it means that the information about users and resources are not controlled centrally in server, and are incomplete sometimes. It will give more obvious prominence to the language's advantage of interoperability at the level of semantics and powerful reasoning ability. Of course, it will bring other problems too, such as the resolution of policy confliction, the problem of trust. Our future work will also include how to resolve these problems.

Acknowledgments

This work is supported in part by National Key Basic Research and Development Program of China under grant 2003CB317004 and in part by JSNSF under grant BK2003001. We would like to thank the members of WonderSpace group for their suggestions on this paper.

References

1. I. Horrocks, P. F. Patel-Schneider: A Proposal for an OWL Rules Language. WWW (2004) ACM
2. I. Horrocks, P. F. Patel-Schneider, and F. Harmelen: From SHIQ and RDF to OWL: The Making of a Web Ontology Language. Journal of Web Semantics (2003)
3. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean: SWRL: A semantic web rule language combining owl and ruleml. W3C Member Submission, 21 May 2004. Available at http://www.w3.org/Submission/SWRL/.
4. I. Horrocks, B. Parsia, P. F. Patel-Schneider, and J. Hendler: Semantic Web Architecture: Stack or Two Towers? PPSWR (2005): Accepted Papers
5. J. M. Bradshaw, S. Dutfield, P. Benoit, and J. D. Woolley:KAoS: Toward An Industrial-Strength Open Agent Architecture. Software Agents, J.M. Bradshaw (ed.), AAAI Press (1997) 375-418
6. J.B.D Joshi.: Access-control language for multidomain environments. Internet Computing, IEEE Volume 8, Issue 6, Nov.-Dec(2004) 40 - 50
7. L. Kagal, T. Finin, and A. Joshi: A policy language for a pervasive computing environment. IEEE 4th International Workshop on Policies for Distributed Systems and Networks (2003). http://citeseer.ist.psu.edu/kagal03policy.html
8. L. Kagal: Rei Ontology Specifications, Ver 2.0. http://www.cs.umbc.edu/~lkagal1/rei/.
9. N. Damianou, N. Dulay, E. Lupu, and M. Sloman: The ponder policy specification language. The Policy Workshop (2001) , Bristol U.K. Springer-Verlag, LNCS 1995
10. P.F. Patel-Schneider, P. Hayes, I. Horrocks (eds.): OWL: Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation 10 February 2004. Latest version is available at http://www.w3.org/TR/owl-semantics/
11. P. Hayes (ed.):RDF Semantics. W3C Recommendation 10 February 2004. Latest version is available at http://www.w3.org/TR/rdf-mt/
12. T. Moses, Entrust Inc: eXtensible Access Control Markup Language (XACML), Ver 2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

Rule-based and Ontology-based Policies: Toward a Hybrid Approach to Control Agents in Pervasive Environments

Alessandra Toninelli[1], Jeffrey M. Bradshaw[2], Lalana Kagal[3], Rebecca Montanari[1]

[1]Dipartimento di Elettronica, Informatica e Sistemistica
Università di Bologna
Viale Risorgimento, 2 - 40136 Bologna - Italy
{atoninelli, rmontanari}@deis.unibo.it

[2] Florida Institute for Human and Machine Cognition (IHMC)
40 S. Alcaniz Street, Pensacola, FL 32502, USA
jbradshaw@ihmc.us

[3]MIT CSAIL
32 Vassar Street, Boston, MA, USA
lkagal@csail.mit.edu

Abstract. *Policies are being increasingly used for controlling the behavior of complex multi-agent systems. The use of policies allows administrators to regulate agent behavior without changing source code or requiring the consent or cooperation of the agents being governed. However, policy-based control can sometimes encounter difficulties when applied to agents that act in pervasive environments characterized by frequent and unpredictable changes. In such cases, we cannot always specify policies a priori to handle any operative run time situation, but instead require continuous adjustments to allow agents to behave in a contextually appropriate manner. To address these issues, some policy approaches for governing agents in pervasive environments specify policies in a way that is both context-based and semantically-rich. Two approaches have been used in recent research: an ontology-based approach that relies heavily on the expressive features of Description Logic (DL) languages, and a rule-based approach that encodes policies as Logic Programming (LP) rules. The aim of this paper is to analyze the emerging directions for the specification of semantically-rich context-based policies, highlighting their advantages and drawbacks. Based on our analysis we describe a hybrid approach that exploits the expressive capabilities of both DL and LP approaches.*

1. Introduction

The multi-agent paradigm offers a promising software engineering approach for the development of applications in complex environments [1]. By their ability to operate autonomously without constant human supervision, agents can perform tasks that would be impractical or impossible using traditional software techniques [2]. However, this autonomy, if unchecked, has also the potential of causing severe damage if agents are poorly designed, buggy, or malicious.

Explicit policies can help in dynamically regulating the behavior of agents and in maintaining an adequate level of security, predictability, and responsiveness to human control. Policies provide the dynamic bounds within which an agent is permitted to function autonomously and limit the possibility of unwanted events occurring during operations. By changing policies, agent behavior can be continuously adjusted to accommodate variations in externally imposed constraints and environmental conditions without modifying the agent code or requiring the cooperation of the agents being governed [3].

Until recently, policies have been primarily exploited to govern complex distributed systems within traditional computing environments that rely on a relatively fixed set of resources, users, and services. However, with the Internet becoming ubiquitous, researchers started to investigate how to develop adequate policy-based techniques for controlling agent behavior within pervasive environments [4]. The dynamicity, unpredictability and heterogeneity of pervasive environments complicate the design of policy languages and techniques for agent control. Resources are not pre-determined, interacting agents are not always known a priori and, if agents roam across different network localities, they have different resource visibility and availability, depending on their location and on other context-dependent information, such as local security policies and resource state. In this setting, agents need to be provided with a semantically clear and interoperable description of the context where they execute and need to acquire, reason about and negotiate the policies that rule their behavior in each new context, so that they can decide whether to adhere or not. In addition, policies for controlling agent behavior cannot always be specified beforehand to handle any operative run-time situation, but may require a high rate of dynamic and continuous adjustments to allow agents to act in any execution context in the most suitable way and to accommodate context changes.

To address these issues, some policy approaches for pervasive environments are starting to emerge that share common design principles [5, 6, 7]. A significant design concept that guides these approaches is the use of contextual information for driving policy specifications. Since context is a prime quality of pervasive environments, it should explicitly appear in policy specifications. In fact, in pervasive scenarios it is almost impossible to know the identities or roles of all agents that are likely to interact and request services in advance. Instead of defining identity- or role-based policies, administrators may more easily define the conditions for making resources available and

for allowing or denying agents resource visibility and access, according to the context of their operating conditions.

Another important design principle is the adoption of a semantically-rich representations for policy definition. Semantically-rich representations permit both structure and properties of the elements of a pervasive system and the management operations themselves (e.g., policies) to be described at a high level of abstraction, thus enabling policy conflict detection and harmonization.

Recent research efforts in the area of semantically-rich context-based approaches to policy representation follow one of two possible directions. *Ontology-based* approaches rely largely on the expressive features of Description Logic languages, such as OWL [7], to classify contexts and policies, thus enabling deductive inferences and static policy conflict resolution to be performed. In contrast, *rule-based* approaches take the perspective of Logic Programming to encode rules in a clear, logic-like way. Moreover, a rule-based approach facilitates the straightforward mapping of policies to lower level enforcement mechanisms thanks to its concise and understandable syntax.

The scope of this paper is to analyze the emerging directions for the specification of semantically-rich context-based policies, highlighting their advantages and drawbacks. Based on our analysis we describe a hybrid approach that exploits the expressive capabilities of both approaches. On the one hand, it should rely on an ontology-based approach to enable policy classification, comparison and static conflict resolution. On the other hand, it should be able to reap the benefits of a rule-based approach, thus enabling the efficient enforcement of policies defined over dynamically determined values.

The structure of the paper follows. Section 2 outlines some fundamental requirements for policy languages to enable the specification of semantic context-based policies. Section 3 analyzes how some relevant well-known approaches to semantic policy representation, i.e., KAoS and Rei, deal with the aforementioned requirements. The comparison allows us to discuss, in Section 4, a possible direction toward the integration of ontology-based and rule-based policy approaches in order to exploit their full advantages. Conclusions and future work are presented in Section 5.

2. Novel Requirements for Semantic Context-based Policies

The control of agent behavior in pervasive scenarios raises novel requirements for the design of policy languages and policy run-time environments. In pervasive scenarios, users, on behalf of whom agents act, typically move from one environment to another, thus determining continuous variations in their physical position and in their execution context, including the set of entities and resources they may be able to interact with. Moreover, users can access the network using various devices, e.g., laptops, PDAs or mobile phones, which exhibit different capabilities in terms of resources and computational abilities. As it is not possible to exactly predict all the interactions an entity may be involved in and the kind of resources it may wish to have access to, policy-based

control cannot rely on any precise knowledge about the subjects/events/actions that need to be governed.

To deal with such environmental characteristics, recent research efforts propose to adopt semantically-rich representations to express policy and domain knowledge. The adoption of Semantic Web languages to specify and manage policies in pervasive computing scenarios brings several advantages. In fact, semantically-rich representations ensure that there is a common understanding between previously unknown entities about their capabilities, the current execution context and the actions they are permitted or obliged to perform. Moreover, modeling policies at a high level of abstraction simplifies their description and improves the analyzability of the system. Semantic Web languages also enable expressive querying and automated reasoning about policy representation.

Another emerging direction suggests that, in order to deal with the dynamic context changes that are typical of pervasive applications, it may be advantageous to build policies directly over context conditions, i.e., to consider context as a primary element in the specification of policies [8]. Context is a complex notion that has many definitions. Here we consider context as any information that is useful to characterize the state or the activity of an entity, e.g., its location or its characteristics, and any useful information about the world in which this entity operates, e.g., date and time. In pervasive environments, where client users are typically unknown and where context operating conditions frequently change even unpredictably, the specification of context-based policies, instead of traditional subject-or role-based ones, allows to control the behavior of entities without having to foresee all the possible interactions that an entity may have with other entities and resources.

The adoption of a semantic context-based policy approach to control pervasive systems requires the definition of a policy model that can precisely identify the basic types of policies required to control agents, can specify how to express and represent context and related policies in a semantically expressive form, and how to enforce them. In this paper we focus particularly on specification rather than on enforcement issues. To this extent, we consider the following as basic requirements for a semantic-based policy language:

- the ability to model and represent the contexts in which agents operate and to which policies are associated, at a high level of abstraction.
- the ability to define what actions are permitted or forbidden to do on resources in specific contexts (*authorizations* or *permission/prohibition* policies);
- the ability to define the actions that must be performed on resources in specific contexts (*obligations*).

The aim of this paper is not to provide a general survey of the state-of-the-art in context-based policy representation, but to carefully analyze how some relevant semantic policy approaches deal with the specification of semantic context-based policies. We first present KAoS, followed by Rei, both of which were originally designed for governing agent behavior and that represent, respectively, significant examples of ontology-based and rule-based policy languages. Then, from this analysis, we derive some suggestions toward the design of a hybrid policy approach.

3. Semantic Approaches to Context-based Policy Specification

To illustrate the expressive capabilities of the two considered policy frameworks, i.e., KAoS and Rei, let us consider a usage scenario that is likely to become more and more usual in the next future. Let us consider the case of a traveler, Alice, who is waiting at the airport for her flight to leave. The airport is equipped with several 802.11b hot spots providing travelers with wireless connectivity for their portable devices, e.g., laptops, PDA and mobile phones. Airline companies may also provide additional services and resources to travelers, such as the possibility to print documents stored on their devices using public printers that are placed in various areas of the airport. Moreover, while waiting to board, users may wish to share files with other users, by exploiting the wireless connectivity available at the airport. Since she has to wait a couple of hours for her plane to leave, Alice starts to work on some documents she has on her laptop. Therefore, she may wish to access the printer that is available in the waiting area around her boarding gate to print the documents she needs. In addition, as she likes jazz music very much, she would like to exchange music files with other travelers waiting in the airport hall. These activities need to be regulated by appropriate policies. In particular, the following policies governing adequate access to services and resource sharing may apply. We will use these policies as a running policy example throughout the rest of the paper.

LocationSharing Policy
Users that are currently co-located with the owner of the policy, i.e., with her device, are authorized to access the shared files stored on the owner device.

This policy may be instantiated and enforced by Alice to share her music files with co-located travelers in a secure way, depending on current context conditions.

PrinterAccess Policy
Travelers that are flying with a company of the Sky Team group, and are currently located in the airport area including gate from 31 to 57 are authorized to access the printer.

This policy may be enforced by the provider of a printing service that is offered to travelers flying with the Sky Team alliance in some areas of the airport. The enforcement of this authorization should ensure that travelers having proper rights are enabled to access the service. Furthermore, if the default behavior of the system states that everything that is not explicitly permitted is prohibited, this policy also prevents unauthorized travelers from accessing the service.

In the following sections we first analyze KAoS, and subsequently Rei, showing how they deal with the specification of the previously described policies.

KAoS

KAoS is a framework that provides policy and domain management services for agent and other distributed computing platforms [9, 10, 17]. It has been deployed in a wide variety of multi-agent and distributed computing applications. KAoS policy services allow for the specification, management, conflict resolution and enforcement of policies within agent domains. KPAT, a powerful graphical user interface, allows non-specialists to specify and analyze complex policies without having to master the complexity of OWL.
KAoS adopts an ontology-based approach to semantic policy specification. In fact, policies are mainly represented in OWL [7] as ontologies. The KAoS policy ontologies distinguish between authorizations and obligations. In KAoS, a policy constrains the actions that an agent is allowed or obliged to perform in a given context. In particular, each policy controls a well-defined action, whose subject, target and other context conditions are defined as property restrictions on the action type. Figure 1a shows an example of KAoS authorization, which represents the *PrinterAccess* policy previously described. The property *performedBy* is used to define the class to which the actor must belong for the policy to be satisfied.

```
<owl:Class rdf:ID="SkyTeamGate31-57PrinterAccessAction">
<owl:intersectionOf rdf:parseType="Collection">
 <owl:intersectionOf rdf:parseType="Collection">
 <rdfsowl:Class rdf:about="&action;AccessAction"/>
  <owl:Restriction>
      <owl:onProperty rdf:resource="&action;performedBy"/>
      <owl:allValuesFrom rdf:resource="#SkyTeamCustomer"/>
 </owl:Restriction>
  <owl:Restriction>
      <owl:onProperty rdf:resource="&action;accessedEntity"/>
      <owl:allValuesFrom rdf:resource="#Printer31-57"/>
  </owl:Restriction>
</owl:intersectionOf>
 </owl:Class>


    < policy:PosAuthorizationPolicy rdf:ID=" SkyTeamGate31-57PrinterAccess">
      < policy:controls rdf:resource="# SkyTeamGate31-57PrinterAccessAction"/>
      <policy:hasSiteOfEnforcement rdf:resource="&some-ontology;TargetSite"/>
      <policy:hasPriority>10</policy:hasPriority>
    </policy:PosAutihorizationPolicy>
```

a)

```
    <owl:Class rdf:ID="SkyTeamCustomer">
    <rdfs:subClassOf rdf:resource="&some-ontology;Customer"/>
     <rdfs:subClassOf>
       <owl:Restriction>
            <owl:onProperty rdf:resource="&some-ontology;firm"/>
            <owl:allValuesFrom rdf:resource="&some-ontology;SkyTeamAlliance"/>
      </owl:Restriction>
```

```
</rdfs:subClassOf>
```

b)

Fig. 1. KAoS policy examples.

In KAoS, context conditions that constrain a policy may be specified through the definition of appropriate classes defined via property restrictions. In particular, two main properties, i.e., the *hasDataContext* and the *hasObjectContext* properties, and their sub-properties are used to characterize the action context. Some sub-properties are defined in the KAoS ontology, like for instance the ones defining the actor (*performedBy*), the time and the target resource (*accessedEntity*) of an action, while others may be created within domain-specific ontologies. Figure 1b shows the definition of a class, namely *SkyTeamCustomer*, which represents all the individuals that are flying with a company belonging to the Sky Team alliance. This class is defined as a subclass of the *Customer* class, having the *affiliation* property restricted to the Sky Team.

As these examples show, KAoS is based on an *ontological* approach to policy specification, which exploits OWL, i.e., description logic, features to describe and specify policies and context conditions. In fact, contexts and related policies are expressed as ontologies. Therefore, KAoS is able to classify and reason about both domain and policy specification basing on ontological subsumption, and to detect policy conflicts statically, i.e., at policy definition time.

However, a pure OWL approach encounters some difficulties with regard to the definition of some kinds of policies—specifically those requiring the definition of *variables*. For instance, by relying purely on OWL, we could not define policies such as the *FileSharing* policy, which defines constraints over property values that refer to statically unknown values, e.g., the policy owner location. Other examples include policies that contain parametric constraints, which are assigned a value only at deployment or run time. For this reason, KAoS developers have introduced role-value maps as OWL extensions and implementing them within the Java Theorem Prover, used by KAoS [11, 17]. The adoption of role value maps, description logic-based concept constructors that were originally introduced in the KL-ONE system [12], allows KAoS to specify constraints between property values expressed in OWL terms, and to define policy sets, i.e., groups of policies that share a common definition but can be singularly instantiated with different parameters. The proposed extensions effectively add sufficient expressive flexibility to KAoS to represent the policies discussed in this paper. However, non-experienced users may have difficulties in writing and understanding these policies without the help of the KPAT graphical user interface.

Rei

Rei is a policy framework that permits to specify, analyze and reason about declarative policies defined as norms of behavior [4, 6]. Rei adopts a rule-based approach to specify

semantic policies. Rei policies restrict domain actions that an entity can/must perform on resources in the environment, allowing policies to be developed as contextually constrained deontic concepts, i.e., right, prohibition, obligation and dispensation. The first version of Rei (Rei 1.0) is defined entirely in first order logic with logical specifications for introducing domain knowledge [13]. The current version of Rei (Rei 2.0), that we analyze in this paper, adopts OWL-Lite to specify policies and can reason over any domain knowledge expressed in either RDF or OWL [4].

A policy basically consists of a list of rules and a context that is used to define the policy domain. Rules are expressed as OWL properties of the policy. In particular, the *policy:grants* property is used to associate a deontic object with a policy either directly or via a *policy:Granting*. Figure 2 shows the Rei 2.0 policy specification of the *LocationSharing* policy. In order to specify context conditions, one or more constraints must be defined. A constraint, which may be simple or boolean, i.e., the boolean combination of a pair of simple constraints, defines a set of actors or a set of actions that fulfill a certain property. A simple constraint, as shown in Figure 2b, is modeled as a triple consisting of a subject, a predicate and an object, which defines the value of the property for the entity, following a pattern that is typical of logical languages like Prolog.

A constraint can be associated to a policy at three different levels. The first possibility is to impose a constraint within the definition of a deontic object, by means of the *deontic:constraint* property, as shown in Figure 2c. In this case, the constraint can be expressed over the actor, the action to be controlled or over generic environmental states, e.g., the time of the day. Additional constraints can be imposed within the Granting specification over the entity the granting is made to, the deontic object the granting is made over and, again, over generic environmental states. Finally, it is possible to express a set of constraints directly within the policy definition through the *policy:context* property. These constraints are generically defined as conditions over attributes of entities in the policy domain.

```
<policy:Policy rdf:ID="FileAccessPolicy">
    <policy:actor rdf:resource="#requester"/>
    <policy:grants rdf:resource="#Perm_FileAccess"/>
</policy:Policy>
<policy:Policy rdf:ID=”FileSharingPolicy”>
  ...
</policy:Policy>
```

a)

```
<constraint:SimpleConstraint rdf:ID="LocationOfUser">
     <constraint:subject rdf:resource="&some-ontology;user"/>
     <constraint:predicate rdf:resource="&some-ontology;location"/>
     <constraint:object rdf:resource="#user-location"/>
</constraint:SimpleConstraint>

<constraint:SimpleConstraint rdf:ID="CoLocatedWithUser">
     <constraint:subject rdf:resource="#requester"/>
     <constraint:predicate rdf:resource="&some-ontology;location"/>
```

```
     <constraint:object rdf:resource="#user-location"/>
</constraint:SimpleConstraint>

<constraint:And rdf:ID="Constraint_CoLocated">
     <constraint:first rdf:resource="#LocationOfUser"/>
     <constraint:second rdf:resource="#CoLocatedWithUser"/>
</constraint:And>
```

b)

```
<deontic:Permission rdf:ID="Perm_FileAccess">
    <deontic:actor rdf:resource="#requester"/>
    <deontic:action rdf:resource="&some-ontology;AccessToSharedFiles"/>
    <deontic:constraint rdf:resource="#Constraint_CoLocated"/>
</deontic:Permission>
```

c)

Fig. 2. Rei policy examples.

Rei 2.0 uses OWL-Lite for the specification of policies and of domain-specific knowledge. Though represented in OWL-Lite, Rei still allows the definition of variables that are used as placeholders as in Prolog. In fact, as shown in Figure 2b, the definition of constraints follows the typical pattern of rule-based programming languages, like Prolog, i.e., defining a variable and the required value of that variable for the constraint to be satisfied. In this way, Rei overcomes one of the major limitations of the OWL language, and more generally of description logics. i.e., the inability to define variables. For example, as shown in Figure 2, Rei allows developers to express a policy stating that a user is allowed to access the shared files of another user if they are located in the same area, whereas pure OWL would not allow for the definition of the "same as" concept. Therefore, Rei's rule-based approach enables the definition of policies that refer to a dynamically determined value, thus providing greater expressiveness and flexibility to policy specification.

On the other hand, the choice of expressing Rei rules similarly to declarative logic programs prevents it from exploiting the full potential of the OWL language. In fact, Rei rules knowledge is treated separately from OWL ontology knowledge due to its different syntactical form. OWL inference is essentially considered as an oracle, i.e., the Rei policy engine treats inferences from OWL axioms as a virtual fact base. Hence, Rei rules cannot be exploited in the reasoning process that infers new conclusions from the OWL existing ontologies, which means that the Rei engine is able to reason about domain-specific knowledge, but not about policy specification. As a main consequence of this limitation, Rei policy statements cannot be classified by means of ontological reasoning. Therefore, in order to classify policies, the variables in the rules need to be instantiated, which reduces to a constraint satisfiability problem. Let us consider, for example the previously described *PrinterAccess* policy. Unlike KAoS, Rei does not allow for a policy disclosure process that retrieves policies controlling a specific *type* of action. Hence, the user willing to use the printer could only try to access it and see what the Rei engine has answered, with regard to this particular access. For the same reason, Rei cannot statically detect

conflicts, like KAoS does, but it can only discover them with respect to a particular situation.

4. Toward a Hybrid Approach to Semantic Policy Specification?

The management of context and related policies is a demanding task and requires the appropriate description of context and subsequent policies. Our analysis of current approaches to semantic context-based policy specification has outlined two main research directions, which move from two opposite sides.

On one side, a purely ontology-based approach exploits description logic, e.g., OWL, to describe contexts and associated policies at a high level of abstraction, in a form that allows their classification and comparison. This feature is essential, for instance, in order to detect conflicts between policies before they are actually enforced, thus granting interoperability among entities belonging to different domains that adopt different policies. In fact, by means of a preliminary analysis of policy typologies holding in different domains, the required behaviors of each domain can be compared and harmonized, if needed, avoiding the cost of failures due to conflicts arising in the enforcement phase. Another interesting application of an ontology-based approach lies in the possibility of exploiting policy description to facilitate negotiation in policy disclosure. As an entity may wish to interact with potentially untrusted entities, negotiating policy disclosure may help interacting parties in the effort of reaching an agreement about their mutual behavior without imposing too heavy limitations to their privacy.

On the other side, a rule-based approach relies on the features of logic programming languages, e.g., Prolog, to enable evaluation and reasoning about concrete context and policy instances. In fact, from the enforcement point of view, policy rules can be considered as “instructions” to be executed provided that their activating conditions, i.e., contexts, are evaluated to be true. This perspective suggests that contexts and related policies should be expressed in a clear, concise and expressive way to facilitate their evaluation and enactment, similarly to the code of a programming language that needs to be compiled or interpreted. For example, the language should allow for the definition of policies over dynamically determined constraints, including run time variables, as this is a crucial expressive feature that most programming languages offer.

KAoS and Rei represent intermediate approaches between the two opposite approaches previously described. KAoS was originally based only on description logic, provided by the OWL language features, but current features aim at overcoming the intrinsic limitations of OWL as a description logic-based language, i.e., the inability to allow variable-based reasoning [11, 17]. Rei, the first version of which was strongly oriented to a declarative logic programming approach, has recently moved from a Prolog-like syntax to an OWL encoding that permits ontological reasoning over domain knowledge (but not over policy rules), mainly to solve the extensibility problem of Rei 1.0.

We claim that a policy framework for pervasive computing systems should be able to provide support to context modeling and evaluating with different levels of granularity and flexibility. In particular, we suggest the possibility of an integrated approach that exploits both description logic (DL) and logic programming (LP). At a higher level of granularity and abstraction, DL should be exploited to classify contexts and related policies, thus allowing static conflict resolution and favoring gradual policy disclosure between interacting parties. At a more operational level, LP should be used to encode rules in a clear and expressive fashion that may also facilitate their enforcement.

Let us consider, for instance, the *LocationSharing* policy. This policy can be described using description logic through the definition of the ontological concepts of location context and co-location, thus enabling classification, comparison and static conflict detection with other policies that are related with the same concepts. On the other side, logic programming can be used to encode the rule that effectively enforces the policy, namely: if user Y is located in the same place as user X, then user Y is allowed to access user X's shared files.

Let us illustrate with an example how a policy framework could benefit from such a hybrid approach to policy specification. Let us suppose that Bob is waiting to check-in at the airport and wishes to share some music files with other travelers at the airport. In order not to waste battery, he would like to avoid a random approach where he just tries to access other users devices to share files with them, without knowing in advance if the access will be permitted or denied. To this extent, before attempting to send or receive files from his portable device, he asks other users to disclose their public access control policies. Let us suppose that the *LocationSharing* policy is in force and active on Alice laptop. Upon receiving Bob request for policy disclosure, she retrieves and sends the policies that controls the "file sharing" action type, i.e., the *LocationSharing* policy. At this point, Bob can statically check for conflicts between Alice's policy and his own policies controlling the same type of action, i.e., file sharing. If, for instance, Bob has enforced a policy to control file sharing that does not include any location context, then he can deduce that there is no conflict between his own policy and Alice's. It is worth noting that policy disclosure and conflict detection is enabled by the ontology-based definition of policies. On the other hand, when Bob actually tries to access Alice shared files, the access control policy is enforced on the basis of its rule-based definition, by evaluating the current value of variables, i.e., Alice and Bob current location. Let us note that, due to the dynamic nature of the policy whose evaluation can be made only at access time, it may still be possible that Bob is not allowed to access Alice files because of his current location. However, thank to the preliminary policy disclosure phase, Bob is able to decide whether he agrees to adhere to a policy that imposes some conditions on user location. If, for instance, Bob is waiting at the check-in desk and he already knows that his location will not change in the next hour because there is a very long queue, then he may decide to choose another user that does not impose any condition on location.

It is worth stating that an integrated approach like the one we have described would require the establishment of a semantic and inferential correspondence between DL and LP. This is a complex issue, which nonetheless may be addressed, as demonstrated, for

instance, by recent work. For example, the approach described in [15] could represent a valid guideline toward a viable integration process. According to the authors, the idea was on the one hand to enable to "build rules on top of ontologies", i.e., enable the rule knowledge base to have access to DL ontological definitions for vocabulary primitives, and on the other hand to enable to "build ontologies over rules", i.e., enable ontological definitions to be supplemented by rules, or imported into DL from rules. Let us note that Rei seems to have taken the first approach, as in its latest version it allows to specify policy rules using policy and domain ontologies. KAoS, by means of appropriate extensions of the OWL language, is aiming at supplementing its ontological specification of policies with rules. In particular, the authors of [15] propose a mapping between DL and LP, based on the consideration that, under appropriate restrictions, both logics can be considered as restricted sets of first order logic. As a final consideration, we believe that the way toward interoperation between rules and ontologies could be further explored to achieve more powerful and flexible expressive means for the specification of context-based policies.

5. Conclusions and Future Work

The specification of semantically-rich context-based policies to regulate agent behavior in pervasive environments is a complex task that requires appropriate representations to describe both context information relevant for policy specification and the policies themselves. Our analysis of current approaches to semantically-rich context-based policy specification has described two main research directions that are moving toward the middle from two opposite sides, i.e., an ontology-oriented approach, based on description logic features, and a rule-oriented approach, based on logic programming. The paper proposes a hybrid approach to policy specification that allows better handling of the highly dynamic, uncertain and heterogeneous conditions that are typical of the pervasive environments where agents operate. This paper has analyzed the problem and the issues to be solved in developing such a hybrid policy approach. Further investigation is needed to conduct more formal and thorough analyses of existing and proposed systems in order to understand their strengths and weaknesses, and to propose the basis for new research and development. Along this direction, stimulating ideas and results can come from the investigation of existing proposals, such as SWSL [16], which describes the attempt to combine first-order logic with rule based languages to specify the Semantic Web Services ontologies as well as individual Web services.

References

1. Jennings, N., "An agent-based approach for building complex software systems"., Communications of the ACM, 44(4), pp. 35-41, 2001.

2. Bradshaw, J. M. (Ed.). Software Agents. Cambridge, MA: The AAAI Press/The MIT Press, 1997.
3. Bradshaw, J. M., Jung, H., Kulkarni, S., & Taysom, W. "Dimensions of adjustable autonomy and mixed-initiative interaction". In M. Klusch, G. Weiss, & M. Rovatsos (Ed.), Computational Autonomy, Springer-Verlag, Berlin, Germany, 2004.
4. Kagal, L.: "A Policy Based Approach to Governing Autonomous Behavior in Distributed Environments". Dissertation submitted to the Faculty of the Graduate School of the University of Maryland for the degree of Doctor of Philosophy, Baltimore County, USA, 2004.
5. Tonti, G., Bradshaw, J. M., Jeffers, R., Montanari, R., Suri, N., Uszok, A.: "Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder", Proc. of the Second International Semantic Web Conference (ISWC2003), LNCS, Vol. 2870. Springer-Verlag, Berlin, pp. 419-437, Sanibel Island, Florida, USA, October 2003.
6. Kagal, L., Finin, T., Joshi, A.: "A Policy Language for Pervasive Computing Environment" In: Proc. of IEEE Fourth International Workshop on Policy (Policy 2003). Lake Como, Italy, pp. 63-76, IEEE Computer Society Press 4-6 June 2003.
7. Van Harmelen, F., et al.: "OWL Web Ontology Language Reference, W3C Recommendation 10 February 2004", http://www.w3.org/TR/owl-ref/.
8. Montanari, R., Toninelli, A., Bradshaw, J.M.: "Context-Based Security Management for Multi-Agent Systems", To be published In: Proc. of the Second IEEE Symposium on Multi-Agent Security and Survivability, IEEE Press, Philadelphia, USA, 30-31 August 2005.
9. Bradshaw, J. M., Uszok, A., Jeffers, R., Suri, N., Hayes, P., Burstein, M. H., Acquisti, A., Benyo, B., Breedy, M. R., Carvalho, M., Diller, D., Johnson, M., Kulkarni, S., Lott, J., Sierhuis, M., & Van Hoof, R. (2003). Representation and reasoning for DAML-based policy and domain services in KAoS and Nomads. *Proceedings of the Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2003).* 14-18 July, Melbourne, Australia. New York, NY: ACM Press, pp. 835-842
10. Uszok, A., et al.: "KAoS policy management for semantic web services". IEEE Intelligent Systems, 19(4), p. 32-41, 2004.
11. Moreau, L., Bradshaw, J., Breedy, M., Bunch, L., Johnson, M., Kulkarni S., Lott J., Suri N., Uszok A.: "Behavioural Specification of Grid Services with the KAoS Policy Language", Proc. of the Cluster Computing and Grid 2005. Cardiff, UK, 9-12 May 2005.
12. Schmidt-Schauss, M.: "Subsumption in KL-ONE is undecidable", In: Proc. of the First Intl Conference on the Principles of Knowledge Representation and Reasoning (KR 1989), Morgan Kaufmann: Los Altos, 1989.
13. Kagal, L.: Rei: "A Policy Language for the Me-Centric Project", HP Labs Technical Report, HPL-2002-270, 2002.
14. N. Damianou, et al., "The Ponder Policy Specification Language," Proc. 2nd Int'l Workshop Policies for Distributed Systems and Networks, LNCS 1995, Springer-Verlag, pp. 18-38, 2001.
15. Grosof, B.N., Horrocks I., Volz, R., and Decker, S.: "Description Logic Programs: Combining Logic Programs with Description Logic", In: Proc. of WWW 2003, 20-24 May 2003, Budapest, Hungary, 2003.
16. Battle S., et al., Semantic Web Service Language (SWSL), Version 1.0. http://www.daml.org/services/swsf/1.0/swsl/ (2005).
17. Uszok, A., Bradshaw, J. M., Jeffers, R., Tate, A. & Dalton, J. (2004). Applying KAoS services to ensure policy compliance for semantic web services workflow composition and enactment. In S. A. McIlraith, D. Plexousakis, F. van Harmelen (Eds.), *The Semantic Web—ISWC 2004,* Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, November 7-11, LNCS 3298, Berlin: Springer, pp. 425-440.

Policy Based Dynamic Negotiation for Grid Services Authorization

Ionut Constandache, Daniel Olmedilla, and Wolfgang Nejdl

L3S Research Center and University of Hannover, Germany
{constandache,olmedilla,nejdl}@l3s.de

Abstract. Policy-based dynamic negotiations allow more flexible authorization in complex Grid environments, and relieve both users and administrators from up front negotiations and registrations. This paper describes how such negotiations overcome current Grid authorization limitations, and how policy-based negotiation mechanisms can be easily integrated into a Grid infrastructure. Such an extension provides advanced access control and automatic credential fetching, and can be integrated and implemented in the new version 4.0 of the Globus Toolkit.

1 Introduction

Grid environments provide the middleware needed for access to distributed computing and data resources. Distinctly administrated domains form virtual organizations and share resources for data retrieval, job execution, monitoring, and data storage. Such an environment provides users with seamless access to all resources they are authorized to. In current Grid infrastructures, in order to be granted access at each domain, user's jobs have to secure and provide appropriate digital credentials for authentication and authorization. However, while authentication along with single sign-on can be provided based on client delegation of X.509 proxy certificates [21] to the job being submitted, the authorization mechanisms are still mainly identity based. Due to the large number of potential users and different certification authorities, this leads to scalability problems calling for a complementary solution to the access control mechanisms specified in the current Grid Security Infrastructure (GSI) [8].

In this paper, following up previous work described in [2], we address the limitations in current grid environments and introduce an extension to the Grid Security Infrastructure and Globus Toolkit 4.0 in which policy-based negotiation mechanisms offer the basis for overcoming these limitations. This extension includes property-based authorization mechanisms, automatic gathering of required certificates, bidirectional and iterative trust negotiation and policy based authorization, ingredients that provide advanced self-explanatory access control to grid resources.

The rest of the paper is organized as follows: in section 2 we describe the current limitations and motivate our approach. Section 3 provides a brief overview of policy based trust negotiation and further section 4 introduces this mechanism to Grid illustrating its benefits. Our proposed architecture along with a description of our implementation is given in section 5, while section 6 presents related approaches to Grid authorization. Finally, section 7 concludes our presentation and describes further work.

2 Motivation

A group of scientists at the Engineering Department of a well known University needs to obtain some simulation data regarding oceanic water waves in order to develop signaling instruments for tsunami hazards avoidance. Fortunately, the University and the Navy Institute have an agreement, allowing University members to utilize any of the Institute scientific instruments, as long as it is not already in use and the University has not exceeded its 40 monthly allocated hours using Institute resources. Both the University and the Navy Institute support Globus Toolkit 4.0 [7] providing the middleware the scientists need to collect the required data from a Wave Tank available at the Institute site.

Alice, the leader of our group of scientists, prepares a job and submits it to the University HPC Center Linux Cluster via the University Grid Portal. Together with the job, Alice delegates to the Linux Cluster an X.509 Proxy Certificate [17] which can further be used by the job to prove that is acting on Alice's behalf. Normally, Alice would directly use her long-term X.509 End Entity Certificate[1] in order to delegate the proxy certificate[2] to the job. However, as she is not in her office today (and therefore she does not have access to her End Entity certificate), she instructs the University Grid Portal to retrieve such a proxy certificate from The University MyProxy [12] online credential repository and to use it to delegate a proxy certificate to the job running on the University HPC Center Linux Cluster. Once the job is executed, it will set up the Navy Institute Wave Tank, retrieve and correct the simulation data and save it to the University Mass Storage Solution.

For each remote resource involved, the job has to authenticate (achieving single sign-on through its delegated proxy certificate) and be authorized (the identity of the certificate is checked against access control lists, for example by mapping the identity of the certificate to a local account by means of a gridmap file). In addition, Alice has been informed that the Navy Institute requires a proof of her University affiliation before being granted access to any instrument. Therefore, Alice has programmed the job to contact the University Community Authorization Service(CAS) [13] and retrieve, after another round of authentication and identity based authorization, a statement attesting Alice's involvement with the University. Using this certificate and assuming that all the other Wave Tank local requirements are fulfilled (the Wave Tank is not in use and the University has not exceeded its allocated hours) authorization is granted at the Navy Institute site. Finally after one more authentication and authorization round with the University Reliable File Transfer Service the corrected and refined data is stored, being available to Alice and her group.

Although it seems that this scenario (depicted in figure 1) fits Alice's and the resource owners' needs, there are some implicit assumptions which are necessary for Alice's job to succeed:

[1] An End Entity certificate is a long-term certificate issued by a certification authority and therefore, its private key must be stored securely to prevent unauthorized access.

[2] A Proxy certificate is a certificate issued by an End Entity certificate or another proxy certificate with a lifetime of several hours. Due to this short lifetime, it is considered safe to store its private key unencrypted, protected only by file system local permissions.

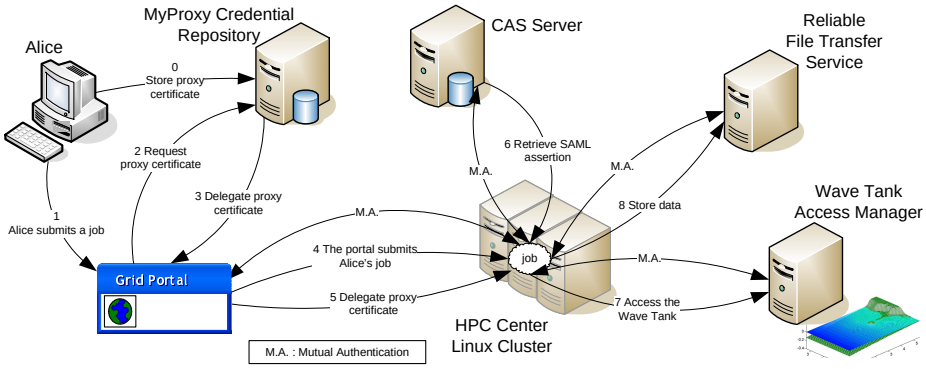


Fig. 1. Grid Scenario

- One credential for all. A job is submitted together with a proxy certificate signed by a Certification Authority (CA) which is further used to authenticate to other resources. This assumes that all resources trust the same CA.
- Identity Based Authorization. Resources, where a job is allowed access, have to know in advance the identity of the certificate.
- Simple Authentication/Authorization. It is based on a one-shot process where the job requests access and the resource grants or denies it.
- Manual Credential Fetching. Users need to find out in advance which credentials are required to access each resource and program their jobs to fetch and give these credentials while authenticating/requiring authorization.

These requirements become liabilities when the Grid grows more complex and the number of resources a job has to access increases. Mapping identities raises serious scalability problems due to the large number of potential users, even more when we take into account the difficult requirement of having a single trusted Certification Authority (which is hard enough to have within one Grid and not feasible when trying to integrate different Grids). Because of these reasons, property based certificates have started to appear (PRIMA [10], VOMS [1], CAS [13] and X.509 attribute certificates [5]) even though there is no standard interface for using them yet. Furthermore, access control is not a simple task, as both a job and a resource might require to specify constraints in the way they disclose their certificates. This asks for extension of the usual one-shot mechanism to an iterative process where the level of trust increases at each iteration [22]. Finally, resources should advertise the credentials they require thus allowing the job to perform dynamic credential fetching. This not only frees job owners of "what resource requires what credential" problem [3] or that of coding credential fetching within their jobs but also allows dynamic selection of resources as they would be self explanatory in terms of authorization requirements.

[3] Currently, users must keep track of the right/required credential for each resource they might access. Application needs may be difficult to predict prior to its execution and the user might not be available when a certain credential is required. On the other hand providing the application with all user credentials is not feasible as it may imply revealing sensitive information.

We argue in the following sections that Grid with support for specifying, advertising and enforcing service level access control policies together with capabilities for automatic fetching of credentials can indeed suit large scale collection of resources, enabling dynamic negotiation for authorization and access granting based on parties properties, and can be implemented on top of the Globus Toolkit 4.0.

3 Trust Negotiation

Distributed environments like the Web, P2P systems or Grids are built with the assumption that service providers and consumers are known to each other. In common scenarios before allowing access to (possibly) sensitive resources, entities establish trust relations by having clients pass a registration phase. This registration phase consists of the creation of an account, the addition of the user identity to an access control list or some offline contact specifying who provides the service, under what conditions and to which consumers. This becomes a liability due to the lengthy process the registration phase supposes (resource administrators often have to manually verify user data e.g. associations with institutions or projects), which delays resource usage and restricts its availability even if the client is entitled to access it. Moreover clients are required to reveal sensitive information (name, residence, position or credit card number) with no mean of imposing their own requirements checking the service provider reliability.

Trust relations are constructed based on a set of digital credentials (e.g. certificates, signed assertions, capabilities or roles) which are disclosed by two entities to prove certain properties they pose. Digital credentials bind the identity of the holder to a public key, are signed by a credential issuer and may attest a quality of the holder. Since the credential may include personal information (e.g., roles and capabilities) users should be entitled to have their own requirements with regard to the entity to which this information is released. In such a case, a more adequate approach would be that distributed environments treat clients and service providers equally, similarly to P2P systems, both having the ability of imposing restrictions on resource disclosure (being resource services or credentials).

Trust negotiation [22] provides a gradual establishment of trust between parties through requests for and disclosure of credentials in an iterative and bilateral process. The requests for credentials can be viewed as the resource authorization policies, enabling upon satisfaction the authorization decision for accessing the resource. Trust negotiation approach distinguishes from the identity based access control in the following regards:

- Trust is established between previously unknown parties based on their properties.
- Providers and consumers can define policies to protect their resources (credentials or services provided).
- Authorization is established incrementally through a sequence of mutual requirements and disclosures of credentials.
- The authorization process may involve more than two entities and their trusted credential issuer, as requirements for a credential may determine a new negotiation with a third entity, which at its turn may call for another negotiation and so on.

Trust negotiation is triggered when one party requests to access a resource owned by another party. The goal of a trust negotiation is to find a sequence of credentials

$(C_1, \cdots, C_k, R)$ where R is the resource where access is attempted, such that when credential C_i is disclosed, its access control policy has been satisfied by credentials disclosed earlier in the sequence, or to determine that no such credential disclosure sequence exists.

Having introduced the concepts behind trust negotiation we concentrate in the next sections on our vision of a policy based trust negotiation mechanism which enables an automated authorization scheme over a Grid environment.

4 Dynamic Negotiation for Grid Services Authorization

Our initial example (illustrated in section 2) involved two distinct domains sharing resources over a Grid: one administrated by the University and the other by the Navy Institute. University resources (Grid Portal, MyProxy Credential Repository, Linux Cluster, CAS) have been carefully setup to interoperate and previous contacts with the Navy Institute have established the requirements for addressing its instruments. In this section we provide a more flexible scenario able to accommodate a large, loosely coupled Grid environment, reducing the current management overhead.

We propose a scheme in which entities advertise their authorization requirements as access control policies and are able to query for these policies and fulfill them through automatic credential fetching. Credentials can be protected by policies that have to be satisfied by the requesting part before having them revealed. Acting in a self describing environment, services and clients will automatically negotiate authorization by iteratively increasing their trust relationship, through credential disclosures until a decision regarding authorization can be made. The following scenario reflects these new functionalities:

Alice attends a Grid Conference in another country. She can not log into the University Grid Portal as for security issues it can only be addressed from the University network. Luckily the conference organizers provide a policy enabled Grid Portal and Alice can use it to submit her job. She logs into the Conference Grid Portal with her registration number and instructs the portal to obtain a delegated certificate from the University MyProxy Server.

Policies set up by the Conference Grid Portal administrators permit user requests for delegated certificates if the user can prove to be registered at the conference and the MyProxy Server contacted belongs to an accredited university. The University MyProxy Server has no problem in disclosing a credential signed by the State Ministry of Education attesting University accreditation, but when asked to delegate a credential on behalf of a user, the University MyProxy Server has its own requirement as to receive a credential signed by IEEE or Verisign. The Conference Grid Portal has an IEEE signed credential attesting its organization under IEEE supervision but first has to check whether this certificate is protected by a policy. Indeed this is the case. The policy states that the entity asking for the IEEE credential has to prove her affiliation with a university. This has already been achieved through the certificate previously disclosed by the University MyProxy Server so the IEEE signed certificate is disclosed and negotiation succeeds with the Conference Grid Portal retrieving a credential for Alice's job.

For performance issues, Alice decides to submit her job to a Linux Cluster belonging to the Research Center for Aeronautical Sciences. For the job to be submitted, a

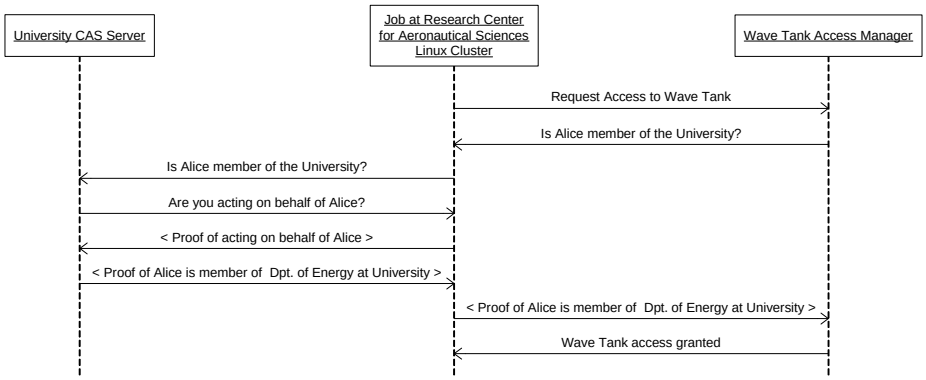


Fig. 2. Simplified sequence diagram of the job negotiations

new round of negotiation takes place as the Linux Cluster policies require the client to provide a credential attesting her acting as member of a project present in the State Ministry of Education database. The Conference Grid Portal forwards a query to the State Ministry of Education CAS and, by providing Alice's job credential, retrieves a certificate attesting that Alice participates in a project regarding signaling instrumentation. By using this certificate, job authorization is granted at the Research Center for Aeronautical Sciences Linux Cluster. As the job needs to contact further resources, a Proxy Certificate is delegated by the Grid Portal to the Linux Cluster.

The job resumes its work by querying Navy Institute policies for Wave Tank access (job negotiations are represented in figure 2). It finds out that it has to prove Alice's association with a university organization. To demonstrate this, the job contacts the University CAS server and retrieves an assertion attesting that Alice is member of Department of Engineering at her university, not before proving to the CAS server that it is acting on Alice's behalf. Again we assume that the other local Navy Institute policies are fulfilled (the number of hours allocated to the University has not been exceeded and the Wave Tank is not in use) so the job can setup the Wave Tank and start receiving data.

For storing the output data with the University Reliable File Transfer (RFT) Service, the job has to provide a credential signed by the University Certification Authority (CA). Such a credential has been previously delegated to the Linux Cluster and, by providing its chain of certificates [4], the job meets the University RFT Service requirements and it is allowed to store the results without further negotiation.

The job has been submitted with only one credential, but then dynamically negotiated authorization at each resource accessed and whenever required, knowing where to retrieve the needed credentials. This was possible due to the contacted grid services ability to advertise policies, to indicate what credentials are needed and to specify where to retrieve them from. Resources were shared with no implied previous interactions with

[4] The certificate chain has at its root Alice X509 End Entity Certificate issued by the University CA

entities interested in accessing them, and administrators involvement reduces to setting policies for access control and credential disclosure.

5 Architecture Overview and Implementation

This section presents an extension to the recently released Globus Toolkit version 4.0 (GT 4.0), which allows advanced access control mechanisms and policy based negotiations, as depicted in figure 2. One of our main design goals was easy integration with current grid services paradigms and, therefore, our extension is backwards compatible and provides a straightforward installation with almost no additional effort.

The GT 4.0 Authorization Framework [9] supports grid service pluggable Policy Decision Points (PDPs) [9]. A PDP intercepts client calls and is responsible for authorizing them. It must return an authorization decision which can be either "granted" or "denied". In addition, a chain of PDPs can be specified for authorization allowing a series of verifications to be performed. In this case, the final decision is the conjunction of all of them, that is, all of them must return "granted" in order to permit access. This introduces a set of limitations as configuring custom PDPs for each access policy forces the client to follow only one chain of requirements and be granted access only if all policies have been satisfied. We desire to support also disjunction in the authorization decision and in this way provide to a larger number of clients the possibility of accessing the resource through the satisfaction of one chain of policies from several (presumably) available with the resource. This boosts the authorization mechanism by allowing resources administrators to have a single point of configuration for all authorization policies and permitting access to clients holding different sets of attributes. Moreover, because clients are offered several options for authorization, they may be able to choose according to their preferences (expressed in their own policies) which credentials are to be disclosed (e.g., credentials available locally or revealing less sensitive information).

Due to the restricting support of GT 4.0 in terms of service PDPs decision aggregation, we have developed a custom Interceptor PDP responsible for client call filtering and checking of a successfully completed negotiation process while the policy requirements and proofs are handled outside the PDP through the integration of a Negotiation Module into the service functionality. Check figure 3 for an architectural overview.

The Interceptor PDP allows client calls to a service in case no extra requirement is needed for the requested operations. Otherwise, they are denied with a negotiation exception sent to the client, until a successful policy based trust negotiation is completed. The Interceptor PDP and the operations which can directly be accessed without negotiations are configured through a security descriptor pointed out in the service deployment descriptor.

If authorization is denied, the client can start a negotiation with the service, having his negotiation module contact the service negotiation module and requesting access. The service sends back the access policy for the operation requested and the client can answer disclosing the credentials specified in the service policy or sending its own access policy for those credentials.

A grid service describes the operations supported and their parameters through its WSDL file. At the WSDL level the service is identified as a port type having a name

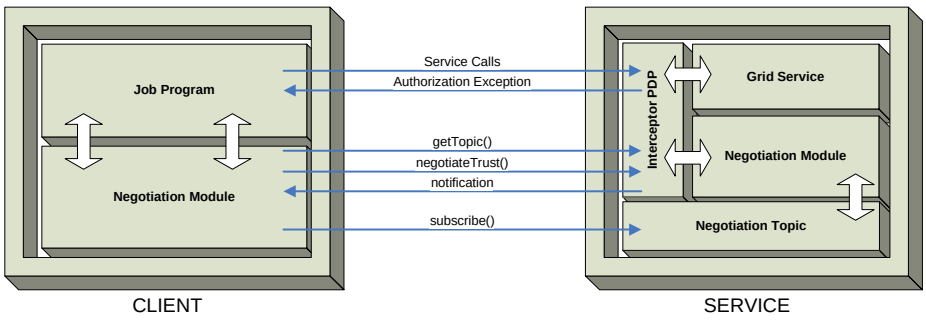


Fig. 3. Architecture Overview

and several operations, and using messages to carry input and output data. GT4.0 provides a useful feature in writing WSDL code for a grid service, the WSDLPreprocessor namespace which contains a tag "extends" permitting the inclusion of definitions of other port types in the current grid service definition. Using this feature it is possible to include the functionality of a service (giving that this functionality is also implemented in code) into another service. We have defined a WSDL file for a port type called TrustNegotiation, which describes messages targeted for trust negotiation and exposes two extra operations (getTopic and negotiateTrust) enabling the exchange of policies and proofs between clients and services. By extending the TrustNegotiation port and having the functionality of the two extra operations implemented in the Negotiation Module, a general grid service is enhanced with trust negotiation capabilities. We emphasize that in order to confer this functionality to a usual grid service we have to modify only its descriptors (that is, configuration) files.

The disclosure of a policy from one entity to another may result in a credential being fetched from a service which at its turn may result in another policy based trust negotiation process, making it difficult to predict when one request can finish its evaluation, and thus to estimate how long a client has to wait for the negotiation process to conclude. In order to overcome this problem we used the GT 4.0 support for WS-Notification [23] as a mechanism for asynchronous client information of the grid service decisions: policies required to be satisfied, proofs disclosed or authorization reached.

The WS-Notification family of specifications includes WS-BaseNotification [18] and WS-Topics [20], standardizing asynchronous communications between consumers and providers. WS-Topics specifies how topics can be organized and categorized as items of interest for subscription. We associate each trust negotiation (triggered by the client request for a service operation) with a topic of interest through which messages can be delivered to the client side. Consumers register themselves with a certain topic of interest, while providers deliver the notifications to the consumers, each time the topic has changed. Notification producers (in our case grid services) expose a subscribe operation through which, each consumer (either grid service or client) can register with a certain topic. Notification consumers expose a notify function that producers use to send notifications. The grid service use of notifications is configured also, through the service descriptors and their functionality is supported by the GT4.0 container. The

functionality behind topic assignment, management and client registration for service notifications is implemented in our Negotiation Topic Module (figure 3).

In our design, each client interested in negotiating trust calls the getTopic operation (exposed by the grid service through the extension of the TrustNegotiation port)in order to receive a unique namespace associated with a topic [20] and subscribes to it. Service side policies and disclosed credentials reach the client asynchronously as notifications, while client policies and proofs are pushed to the service through client invocation of the trustNegotiate operation (also exposed by extending the TrustNegotiation port).

The use of the GT 4.0 implementation of the notification paradigm imposes one limitation to the grid service, that of exposing its state as a "Resource" in conformity with the Web Service Resource Framework (WSRF) [19]. "Resources" are used for storing a web service state from one invocation to another. Thus the only requirement we impose for integrating our policy based architecture for grid service authorization is having the service use such a "Resource". The addition of a resource to a grid service is straightforward, with only minimal additions to its descriptor files and code (only a matter of having the service implement the interface *Resource* with no methods).

The client has a complementary functionality to that of a grid service. For client programming we have developed a jar file and an API, to facilitate easy integration of trust negotiation capabilities to client code. The client has to use one class (GridClientTrustNegotiation) for setting the grid service address and the namespace of the negotiation topic received. Also the client has to hide the service invocations by implementing one interface (SendWrapper), part of our API, for using the stub of the service with whom he wants to negotiate trust.

Service stubs mask the intricacies of communication with a service and can be generated automatically from the grid service WSDL file. In Java, stubs are represented as objects with methods for each operation exposed by the service. Since each stub is specifically generated for the service it targets to invoke, we have developed an interface (SendWrapper) abstracting the calls made to a trust negotiation enabled service stub (supporting our defined getTopic and negotiateTrust operations). The client has to implement this interface and one of its functions (sendMessage) to call the trustNegotiate operation on the stub. By implementing the same interface (SendWrapper), we have provided to the client an object/class he can use for automatic fetching of credentials from a CAS server.

On the client side a thread registers with the topic returned by the grid service and listens for notifications. SendWrappers are used on the client side for sending queries to the service with which the negotiation process is undergoing, or to third party services (e.g., CAS) for credential retrieval. The same client code can be used by the service to register for negotiation with other services or to retrieve credentials, as one service may be the client of another service with whom it negotiates trust.

Credentials used in our implementation may virtually be expressed under any kind of format (certificates, signed assertions, signed RDF statements) as the policies and the negotiation modules are decoupled of credential types. Currently we have support for X.509 v3 Certificates holding in extensions the owner attributes. In addition to this, we have integrated the use of proxy certificates which carry SAML Assertions [15] retrieved from a Community Authorization Service (CAS). A SAML Assertion indicates its issuer, the subject of the statement, a resource, and an action allowed on this re-

source. The client can wrap such a SAML Assertion in a proxy certificate and push it as a proof satisfying a policy disclosed during a trust negotiation process.

The architecture we propose is general enough to deal with different policy languages and policy evaluation procedures, which are implemented in the negotiation module of each entity. In our implementation we use a negotiation module based on the PEERTRUST project and language [6] with built-in support and verification of X.509 v3 certificates. The entire implemention is done in Java and uses an inference engine to reason over policies and credentials. We have tested our extension accessing different Grid services and automatically fetching credentials from a CAS server with positive results. Our implementation is distributed as a jar file which together with the small changes to the appropriate configuration files allows users and administrators to easily integrate it with current GT 4.0 grid services.

6 Related Work

In this section we provide a brief overview of related approaches to Grid authorization. The common characteristic of these authorization schemes is the replacement of identity based access with the usage of user attributes in the authorization process. In this regard, the major improvement comes from the increased number of entities able to be accommodated by a Grid environment released of the consistent mapping of client identities to user accounts. Nevertheless, despite its benefits for increased scalability, this approach still places on the client side the burden of appropriate certificate provision and retrieval for every accessed location. In addition, the authorization schemes presented further, tend to be client centered, with attribute requirements and their demonstrated possesion enforced only for the client side and no policy imposed to the service itself.

Virtual Organization Management Service (VOMS) [1] supports attribute based access control to the Globus Toolkit Resource Allocation Manager (GRAM) responsible for job execution. Clients retrieve pseudo certificates containing client attributes (groups and roles) from VOMS compliant servers and include them in a non-critical extension of a usual proxy certificate. Such proxy certificates are pushed to the resource together with the submitted jobs, and based on their contained attributes an authorization decision is made.

PERMIS [3] project has as its main goal the construction of an X.509 role based Privilege Management Infrastructure that could accommodate diverse role oriented scenarios. PERMIS consists of two subsystems: the privilege allocation subsystem which issues user X.509 Attribute Certificates and stores them in LDAP directories for later retrieval and the privilege verification subsystem which pulls the user certificates and the policies regarding user roles from a pre-configured list of LDAP directories (clients can also push certificates to the privilege verification subsystem).

AKENTI [16] uses distributed policy certificates signed by stakeholders from different domains in order to make a decision regarding access to a resource. Akenti uses an XML format for representing three types of certificates: *attribute certificates* binding an attribute-value pair to the principal of the certificate, *use-condition certificates* indicating lists of authoritative principals for user attributes and containing relational expressions of required user attributes for access rights and *policy certificates* consisting of trusted CAs and stakeholders issuing use-condition certificates and lists of URLs

from where use-condition and attribute certificates can be retrieved. Akenti engine authenticates clients based on their X.509 certificate, gathers in a pull model certificates available with the authenticated identity (attribute certificates) and those of the accessed resource (use-condition certificates) and makes a decision regarding user rights at the resource.

System for Privilege Management and Authorization (PRIMA) [10] manages and enforces privilege and policy statements expressed in X.509 Attribute Certificates. In PRIMA design, resource administrator and stakeholder issued certificates are revealed by the client to the resource Policy Enforcement Point (PEP). The PEP validates user attributes and verifies with the resource Policy Decision Point (PDP) if the issuers are authoritative for user's presented privileges. All acknowledged privileges are gathered by the PEP in a policy further presented to the PDP for verification against the configured access control policies. The PDP returns to the PEP an authorization decision and a set of recommendations (file access permissions, user quotas and network access [11]) for setting up a local account with support for the user validated privileges.

Similar to the above discussed authorization approaches, our architecture accommodates user attributes for access granting, but in addition offers solutions for client query of resource access policies and their compliance through negotiations and automatic credential fetching. Furthermore we allow clients to define and enforce their own policies for service authorization.

Another area of research is concerned with the standardization of policy representation. Currently there is an extensive effort in enabling eXtensible Access Control Markup Language (XACML) [4] usage for authorization management in Grid environments. XACML is an OASIS standard for specifying access control policies and the associated request/response formats. It allows use and definition of combining algorithms which provide a composite decision over policies governing the access requirements of a resource. Future versions of Globus Toolkit are expected to be delivered with an integrated XACML authorization engine. However, current versions of XACML are not yet expressive enough to deal with some of the requirements for the integration of our approach (e.g. delegation of authority).

7 Conclusions and Further Work

This paper described current authorization mechanisms in Grids together with their limitations. We described how support for policy-based negotiation can be integrated into a Grid infrastructure, allowing advanced access control and automatic credential fetching, in order to provide a solution to most of those limitations. We discussed how to easily integrate these extensions into the Globus Toolkit 4.0 and presented our current implementation.

Our future research will focus on online credential repositories in order to extend the automatic credential fetching capabilities. We have already integrated CAS servers and we plan to do the same with MyProxy repositories. Unfortunately, these two types of online credential repositories do not share a common interface (requiring an ad-hoc integration by means of wrappers) and do not yet allow us to store arbitrary credentials which would be desirable for more complex scenarios.

We also investigate policy representation in XACML and the possibility of relying on the SAML support for querying and exchanging policies [14].

In addition, we are planning to perform more experiments in order to study the performance loss induced by policy integration into current grid services. However, although we know in advance that access will be somewhat more expensive than based on current approaches, we are certain that it will pay off as many of the tasks that have to be done manually so far can then be done automatically. On the other hand if we consider that Grid is designed to enable resource sharing for large computational jobs, whose requirements can not be met locally, an increased setup time is less significant compared to the ability of dynamic satisfaction of resource authorization policies.

Acknowledgments

This research was partially supported by the Network of Excellence REWERSE (`http://rewerse.net`, IST-506779)

References

1. R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A. Gianoli, K. Lőrentey, and F. Spataro. Voms: An authorization system for virtual organizations. In *Proceedings of the 1st European Across Grids Conference*, Santiago de Compostela, Feb. 2003.
2. J. Basney, W. Nejdl, D. Olmedilla, V. Welch, and M. Winslett. Negotiating trust on the grid. In *2nd WWW Workshop on Semantics in P2P and Grid Computing*, New York, USA, may 2004.
3. D. Chadwick and O.Otenko. The permis x.509 role based privilege management infrastructure. In *7th ACM Symposium on Access Control Models and Technologies*, 2002.
4. eXtensible Access Control Markup Language. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.
5. S. Farrel and R. Housley. An internet attribute certificate profile for authorization, rfc3281.
6. R. Gavriloaie, W. Nejdl, D. Olmedilla, K. E. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *1st European Semantic Web Symposium (ESWS 2004)*, volume 3053 of *Lecture Notes in Computer Science*, pages 342–356, Heraklion, Crete, Greece, may 2004. Springer.
7. Globus toolkit 4.0. http://www.globus.org/toolkit/docs/4.0/.
8. Grid security infrastructure. http://www.globus.org/security/overview.html.
9. Gt 4.0 authorization framework. http://www.globus.org/toolkit/docs/4.0/security/authzframe/.
10. M. Lorch, D. Adams, D. Kafura, M. Koneni, A. Rathi, and S. Shah. The prima system for privilege management, authorization and enforcement in grid environments. In *Proceedings of the 4th Int. Workshop on Grid Computing - Grid 2003*, Phoenix, AZ, USA, Nov. 2003.
11. M. Lorch and D. G. Kafura. The prima grid authorization system. *J. Grid Comput.*, 2(3):279–298, 2004.
12. J. Novotny, S. Tuecke, and V. Welch. An online credential repository for the grid: MyProxy. In *Symposium on High Performance Distributed Computing*, San Francisco, Aug. 2001.
13. L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, 2002.
14. Saml 2.0 profile of xacml v2.0. http://docs.oasis-open.org/xacml/2_0/access_control-xacml-2.0-saml-profile-spec-os.pdf.

15. Security assertion markup language. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security.
16. M. R. Thompson, A. Essiari, and S. Mudumbai. Certificate-based authorization policy in a pki environment. *ACM Trans. Inf. Syst. Secur.*, 6(4):566–588, 2003.
17. S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. *Internet X.509 Public Key Infrastructure Proxy Certificate Profile*. http://www.ietf.org/internet-drafts/draft-ietf-pkix-proxy-10.txt, Dec. 2003.
18. Web services base notification. http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf.
19. Web services resource framework. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.
20. Web services topics. http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.pdf.
21. V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, and F. Siebenlist. X.509 proxy certificates for dynamic delegation. In *3rd Annual PKI R&D Workshop*, Apr. 2004.
22. W. H. Winsborough, K. E. Seamons, and V. E. Jones. Automated trust negotiation. DARPA Information Survivability Conference and Exposition, IEEE Press, Jan 2000.
23. Ws-notification. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn.

A Logic Based SLA Management Framework

Adrian Paschke[1], Jens Dietrich[2], Karsten Kuhla[3]

[1]Internet-based Information Systems, Technische Universität München
Paschke@in.tum.de
[2]Information Sciences & Technology, Massey University
J.B.Dietrich@massey.ac.nz
[3]FileAnts GmbH München
Kuhla@fileants.com

Abstract. Management, execution and maintenance of Service Level Agreements (SLAs) in the upcoming service oriented IT landscape need new levels of flexibility and automation not available with the current technology. In this paper we evolve a rule based approach to SLA representation and management which allows a clean separation of concerns, i.e. the contractual business logic are separated from the application logic. We make use of sophisticated, logic based knowledge representation (KR) concepts and combine adequate logical formalisms in one expressive logic based framework called **ContractLog.** ContractLog underpins a declarative rule based SLA (**RBSLA**) language with which to describe SLAs in a generic way as machine readable and executable contract specifications. Based on ContractLog and the RBSLA we have implemented a high level architecture for the automation of electronic contracts - a rule-based Service Level Management tool (**RBSLM**) capable of maintaining, monitoring and managing large amounts of complex contract rules.

1 Why declarative rule-based SLA representation?

Our studies of a vast number of SLAs currently used throughout the industry have revealed that today's prevailing contracts are plain natural language documents. Consequently, they must be manually provisioned and monitored, which is very expensive and slow, results in simplified SLA rules and is not applicable to a global distributed computing economy, where service providers will have to monitor and execute thousands of contracts. First basic automation approaches and recent commercial service management tools[1] directly encode the contractual rules in the application logic using standard programming languages such as Java or C++. The procedural control flow must be completely implemented and business logic, data access and computation are mixed together, i.e. the contract rules are buried implicitly in the application code. SLAs are therefore hard to maintain and can not be adapted to new requirements without extensive reimplementation efforts. Consequently, the upcoming service orientation based on services that are loosely coupled across heterogeneous, dynamic environments needs new ways of knowledge representation with a high degree of flexibility in order to efficiently manage, measure and continuously monitor and enforce complex SLAs. Examples which illustrate this assertion are *dynamic rules*, *dependent rules*, *graduated rules* or *normative rules with exceptions/violations*

[1] e.g. IBM Tivoli Service Level Advisor, HP OpenView, Remedy SLAs

of rights and obligations, in order to pick up a few examples which frequently occur in SLAs:

- **Graduated rules** are rule sets which e.g. specify graduated high/low ranges for certain SLA parameters so that it can be evaluated whether the measured values exceed, meet or fall below the defined service levels. They are often applied to derive graduate penalties or bonuses. Other examples are service intervals, e.g., **"***between 0 a.m. and 6 a.m. response time must be below 10 s, between 6 a.m. and 6 p.m. response time must be below 4 s …"* or exceptions such as *maintenance intervals*.
- **Dependent rules** are used to adapt the quality of service levels. For example an reparation service level must hold, if a primary service level was missed (for the first time / for the umpteen times), e.g.: *"If the **average availability** falls below 97 % then the **mean time to repair** the service must be less than 10 minutes."*
- **Dynamic rules** either already exist within the set of contract rules or are added dynamically at run time. They typically define special events or non regular changes in the contract environment, e.g. a rule which states that *there might be an unscheduled period of time which will be triggered by the customer. During this period the bandwidth must be doubled*.
- **Normative rules with violations and exceptions** define the rights and obligations each party has in the present contract state. Typically, these norms might change as a consequence of internal or external events / actions, e.g. an obligation which was not fulfilled in time and hence was violated, might raise another obligation or permission: "*A service provider is obliged to repair an unavailable service within 2 hours. If she fails to do so the customer is permitted to cancel the contract.*"

The code of pure procedural programming languages representing this type of rules would be cumbersome to write and maintain and would not be considered helpful in situations when flexibility and code economy are required to represent contractual logic. In this paper we describe a declarative approach to SLA representation and management using sophisticated, logic based knowledge representation (KR) concepts. We combine selected adequate logical formalisms in one expressive framework called **ContractLog.** It enables a clean separation of concerns by specifying contractual logic in a formal machine-readable and executable fashion and facilitates delegating service test details and computations to procedural object-oriented code (Java) and existing management and monitoring tools. ContractLog is extended by a declarative **r**ule **b**ased **SLA** language (**RBSLA**) in order to provide a compact and user-friendly SLA related rule syntax which facilitates rule interchange, serialization and tool support. Based on ContractLog and the higher-level RBSLA we have implemented a **r**ule **b**ased **s**ervice **l**evel **m**anagement (**RBSLM**) prototype as a test bed and proof of concept implementation. The essential advantages of our approach are:

1. Contract rules are separated from the service management application. This allows easier maintenance and management and facilitates contract arrangements which are adaptable to meet changes to service requirements dynamically with the indispensable minimum of service execution disruption at runtime, even possibly permitting coexistence of differentiated contract variants.
2. Rules can be automatically linked (rule chaining) and executed by rule engines in order to enforce complex business policies and individual contractual agreements.
3. Test-driven validation and verification methods can be applied to determine the correctness and completeness of contract specifications against user requirements [1] and large rule sets can be automatically checked for consistency. Additionally, explanatory reasoning chains provide means for debugging and explanation. [2]

4. Contract norms like rights and obligations can be enforced and contract/norm violations and exceptions can be (proactively) detected and treated via automated monitoring processes and hypothetical reasoning. [2]
5. Existing tools, secondary data storages and (business) object implementations might be (re)used by an intelligent combination of declarative and procedural programming.

The contribution of this paper is twofold. First it argues that in a dynamic service oriented business environment a declarative rule based representation of SLAs is superior to pure procedural implementations as it is used in common contract management tools and gives a proof of concept solution. Second it presents a multi-layered representation and management framework for SLAs based on adequate logical concepts and rule languages. The paper is organised as follows. We give an overview on our approach in section 2 and describe its main components, in particular the expressive logical ContractLog framework, the declarative XML based RBSLA language and the RBSLM tool in the sections 3 to 5. In section 6 we present a use case in order to illustrate the representation, monitoring and enforcement process. Finally, in section 7 we conclude with a short summary and some final remarks on the performance and usability of the presented rule-based SLA management approach.

2 Overview

Fig. 1 shows the general architecture of our **r**ule **b**ased **s**ervice **l**evel **m**anagement tool (**RBSLM**). The rule engine Mandarax (1) acts as execution component and inference engine for the formalized, logical contract rules. The rules are represented on basis of our expressive, logical **ContractLog** framework (2) and are imported over the Prova language extension into the internal knowledgebase of the rule engine. An additional declarative language format based on XML[2], called the **r**ule **b**ased **SLA** (**RBSLA**) language, is provided to facilitate machine-readability, reusability, rule interchange, serialisation, tool based editing and verification. A mapping is defined which transforms the RBSLA into executable ContractLog rules. The graphical user interface - the **Contract Manager** (4) - is used to write, edit and maintain the SLAs which are persistently stored in the contract base (3). The **repository** (5) con-

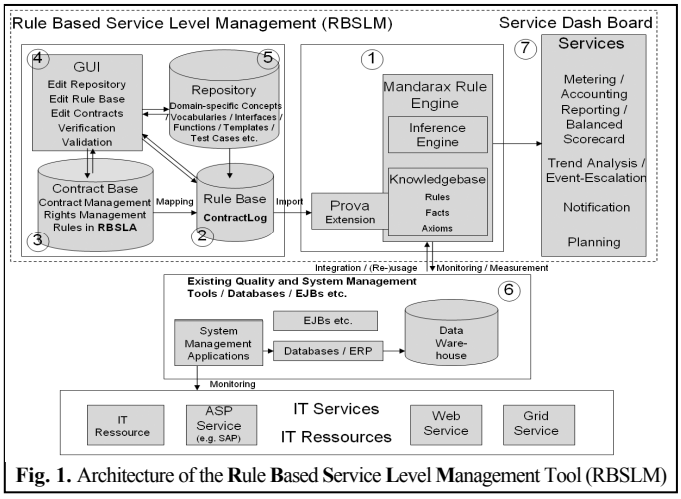


Fig. 1. Architecture of the **R**ule **B**ased **S**ervice **L**evel **M**anagement Tool (RBSLM)

[2] The semantic web rule standard RuleML (in particular object oriented RuleML)

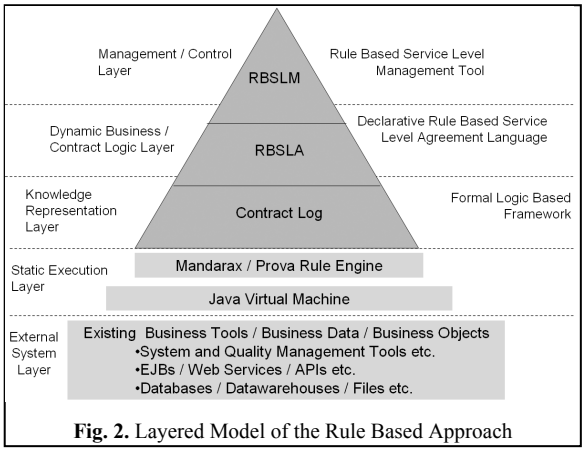


Fig. 2. Layered Model of the Rule Based Approach

tains typical rule templates and predefined SLA domain specific objects, built-in metrics and contract vocabularies (ontologies) which can be reused in the SLA specifications. During the enforcement and monitoring of the SLAs existing external business object implementations, quality and system management tools and external databases can be integrated (6). Finally, the **Service Dash Board** (7) visualizes the monitoring results and supports further SLM processes, e.g. reports on violated services, metering and accounting functionalities, notification services etc. Figure 2 summarizes the components of our rule based SLM approach in a layered model. In the following three sections we give a more detailed insight into the different layers described in figure 2 with a particular focus on the RBSLA serialization syntax.

3 ContractLog Framework

Table 1. Main logic concepts of ContractLog

Logic	Usage
Derivation rules (horn rules with NaF)	Deductive reasoning on contract rules.
Event-Condition-Action rules (ECA)	Active sensing and monitoring, event detection and "situative" behaviour by event-triggered actions.
Event Calculus (temporal reasoning a la "transaction logic")	Temporal reasoning about dynamic systems, e.g. effects of events on the contract state (fluents).
Defeasible / Courteous logic (priorities and conflicts)	Default rules and priority relations of rules. Facilitates conflict detection and resolution as well as revision/updating and modularity of rules.
Deontic Logic (contract norms) with norm violations and exceptions	Rights and obligations modelled as deontic contract norms and norm violations (contrary-to-duty obligations) and exceptions (condit. defeasible obligations).
Description logic (domain descriptions) / Description Logic Programs	Semantic domain descriptions (e.g. contract ontologies) in order to hold rules domain independent. Facilitates exchangeability and interpretation.
Object-oriented typed logic and procedural attachments	Typed terms restrict the search space. Procedural attachments integrate object oriented programming into declarative rules.

Table 1 summarizes the main concepts used in ContractLog. In the following we sketch the basic logical components. More details can be found in [2-4] and on our project site[5].

Typed Derivation Rules, Procedural Attachments and external Data Integration: Derivation rules based on horn logic supplemented with negation as failure (NaF) and

rule chaining enable a compact representation and a high degree of flexibility in automatically combining rules to form complex business policies and graduated dynamic contract rules. [6] On the other hand procedural logic as used in programming languages is highly optimized in solving computational problems and many existing business object implementations such as EJBs as well as existing system management and monitoring tools already provide useful functionalities which should be reused. Procedural attachments and the typed logic[3] used in ContractLog offer a clean way of integrating external programs into logic based rule execution paving the way for intelligently accessing or generating data for which the highest level of performance is needed and the logical component is minimal. This supports a smooth integration of facts managed by external systems (databases accessed via optimized query languages like SQL; systems, accessed using web services etc.) and avoids replication, because references are resolved at query time - which is crucial, as in SLA management we are facing a knowledge intensive domain which needs flexible data integration from multiple rapidly changing data sources, e.g. business data from data warehouses, system data from system management tools, process data from work flows, domain data from ontologies etc. Additionally, the tight integration with Java enables (re-)using existing business objects implementations such as EJBs and system management tools and allows for active sensing/monitoring and effecting via triggered actions in ECA rules.

ECA Rules: A key feature of a SLA monitoring system is its ability to actively detect and react to events. We implemented support for active ECA rules: *eca(T,E,C,A)*. Each term *T* (time), *E* (event), *C* (condition) and *A* (action) references to a derivation rule which implements the respective functionality of the term. The additional term *T* (time) is introduced to define monitoring intervals or schedules in order to control monitoring costs for each rule and to define the validity periods. Example:

eca(everyMinute, serviceUnavailable, notScheduledMaintanance , sendNotification)			
everyMinute(DT) ←	*serviceUnavailable(DT)* ←	*notScheduledMaintanance(DT)* ←	*endNotification(DT)* ←

Rule chaining combining derivation rules offers maximum flexibility to build complex functionalities, which can be referenced and reused in several ECA rules. More details on the ECA implementation can be found in [2-4].

Event Calculus: The Event Calculus (EC) [7] defines a model of change in which *events* happen at *time-points* and *initiate* and/or *terminate time-intervals* over which some *properties* (time-varying **fluents**) of the world hold. We implemented the classical logic formulations using horn clauses and made some extensions to the core set of axioms to represent derived fluents, delayed effects (e.g. validity periods and deadlines of norms), continuous changes (e.g. time-based counter) and epistemic knowledge (planned events for hypothetical reasoning) [2, 4]:

Classical Domain independent predicates/axioms	**ContractLog Extensions**
happens(E,T) event E happens at time point T	*valueAt(P,T,X)* parameter P has value X at time point T
initiates/terminates(E,F,T) E initiates/terminates fluent F	*planned(E,T)* event E is believed to happen at time point T
holdsAt(F,T) fluent F holds at time point T	*derivedFluent(F)* derived fluent F

[3] ContractLog supports typed variables and constants based on the object-oriented type system of Java or other OO typed systems.

The EC and ECA rules might be combined and used vice versa, e.g. fluents might be used in the condition parts of ECA rules or ECA rules might assert detected events to the EC knowledgebase. The EC models the effects of events on changeable SLA properties (e.g. deontic contract norms such as rights and obligations) and allows reasoning about the contract state at certain time points. Its rules define complex transaction logics with state changes similar to workflows. This is very useful for the representation of deontic contract norms and exceptions or violations of norms.

Deontic Logic with Norm Violations and Exceptions: Deontic Logic (DL) studies the logic of normative concepts such as obligation (O), permission (P) and prohibition (F). However, classical standard deontic logic (SDL) offers only a static picture of the relationships between co-existing norms and does not take into account the *effects of events on the given norms, temporal notions* and *dependencies between norms*, e.g. violations of norms or exceptions. Another limitation is the inability to express *personalized statements*. In real world applications deontic norms refer to an explicit concept of an agent. We extended the concepts of DL with a role-based model and integrated it in the Event Calculus implementation in order to model the effects of events/actions on deontic norms [2]. This enables the definition of institutional power assignment rules (e.g. empowerment rules) for creating institutional facts which are initiated by a certain event and hold until another event terminates them. Further, we can define complex dependencies between norms in workflow like settings which exactly define the actual contract state and all possible state transitions and which allow representing norm violations and their conditional secondary norms, e.g. contry-to-duty (CTD) obligations as well as exceptions of (defeasible prima facie) norms.. A deontic norm consists of the normative concept (*norm N*), the subject (*S*) to which the norm pertains, the object (*O*) on which the action is performed and the action (*A*) itself. We represent a role based deontic norm $N_{s,o}A$ as an EC fluent: *norm(S, O, A)*, e.g. *inititates(e1,* ***permit(s,o,a)****, t1)*. We implemented typical DL inference axioms in ContractLog such as $O_{s,o}A \rightarrow P_{s,o}A$ or $F_{s,o}A \rightarrow W_{s,o}A$ etc. and further rules to deal with deontic conflicts (e.g. $P_{s,o}A \wedge F_{s,o}A$), exceptions (E) ($E \rightarrow O_{s,o}\neg A$), violations (V) of deontic norms (e.g. $O_{s,o}A \wedge \neg A$) and contrary-to-duty (CTD) obligations ($V \rightarrow O_{s,o}CTD$) or other "reparational" norms. In particular *derived fluents* and *delayed effects* (with *trajectories and parameters* [2]) offer the possibility to define norms with deadline-based validity periods, (time-based) violations of contract norms and conditional secondary norms e.g., contrary-to-duty (CTD) obligations. A typical example which can be found in many SLAs is a primary obligation which must be fulfilled in a certain period, but if it is not fulfilled in time, then the norm is violated and a certain "reparational" norm is in force, e.g., a secondary obligation to pay a penalty or a permission to cancel the contract etc. [2-4]

Remark. DL is plagued by a large number of paradoxes. We are aware of this. However, because our solution is based on temporal event logic we often can avoid such conflicts, e.g. a situation where a violated obligation and a CTD obligation of the violated obligation are true at the same time is avoided by terminating the violated obligation so that only the consequences of the violation (CTD obligation) are in effect. Other examples are **defeasible prima facie obligations** ($O_{s,o}A$) which are subject to exceptions ($E \rightarrow O_{s,o}\neg A$) and lead to contradictions, i.e. $O_{s,o}\neg A$ and $O_{s,o}A$ can be derived at the same time. We terminate the general obligations in case of an exception and ini-

tiate the conditional more specific obligation till the end of the exceptional situation. After this point the exception norm is terminated and we re-initiate the initial "default" obligation. Note that we can also represent norms which hold initially via the *initially* axiom in order to simulate "non-temporal" norms. A third way is to represent conflicts as **defeasible deontic rules** with defined priorities (*overrides*) between conflicting norms, i.e. we weaken the notion of implication in such a way that the counterintuitive sentences are no longer derived (see. defeasible logic).

Defeasible Logic: We adapt two basic concepts in ContractLog to solve conflicting rules (e.g. conflicting positive and negative information) and to represent rule precedences: Nute's defeasible logic (DfL) [8] and Grosof´s Generalized Courteous Logic Programs (GCLP) . There are four kinds of knowledge in DfL: *strict rules, defeasible rules, defeaters and priority relations*. We base our implementation on a meta-program [9] to translate defeasible theories into logic programs and extended it to support priority relations *r1>r2 : overrides(r1,r2)* and conflict relations in order to define conflicting rules not just between positive and negative literals, but also between arbitrary conflicting literals. Example:

Rule1 "discount": All gold customers get 10 percent discount."
Rule2 "nodiscount": Customers who have not paid get no discount."
ContractLog DfL: … overrides(discount, nodiscount) … // rule 1 overrides rule 2

GCLP is based on concepts from DfL. It additionally implements a so called *Mutex* to handle arbitrary conflicting literals. We use DfL to handle conflicting and incomplete knowledge and GLCP for prioritisation of rules. A detailed formulation of our implementation can be found in [4].

Description Logics: Inspired by recent approaches to combine description logics and logic programming [10] we have implemented support for RDF/RDFS/OWL descriptions to be used in ContractLog rules. At the core of our approach is a mapping from RDF triples (constructed from RDF/XML files via a parser) to logical facts: RDF triple:*subject predicate object* → LP Fact: *predicate(subject, object)*, e.g.:

$a : C$, i.e., the individual *a* is an instance of the class *C*: *type(a,C)*

$< a, b >: P$, i.e., the individual *a* is related to the individual *b* via the property *P: property(P,a,b)*

On top of these facts we have implemented a rule-based inference layer and a class and instance mapping[4] to answer typical DL queries (RDFS and OWL Lite/DL inference) such as class-instance membership queries, class subsumption queries, class hierarchy queries etc. This enables reasoning over large scale DL ontologies and it provides access to ontological definitions for vocabulary primitives (e.g. properties, class variables and individual constants) to be used in LP rules. In addition to the existing Java type system, we allow domain independent logical objects in rules to be typed with external ontologies (taxonomical class hierarchies) represented in RDF, RDFS or OWL.

[4] to avoid backward-reasoning loops in the inference algorithms

4 RBSLA language

Real usage of a formal representation language which is usable by others than its inventors immediately makes rigorous demands on the syntax: declarative syntax, comprehension, readability and usability of the language by users, compact representation, exchangeability with other formats, means for serialization, tool support in writing and parsing rules etc. The **r**ule **b**ased **SLA** language (**RBSLA**) tries to address these issues. It stays close to the emerging XML-based Rule Markup language (RuleML) in order to address interoperability with other rule languages and tool support. Therefore, it adapts and extends RuleML to the needs of the SLA domain.

RuleML is a standardization initiative with the goal of creating an open, vendor neutral XML/RDF-based rule language. The initiative develops a modular specification and transformations via XSLT from and to other rule standards/systems. RuleML arranges rule types in a hierarchical structure comprising reaction rules, transformation rules, derivation rules, integrity constraints, facts and queries. Since the object oriented RuleML (OO RuleML) specification 0.85 it adds further concepts from the object-oriented knowledge representation domain namely user-level roles, URI grounding and term typing and offers first ideas to prioritise rules with quantitative or qualitative priorities. However, the latest version 0.88 is still mostly limited to deduction rules, facts and queries. Currently, reaction rules have not been specified in RuleML and other key components needed to efficiently represent SLAs such as procedural attachments on external programs, complex event processing and state changes as well as normative concepts and violations to norms are missing; in order to pick up a few examples. As such improvements must be made. RBSLA therefore adds the following aspects to RuleML:

- *Typed Logic and Procedural Attachments*
- *External Data Integration*
- *Event Condition Action Rules with Sensing, Monitoring and Effecting*
- *(Situated) Update Primitives*
- *Complex Event Processing and State Changes (Fluents)*
- *Deontic Norms and Norm Violations and Exceptions*
- *Defeasible Rules and Rule Priorities*
- *Built-Ins, Aggregate and Compare Operators, Lists*
- *If-Then-Else-Syntax and SLA Domain Specific Syntax*

It is very important to note, that serialization of RBSLA in XML using RuleML does not require any new constructs, i.e., it can be done by using existing RuleML features. However, for the reason of brevity and readability RBSLA introduces apposite abbreviations for the prime constructs needed in SLA representation. A XSLT transformation might be applied normalizing the syntax into the usual RuleML syntax. We first present the abbreviated, compact RBSLA syntax and we will give an example of the normalization afterwards. Lack of space prevents us from giving an extensive description of the XML Schema representation of the RBSLA model. More details can be found in [11] and the latest version can be downloaded from our website [5]. Here, we blind out some details such as optional “oids” etc. and sometimes use a more compact DTD notation to present the concrete syntax.

Typed Logic: Logical terms (variables (Var), individuals (Ind) and complex terms (Cterm)) are either un-typed or typed. RBSLA supports the following type systems and data types: ***java:type , rdf:type , rdfs:type , owl:type , xsi:type , sql:type***.

Example:
```
<Var java:type="java.lang.Integer">1234</Var>
<Ind xsi:type="xs:nonNegativeInteger">12</Ind>
<Var rdfs:type="rbsla#Provider>Service Provider</Var>
```

Values of primitive data types such as integer, string, decimal, float, date, time etc. can be interchanged between the different type systems, i.e. they are unified and evaluated against each other. We therefore extend the unification process so that the following rules apply:

Untyped-Typed Unification:
1. The un-typed query variable assumes the type of the typed target variable or constant (individual)

Variable-Variable Unification:
2. If the query and the target variable are not assignable, the unification fails otherwise it succeeds
3. If the query variable belongs to a subclass of the class of the target variable, the query variable assumes the type of the target variable.
4. If the query variable belongs to a superclass of the class of the target variable or is of the same class, the query variable retains its class

Variable-Constant Unification:
5. If a variable is unified with a constant (individual) of its superclass, the unification fails otherwise if the type of the constant is the same or a sub-type of the variable it succeeds and the variable becomes instantiated.

Constant-Constant Unification:
6. The type of term from the head of the fact or rule is the same as or inherits from the type of term from the body of the rule or query

Procedural Attachments: $O_m[p^1..p^n] \rightarrow [r^1..r^n]$. Here, O denotes an object or a class which is also an object, m denotes a method invocation, $p^1..p^n$ the parameters and $r^1..r^n$ the list of result objects which might also be a Boolean true or false value. The serialization in RuleML extends complex terms: Cterm(Ctor | Attachment). The first element of an ***<Attachment>*** is either a link on a qualified Java class, a variable bound to a Java object/class instance or a nested complex term which itself constructs/returns an object. The second element specifies the method call: Attachment((Ind|Var|Cterm), Ind). The parameters for the method invocation are the subsequent elements under the Cterm element, which are defined after an attachment. RBSLA supports Java (via reflection), e.g.: **java.lang.Integer$_{parseInt}$[1234]→Integer(1234)**

```
<Cterm java:type="java.lang.Integer">                    // (types are optional)
        <Attachment>
                <Ind java:type="java.lang.Class">java.lang.Integer</Ind>
                <Ind java:type="java.lang.reflect.Method">parseInt</Ind>
        </Attachment>
        <Ind java:type="java.lang.String">1234</Ind>
</Cterm>
```

The result of a method invocation finally replaces the complex term and is used in the further derivation process. Results can be bound to variables via the *Equal* element:

```
<Equal>
        <Var> Varible </Var>
        <Cterm> … Attachment … </Cterm>
</Equal>
```

External Data Integration: RBSLA supports a smooth integration of facts managed by external databases in particular SQL databases or XML/RDF files. It therefore provides different built-in predicates such as **<Location>** (database location with username and password), **<Select>** (SQL select query), **<XML>** (construct a DOM tree from a XML file) etc. to access and query the database in order to reuse the result as facts in a knowledge system.

Event Condition Action Rules: An ECA rule **<Eca>** is a rule with four terms: **<(Time?,Event?,Condition?,Action)>**. Each term defines a reference **<Ref>** on a derivation rule (or a fact) which implements the respective functionality of the ECA term. This offers maximum flexibility. Logical rule chaining between derivation rules facilitates the implementation of complex functionalities which can be referenced and reused in several ECA rules and procedural attachments might be applied for active sensing, monitoring and triggering actions. A reference is semantically interpreted as a RuleML query (conclusion less derivation rule). Therefore, the only semantics we add is that of the ECA paradigm, which executes the ECA rule in a forward directional manner, i.e., it proceeds with the next rule term iff the query on the currently referenced derivation rule succeeds.

Example:
```
<Eca>
        <Time><Ref><Ind>everyMinute</Ind></Ref></Time>
        <Event><Ref><Ind>serviceUnavailable</Ind></Ref></Event>
        <Condition><Ref><Ind>notMaintenance</Ind></Ref></Condition>
        <Action><Ref><Ind>sendNotification</Ind></Ref></Action>
 </Eca>

<Implies>                                   // referenced time derivation rule
        <Atom><Rel>... respective time function ... </Rel></Atom>     //body
        <Atom><Rel>everyMinute</Rel></Atom                            //head
</Implies>
<Implies>                                   // referenced event derivation rule
    [...]
```

Situated Update Primitives: Update primitives change the state of a knowledge system. RBSLA supports primitives to *add* (**<Assert>**) and *delete* (**<Retract>**; **<RetractAll>**) facts and rules. Typically, these primitives are applied in ECA rules (a la transaction logic).

Complex Event Processing and State Changes (Fluents): RBSLA supports complex event processing and temporal reasoning about events/actions and their effects on the internal state of the knowledge system (a la event calculus). Therefore, it defines the following elements:

- **Fluent:** <!Element Fluent(Ind|Var|Cterm)>
- **Parameters:** <!Element Parameter(Ind|Var|Cterm)>
- **Persistent Events:** <!Element Happens((Event|Action),Time)>
- **Believed/Planned Events:** <!Element Planned((Event|Action),Time)>
- **Effects of events:**
    - <!Element Initially(Fluent)>
    - <!Element Initiates((Event|Action), Fluent, Time)>
    - <!Element Terminates((Event|Action), Fluent, Time)>
- **Queries:**
    - <!Element HoldsAt(Fluent, Time)>
    - <!Element ValueAt(Parameter, Time, (Ind|Var|Cterm))>

Fluents or parameters might be used in addition to individuals, variables or complex terms in rules: (Ind|Var|Cterm|Plex|Fluent|Parameter). By default fluents do not hold, but can be defined as initially true.

Deontic Norms and Norm Violations: Deontic Norms such as obligations, permissions or prohibitions are defined as personalized, time-varying fluents: norm(S,O,A)

```
<Oblige><Ind>Subject</><Ind>Object</><Action><Ind>Action</></></Oblige>
<Permit><Ind>Subject</><Ind>Object</><Action><Ind>Action</></></Permit>
<Forbid><Ind>Subject</><Ind>Object</><Action><Ind>Action</></></Forbid>
<Waived><Ind>Subject</><Ind>Object</><Action><Ind>Action</></></Waived>
```

RBSLA defines a special violation event which happens if a norm is violated, e.g. an obligation which is not fulfilled in time: **<Violation>**<Ind>*Violation*</>**</Violation>**

Defeasible Rules and Rule Priorities: Beside strict rules which a represented as normal derivation rules "*head* → *body*" (<Implies>) RBSLA supports defeasible rules "*body* => *head*" and therefore introduces the new rule element **<Defeasible>**. Although, incompatible and conflicting literals between rules in general can be expressed as specializations of RuleML integrity constraints, RBSLA introduces a new **<Mutex>** element (for mutually exclusive, derived von GCLP), e.g.:

```
<Mutex>
        <Atom>   <Rel>discount</Rel> <Var>X</Var>       </Atom>
        <Atom>   <Rel>discount</Rel> <Var>Y</Var>       </Atom>
        <Atom><Cond>
                <Neg><Equal> <Var>X</Var> <Var>Y</Var> </Equal></Neg>
        </Cond></Atom>
</Mutex>
```

An **<Overrides>** element defines the priority of rules or rule sets / modules:

```
<Overrides>
        <Ref><Ind>rule1</Ind></Ref>
        <Ref><Ind>rule2</Ind></Ref>
</Overrides>
```

Built-Ins, Aggregate and Compare Operators, Lists: RBSLA provides different useful built-in functions and predicates to effectively work with variables, numbers, strings, date and time values, lists etc. Here we can only list the interesting ones:

Variables:
- **<Bound>, <Free>**: Given an input variable, test whether it is instantiated or not
- **<Type>**: Given an input variable, return the type name.

Numbers:
- **<Add|Sub|Mult|Div|Mod|Max|Min|Abs>**: Compute two numeric values and return the result.

Strings:
- **"\"** : separator "\" to allow special characters in string such as \n \r \t etc.
- **<Concat>, <Parse>, <Tokenize>** etc.

Date and Time:
- **<Date> <Ind>Year</><Ind>Month</><Ind>Day</> </Date>**
- **<Time><Ind>Hour</>, <Ind>Min</>, <Ind>Sec</> </Time>**
- **<DateTime> ((<Date>,<Time>) | (<Ind> Epoch Value in millis </Ind>))</DateTime>**
- **<TimeSpan>, <Intervall>**
- **<Compare>, <Less>, <Equal>, <More>, <Add>, <Sub>** etc.

Lists:

- **<Plex><Var1/><Var2/>...<VarN/></Plex>**: List of the form [Var1 .. VarN]
- **<Plex><Var>Head</Var><repo><Var>Rest</Var></repo></Plex>:** List of the form [Head|Rest]
- **<Member>**: Test whether an object is member of a list
- **<Element_At>**: Return the object at position X in a list
- **<Append>,<Delete>**: Add/Delete an element/list. The result is the concatenated list.
- **<Head>, <Tail>**: Return the head / the tail of a list
- **<First>, <Last>**: Return the first / last element of a list
- **<Size>**: Return the size of a list

Aggregations:

- **<Sum>,<Max>,<Min>,<Mean>**: Calculate the aggregation of a list and return the result.

Comparison:

- **<Equal>, <LessEqual>, <Less>, <More>, <MoreEqual>, <Between>**

If-Then-Else-Syntax and SLA Domain Specific Syntax: In order to make programming in RBSLA and specification of SLAs more efficient and easier, RBSLA provides an additional If-Then-Else Syntax for rules:

<If> <Atom> ... Body ... </Atom> **</If>**
<Then> <Atom>Head</Atom> **</Then>**
<Else> <Atom>Head of corresponding Naf rule </Atom> **</Else>**

Whilst the if-then part of such a rule maps to a normal RuleML derivation rule *(<Implies>*) the else part maps to a corresponding negated (with Negation as Failure NaF) rule. RBSLA provides direct serialization of reusable SLA metrics **<Metric>**. SLA metrics are used to measure the performance characteristics. They are either retrieved directly from the managed resources such as servers, middleware or instrumented applications or are created by aggregating such *direct metrics* into higher level *composite metrics.* Typical examples of direct metrics are the MIB variables of the IETF SMI such as *number of invocations, system uptime, outage period* etc. which are collected via measurement directives such as management interfaces, protocol messages, URIs etc. Composite metrics use a specific function averaging one or more metrics over a specific amount of time, e.g. *average availability*, or breaking them down according to certain criteria, e.g. *minimum throughput, maximum response time,* etc. Direct metrics contain measurement directives implemented as attachments:

```
<Metric type="rbsla#DirectMetric">
    <Rel><Ind>Metric Name</Ind></Rel>
    <Cterm><Attachment> [...] </Attachment> </Cterm>
</Metric>
```

Composite metrics use the built-in RBSLA aggregate operators over list structures which temporarily store the measurement results of direct metrics or they shift these expensive computations to highly optimized query languages such as SQL. Metrics might be (re)used in rules, e.g. in the event declaration of an ECA rule:

```
<Event><Ref><Ind>serviceUnavailable</Ind></Ref></Event>
<Implies>                                   // referenced rule
    <Less><Metric>...AverageAvailability ...</Metric><Ind >0.98</Ind></Less>
    <Atom><Rel>serviceUnavailable</Rel></Atom>
</Implies>
```

As mentioned before the presented RBSLA syntax can be normalized to the usual RuleML syntax via XSLT, for example:

```
<Oblige>
        <Ind>Service Provider</Ind>
        <Ind>Service Consumer</Ind>
        <Action><Ind>payPenalty</Ind></Atom>
</Oblige>
Maps to: <Cterm rdfs:type="rbsla#Obligation">                    (type are optional)
                <Ctor>oblige</Ctor>
                <Ind rdfs:type="rbsla#Provider">Service Provider</Ind>
                <Ind rdfs:type="rbsla#Consumer">Service Consumer</Ind>
                <Ind rds:type="rbsla#Action>payPenalty</Ind>
        </Cterm>
```

RBSLA comprises an additional ontology (based on RDFS) which describes the SLA domain vocabulary and can be used to semantically annotate and type used objects. This restricts search space to clauses where the type restrictions are fulfilled and makes the derivation process more efficient.

We have implemented a transformation process based on Java which maps the RBSLA rules into the basic logical ContractLog rules (Prova/Prolog syntax). During the transformation process we apply automated "refactorings" on the rules in order to improve the execution efficiency: loops, order of rules / order of prerequisites in rules might matter or performance critical issues like extensive use of negation as failure or cuts might be applied. For example we do some narrowing on multiple rules which share the same set of prerequisites in order to reduce redundancies ($A_1,..,A_N \rightarrow B$ and $A_1,..,A_N \rightarrow C$ becomes $A_1,.., A_N \rightarrow A$; $A \rightarrow B$; $A \rightarrow C$). Other examples are removing disjunctions in the prerequisites (replacing $A \vee B \rightarrow C$ by two new rules $A \rightarrow C$ and $B \rightarrow C$), remove conjunctions from rule heads (transform $B \rightarrow (H \wedge H')$ via Lloyd-Topor transformation into two rules $B \rightarrow H$ and $B \rightarrow H'$), remove function symbols from rule heads or reducing typing information in rule chains etc.

To sum up the additional RBSLA layer solves the demands stated at the beginning of this section, in particular compatibility with other languages via transformations and a declarative, compact and readable rule representation. Additionally it provides means for refactoring and validation of rules during transformation into ContractLog.

5 RBSLM tool

The RBSLM tool splits into the **Contract Manager** (CM) and the **Service Dashboard** (SD). The CM is used to manage, write, maintain and update SLA rules. The SD visualizes the monitoring and enforcement process in the contract life cycle and supports further SLM processes. We first describe the CM.

The CM provides a basic editor for ContractLog rules and a repository approach which represents the structure and meta-data of a knowledge base. It supports two roles: the *business practitioner* and the *rule experts*. The rule experts have a background in logic and are therefore responsible for the basic design of contract rules. They fill the repository with needed and reusable domain specific measurement, monitoring and computing functions, interface implementations to existing databases or system tools, references on existing business objects (e.g. EJBs) as well as rule templates and other domain specific concepts (e.g. contract vocabularies). Addition-

ally, they specify test cases together with certain test data to be used for verification and validation of contract rules in order to ensure the correct usage and a high-quality of SLA rule sets [1]. The business practitioners are involved in the daily business. They make use of the predefined templates to write and maintain the contract rules. They do not need to know any implementation details of used functions, objects or interfaces nor do they need to have a complete overview of all the rules in the contract system, meaning they do not know what the effect is on existing rules when a rule is added or changed. They just use the GUI to adapt the templates and build rules by clicking together the needed functionality. The test cases safeguard the authoring of rules and allow validation of complete rule sets and contracts to detect anomalies like inconsistencies, incompleteness or redundancies referring to the intended goals.

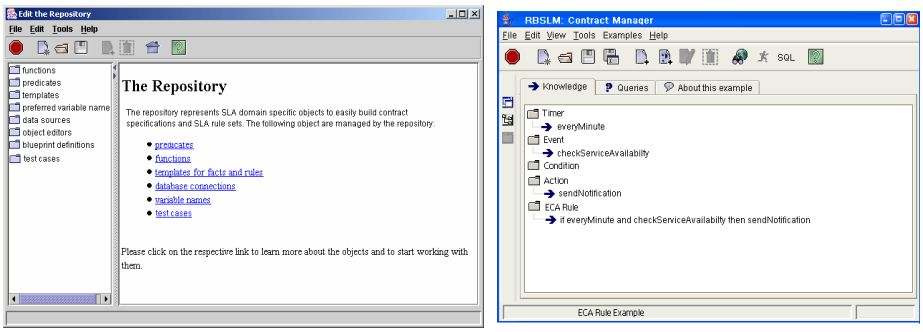


Fig. 3. The ContractManager GUI

Whilst the Contract Manager is used to manage the SLAs and their rules, the Service Dashboard visualizes status information and metrics during the monitoring and execution process. We provide different and adaptable visualisation views in order to satisfy the needs of different user roles e.g. for the customer advisor to face customer complains and problems, for the accountancy to forecast fees and recourse receivables and for the administrator to give detailed information to fix arising problems. Similar to the process of deriving quality metrics from base data to verifying contractual rules, the logic engine can also be used to get meaningful quantities for each of these parties. Predefined parsers and chart formations enable the user to present the information in an adequate way. New user views can be easily plugged into the framework dynamically through the class loader. The underlying Model-View-Controller notifies these views if data are changed in the knowledge base (observer pattern) and triggers the visualization process.

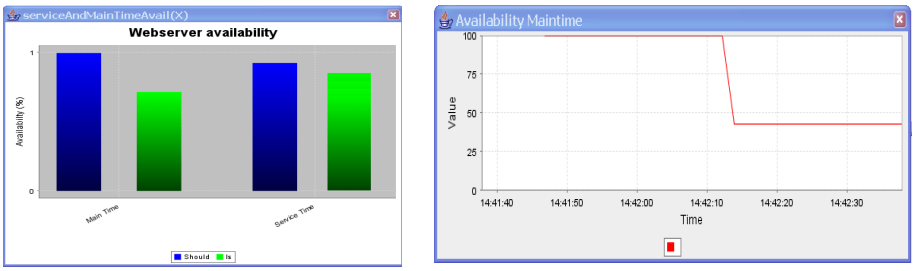


Fig. 4. Service Dashboard with different visualization views

6 Use Case

We now want to illustrate the SLA representation, monitoring and enforcement process. We therefore use the following example agreement:

*The service availability will be measured every t_{check} by a ping on the service. If the service is unavailable, the SP is **obliged** to restore it within $t_{deadline}$. If SP fails to restore the service in $t_{deadline}$ (violation), SC is **permitted** to cancel the contract.*"

RBSLA representation

```
<Eca>                                                    // ECA rule monitoring the service availability
      <Time><Ref><Ind> tcheck</Ind></Ref></Time>
      <Event><Ref><Ind>unavailable</Ind></Ref></Time>
      <Action><Ref><Ind>assertEvent</Ind></Ref></Action>
</Eca>

<Rule>                                                   // referenced time derivation rule
      <If> <Intervall>… [monitoring schedule] … </Intervall>  </If>   // body
      <Then> <Atom><Rel> tcheck</Rel></Atom>  </Then>     // head
</Rule>

<Rule>                                                   // referenced event derivation rule
      <If> <Metric>… [test availability] ... </Metric>   </If>   // body
      <Then> <Atom><Rel> unavailable</Rel></Atom>  </Then>     // head
</Rule>

                  […]

<Initiates>                                              // unavailable event initiates primary obligation
          <Event><Ind>unavailable</Ind></Event>
          <Oblige> … [obligation norm] … </Oblige>
</Initiates>
                  […]
<Rule>                                                   // if deadline elapsed then raise violation event
        <If><ValueAt><Parameter>deadline</Parameter><Time> tdeadline</Time><Ind>0</Ind></ValueAt></If>
        <Then><Assert><Violation><Ind>elapsed</Ind></Violation></Then>
</Rule>

<Initiates>                                              // violation event initiates reparation permission norm
          <Violation></Ind>elapsed</Ind></Violation>
            <Permit> … [cancel contract]  </Permit>
</Initiates>
                  [...]
```

ContractLog representation

```
eca(everyTcheck ,  serviceUnavailable , assertUnavailable)          // ECA rule
Time:    everyTcheck (DT) ← < interval function for  tcheck,>      // referenced derivation rule
Event:   serviceUnavailable(DT) ← not ping(service)                // referenced derivation rules
Action:  assertUnavailable(DT) ← assert(happens(unavailable,T)     // referenced derivation rules

initiates(unavailable, oblige(SP, Service, start()),T)     // defines the primary obligation initiated by an certain event
 terminates(available, oblige(SP, Service, start()),T)  //  defines the event which normally terminates the obligation
trajectory(oblige(SP,Service,start()),T1,deadline,T2,(T2 - T1))  // defines the period in which the norm must be fulfilled
happens(elapsed,T) ← valueAt(deadline,T, tdeadline)        // defines the violation of the obligation norm
initiates(elapsed, permit(SC, Contract, cancel()),T)  // initiates the derived permission to cancel the contract
```

The RBSLA representation is translated via the RBSLA compiler (transformation) into the ContractLog representation. The Prova engine (based on Mandarax rule engine) together with the ContractLog extensions execute and monitor the contract.

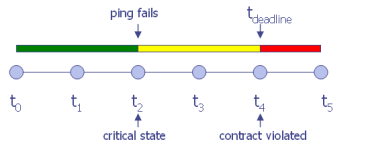


Fig. 5. Contract tracking

We assume that the service becomes unavailable at time t_2 and is restarted by the service provider at time t_5 which is after $t_{deadline}$. The ECA meta interpreter of the ContractLog framework monitors the ECA rule. Every t = t_{check} it pings the service via a procedural attachment and asserts the respective event to the

knowledgebase if the service is unavailable. This leads to the conclusions illustrated in fig. 5. The derived status information can be used to feed periodical reports, enforce rights and obligations or visualize monitoring results on quality aspects in the Service Dashboard.

7 Conclusion and Key Findings

In this paper we have described our declarative rule based approach to SLA representation and management. We have given an insight into the logical core, the ContractLog framework which underpins the declarative RBSLA language. Based on this we have implemented a prototypical rule based service level management tool (RBSLM). In contrast to conventional pure procedural programming approaches our declarative logic based approach simplifies maintenance, management and execution of SLA rules and allows easy combination and revision of rule sets to build sophisticated and graduated contract agreements, which are more suitable in a dynamic service oriented environment than the actually used, simplified rules. Although the framework described in this paper is still a proof of concept implementation, we have attempted to gather some data on its performance and usability. The important reasoning task in SLA monitoring and enforcement is query answering which is known to be only semi-decidable. However, the combination of highly optimized OO programming and database techniques as well as the use of adequate logical formalism implemented on the basis of horn logic and meta-programming techniques makes possible the high efficiency of the framework although it provides rich expressiveness. Usability analyses and qualitative comparisons with other representation approaches such as WSLA have revealed the higher flexibility and automation of our rule based approach.

References

1. Dietrich, J. and A. Paschke. *On the Test-Driven Development and Validation of Business Rules*. in *ISTA05, Massey, New Zealand*. 2005.
2. Paschke, A. and M. Bichler. *SLA Representation, Management and Enforcement - Combining Event Calculus, Deontic Logic, Horn Logic and Event Condition Action Rules*. in *EEE05*. 2005. Hong Kong, China.
3. Paschke, A., M. Bichler, and J. Dietrich. *ContractLog: An Approach to Rule Based Monitoring and Execution of Service Level Agreements*. in *RuleML 2005*. Galway, Ireland.
4. Paschke, A., *ContractLog - A Logic Framework for SLA Representation, Management and Enforcement*. 7/2004, IBIS, TUM, Technical Report.
5. Paschke, A., *RBSLA: Rule-based SLA, http://ibis.in.tum.de/staff/paschke/rbsla/index.htm*..
6. Paschke, A., *Rule Based SLA Management - A rule based approach on automated IT service management (in german language)*. 6/2004, IBIS, TUM, Working Paper.
7. Kowalski, R.A. and M.J. Sergot, *A logic-based calculus of events*. New Generation Computing, 1986. 4: p. 67-95.
8. Nute, D., *Defeasible Logic*, in *Handbook of Logic in Artificial Intelligence and Logic Programming Vol. 3*, D.M. Gabbay, C.J. Hogger, and J.A. Robinson, Editors. 1994, Oxford University Press.
9. Antoniou, G., et al. *A flexible framework for defeasible logics*. in *AAAI-2000*. 2000.
10. Grosof, B.N., et al. *Description Logic Programs: Combining Logic Programs with Description Logic*. in *WWW03*. 2003: ACM.
11. Paschke, A., *RBSLA: A Rule Based Service Level Agreements Language*. 8/2004, IBIS, TUM, Technical Report.

Real-world Trust Policies

Vinicius da S. Almendra[1], Daniel Schwabe[1]

(1) Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio
Rio de Janeiro - Brazil
{almendra,dschwabe}@inf.puc-rio.br

Abstract. One of the most important problems on the semantic web area is the one of trust. The growing exchange of semantic web data raises the need of policies that allow filtering out untrustworthy information. It is necessary, however, to model adequately the concept of trustworthiness, otherwise one may end up with operational trust measures that lack a clear meaning. It is also important to have a path from one's trust requirements to concrete trust policies. Our proposal is to ease the building of this path, through a representation of trust requirements grounded on a specific notion of trust and an algorithm to map this representation to trust policies. We report our ongoing effort on this direction.

1 Introduction

One of the great challenges to the semantic web is the problem of trust. Operational measures of trustworthiness are needed to separate relevant and truthful data from those that are not [2]. However, to be correctly interpreted, these measures must be linked with real-world concepts of trust. They also must meet the trust requirements of their users. Building on the trust concept found in [4], our work aims to pave the path leading from a user's trust requirements to operational trust policies that can be applied to semantic web data, while preserving the relation between the resulting policies and the trust requirements we started with. This relation is important as it enables the user to find out why a piece of data was found trustful. We're focusing on the semantic web data exchange scenario, where an agent receives some data and must decide whether or not to trust them.

Here we report our ongoing effort in this direction, which includes a model to represent trust requirements and a small test implementation, based on the Semantic Web Publishing vocabulary [3].

1.1 Related Work

In [1] we find a very similar work: a Semantic Web Browser with filtering based on trust policies. The user selects the trust policy he wishes to use and then the software filters information using that policy. Besides that, it offers explanations of why each piece of information was trusted. The difference lies in the level of abstraction. The

proposal in [1] offers facilities to express trust policies as pieces of TriQL queries in such a way that it allows an explanation of why a triple was found trustful. Our work deals with representing trust requirements and translating them to trust policies that preserve real-world trust relationships.

1.2 A Motivating Scenario

The scenario we are focusing on is based upon two works: the Semantic Web Publishing scenario [3] and the DBin project [5]. The Semantic Web Publishing scenario has agents embodying two roles: of information providers and of information consumers. An information provider publishes RDF graphs; these graphs contain information and its metadata, such as provenance, publishing date etc. An information consumer gathers these graphs and decides what to do with them, provided that these graphs can be seen as claims of the information provider, rather than definitive facts. The formal meaning of these claims, that is, what statements about the world are being made, is given by a set of *accepted graphs*, which is a subset of the graphs the information consumer receives. It is assumed that the agent will act based solely on information contained in accepted graphs.

The Semantic Web Publishing proposal also enables the user to specify a *trust policy*, that is, a set of conditions that the received information should meet to be accepted. An example of a policy would be "trust all information that comes from direct friends and is about computers".

This scenario can be integrated with the one outlined in [5]: a P2P network where people exchange RDF graphs of interest and store all the received graphs in a local database. Filtering can be applied to hide triples that do not match the user's criteria. One use for this is the implementation of trust policies. The set of visible triples is similar to the one of accepted graphs seen above; we will name it the set of *accepted triples*.

These scenarios are only examples of possible uses of trust and trust policies within the Semantic Web context; many other scenarios are possible, such as Semantic Social Desktops.

2 A Model of Trust

2.1 The Concept of Trust

Following the ideas presented in [4] and [7], we will use the concept of trust as "knowledge-based reliance on received information", that is, an agent decides to trust (or not) based solely on his knowledge, and the decision to trust implies the decision to rely on the truth of the received information to perform some action. We will elaborate some key aspects of this definition below.

- **Knowledge-base trust**. The agent's knowledge includes all information the agent has, which in turn includes information received from other agents and self-gathered information. We will call the subset of this information that the agent has decided to trust "trusted information". Received information that is not trusted will

be called “known information”. The decision to trust is not irrevocable: knowledge can evolve and new evidence may render formerly trusted information untrusted, and vice versa.

- **Trust as reliance**. The idea of reliance means that the agent can use the trusted information to achieve some goal, without further analysis – although this may imply running the risk of taking an inappropriate action if the information is false. For example, if an agent trusts the information that www.mybank.com is the URL of his bank, then he will send his password to this site without further checks: he relies on this information for performing financial transactions. If the agent does not have any action depending on that information, then there is no reliance attitude and the concept of trust does not apply (for example, if he has no relationship with this bank, then the information about the URL does not matter: it is not an object of reliance).
- **Reliance on information**. This concept is about trust on information, where “to trust” means “to move known information to the set of trusted information”. There are other actions that need trust, beyond accepting information: moving money from one account to another, running an unknown software, providing sensitive information to a website, granting access to an intranet etc. Nevertheless, many of these actions rely on knowledge about the action itself, the agents involved and the circumstances: a bank normally relies on a supplied account number and password to grant access to a person's account (actually, it relies on the relation between the person, an account and a password). So, the trust concept as defined here can also be applied to these actions whenever it is possible to factor out the reliance on some kind of received information. In the case of running an unknown software, the agent will trust it or not based on the information he has about that software, e.g. who obtained it, what it does, who is the publisher etc.

There are other important aspects of the concept of trust that are relevant to the proposed model:

- Trust can be seen as a relationship between two agents mediated by a goal: one trusts somebody for something [7]; in our case, trusts somebody for receiving information from him.
- Trust is subjective, that is, each agent may have a different view about what can be trusted. So, from now on we will use the term *trusting agent* to denote from whose viewpoint trust is being evaluated.
- It is of common sense that a person normally trusts himself as a provenance of information, although he might give up on this perception if someone he believes to be wiser (with respect to this particular subject matter) contradicts him. In this work we assume that the default attitude of the trusting agent is to trust everything that comes from himself.

2.2 The Trust Act

When an agent receives information, he must decide whether or not to trust it: this is the trust act [4]. To do this, the agent can use the following elements [6]:

- The context of the received information, that is, metainformation about circumstances (provenance, date, time, location, reason, relation of the provenance with

the trusting agent etc). For example, the sender, the date and the subject of an e-mail are part of its context. However, the context may include information not present in the received information but in the agent's knowledge, like the sender's job, which one might have stored in his personal agenda. This is possible when there is some kind of URI for the sender, which "links" the agent's knowledge with the received information.

- The content of the received information.
- The reputation of the source, that is, what other agents say about it.

These elements provide *information* about the received information. The trust act consists of checking whether or not these elements satisfy some conditions. One trusts a received e-mail when he knows the sender, for example. These conditions will be called *trust requirements*. Continuing the e-mail example, we can formulate the following example trust requirements when reading e-mail:

- To download an e-mail, it must be from a known source.
- To open an e-mail, it must be written in my native language.
- To run an executable attachment, it must have been sent by a close friend and must have been verified by some kind of antivirus software.

As the trust act itself relies on information, it is reasonable to require that it should be based only on trusted information. This has two important side effects:

- A source's reputation becomes part of the context, as it will be composed by trusted information (that is, trusted opinions of other agents) related to the source. Consequently, we will restrict the elements of the trust act to encompass context and content. Notice that it is possible to use untrusted information about reputation to make a trust decision: it is what happens in many reputation systems, where the score used to evaluate an agent is made from opinions of unknown agents.
- Some of the contextual information may come together with the received information. An e-mail carries information about its sender, the date it was posted etc. If the trust acts related to e-mail demands some of this contextual information, these acts will fail (that is, the e-mail will not be trusted) until the contextual information is also the subject of a trust act and gets included in the set of trusted information. So, for a trust act about the content of received information to succeed, prior trust acts about its context should have been made, otherwise the former trust act may fail due to lack of trusted information. For instance, to reason about an e-mail using the sender's name, I must trust that who claims to be the sender *is* the sender indeed.

2.3 Formalizing Trust

From the concepts presented above it is possible to define trust as a predicate over knowledge (*K*), provenance (*p*), context (*c'*) and content (*c*) of information: *T(K,p,c',c)*. This predicate is defined by the set of trust requirements of the trusting agent. From now on, we will call it the *root trust predicate*. Notice that agents with the same knowledge may react differently when faced with the same information, as they may have different trust requirements.

Using the idea of trust requirements as conditions on received information, we can represent one's trust requirements using logical predicates, similarly to [8]. The trust

problem then becomes the one of building the root trust predicate. Instead of tackling the problem of building a single predicate that encodes all trust requirements of an agent (a "top-down" approach), we can try to reduce the problem to building the root trust predicate as a composition of simpler predicates that can be evaluated independently. This way, the root trust predicate becomes a disjunction of these simpler predicates: when faced with some information, it will be trusted if at least one of these simpler predicates is true. We call these simply *trust predicates*.

As an example, the root trust predicate shown below states that the user will trust information if it is "good email" or "good software".

$$T(K, p, c', c\ ) \leftarrow GoodEmail(K, p, c', c) \vee GoodSoftware(K, p, c') \qquad (1)$$

Each of the trust predicates states necessary and sufficient conditions to trust some piece of information. This can be put in evidence by breaking them into conjunctions of other trust predicates:

$$\begin{aligned} GoodEmail(K, p, c', c) \leftarrow\ & IsFromFriend(K, p) \\ & \wedge IsEmail(c) \wedge \neg Old(c') \end{aligned} \qquad (2)$$

What distinguishes trust predicates from other logical predicates is that the former express meaningful conditions for trust *from the agent's viewpoint*. In the example, it does not matter how the name of a source is represented; what matters is whether or not the source is a friend.

Trust predicates should express trust conditions that make sense for the trusting agent, linking his trust-related concepts (e.g. "being a friend", "belonging to a research group", "being a relevant author") to logical conditions on the information available (e.g. "being referenced in my personal FOAF file", "having a link from a specific web page to the source's FOAF", "appearing in the citations of more than five publications"). Another possibility is to define them in terms other trust predicates (e.g. the GoodEmail predicate in the example above: it is a trust predicate composed by other trust predicates), in this case, we will call it a *composite trust predicate*. In contrast, an *elementary trust predicate* is one that has no meaningful decomposition from the agent's point of view and must be built by directly stating conditions on information.

To strengthen the connection with real-world trust, we will apply the formal model with the trust concept presented in [4], where the decision to trust is made based on the appreciation of three things: the subject matter (the content), the entity involved (in our case, the provenance), and the circumstances (the context). Therefore, if trust predicates can be specified to verify these three aspects, then it is possible to solve completely an arbitrary instance of the trust problem (within the limits of expressiveness of the formalisms used) by making a trust predicate that is a conjunction of those trust predicates. We will call this conjunction, namely, the triple <subject matter, entity, circumstances>, a *trust-point* [4]. Accordingly, the root trust predicate can be described as a disjunction of trust-points. In the preceding example, *GoodEmail* is a trust-point: it evaluates the entity (first predicate, *IsFriend*), the subject matter (second

predicate, *IsEmail*) and the circumstances (third predicate, not *Old*). If the three trust predicates are true for the received information, then it is considered trustful. Summarizing, a trust-point is a set of conditions on *who* sent the information, on *what* is the information, and on what *circumstances* surround it. According to this model, the trust act consists of applying the root trust predicate to a known fact to determine whether or not it should be trusted.

It is important to notice that the provenance of information is part of the context in [6], whereas it is factored out in [4]. We adopted the latter approach, grounded on the idea that trust is a relationship between agents and hence the model should capture this explicitly.

2.4 An Example

Looking at the scenario presented and restricting it to the exchange of scientific information among researchers, we can identify many trust predicates, some more general (i.e., they apply to other domains), others more specific. Each predicate is followed by its name in the logical formulas:

- Trust predicates related to entities involved: "works with me" (Colleague), "is a relevant researcher" (IsRes), "is cited by other authors" (IsCited).
- Trust predicates related to the matter: "is a publication" (IsPubl), "is a website" (IsSite), "is a relevant website" (GoodSite), "is cited by a relevant website" (CiteByGoodSite), "is the contact information of a researcher" (InfoRes), "is a relevant publication" (GoodPubl), "is the contact information of a person" (IsCInfo).
- Trust predicates related to circumstances: "is recent enough" (IsRecent), "is old enough", "is newer than my preferred publications", "is hosted in a university" (HostUniv).

With these trust predicates, we can model the trust requirements of two hypothetical agents, John and Mary, concerning acceptance of scientific information.

$$\begin{aligned} T_{John}(K,p,c',c) &\leftarrow GoodPubl_{John}(K,p,c) \vee GoodSite(K,p,c) \\ T_{Mary}(K,p,c',c) &\leftarrow GoodPubl_{Mary}(K,p,c',c) \vee InfoRes(K,p,c) \end{aligned} \quad (3)$$

T_{John} and T_{Mary} are the root trust predicates representing trust requirements of the agents. John is prepared to trust relevant publications and websites; Mary also is prepared to trust relevant publications (although she may have a different concept of what a good publication is, as will become clearer the example), as well as contact information of researchers. *GoodPubl*, *GoodSite* and *InfoRes* are the trust-points involved. Now we proceed to describe these trust-points using the <subject matter, entity, circumstances> template. In Table 1, each trust-point will be described in natural language, and then cast as trust predicates.

Table 1. Description of the trust-points

Trust-point	Subject Matter	Entity	Circumstances
$GoodPubl_{John}$	It is a publication and it is cited by a relevant website	It has been sent by a colleague	-
$GoodPubl_{Mary}$	It is a publication and it is cited by relevant publications	It has been sent by a researcher	It is not recent
GoodSite	It is a website	It has been sent by a researcher	It is hosted in an university
InfoRes	It is contact information of a known researcher	It has been sent by a colleague	-

Now it is possible to specify each trust-point using the trust predicates shown before:

$$\begin{aligned} GoodPubl_{Mary}(K, p, c', c) &\leftarrow IsPubl(c) \wedge IsRes(K, p) \wedge \\ &\quad \exists s(IsCited(K, c, s) \wedge GoodPubl(K, s)) \\ &\quad \wedge \neg IsRecent(c') \\ GoodPubl_{John}(K, p, c) &\leftarrow IsPubl(c) \wedge Colleague(K, p) \wedge \\ &\quad CitedByGoodSite(K, c) \\ GoodSite(K, p, c) &\leftarrow IsSite(K, c) \wedge IsRes(K, p) \\ &\quad \wedge HostUniv(K, c) \\ InfoRes(K, p, c) &\leftarrow Colleague(K, p) \wedge \\ &\quad \exists r(IsCInfo(c, r) \wedge IsRes(K, r)) \end{aligned} \quad (4)$$

One thing to notice is the "reuse" of trust predicates (*IsPubl*, *Colleague*, *IsRes*): we believe this decomposition allows great simplification of the process of building trust-points, as many trust decisions rely on the presence (or absence) of the same properties (for example, "being a colleague", "being a relevant author", "being cited").

2.5 A Model for the Trust Process

Having a model for the trust requirements, now we can proceed to model the dynamics of trust: how the trust acts are combined to form what we call the *trust process*, which is the process by which an agent keeps its trusted facts base with all the facts that the agent trusts, according to its trust-points, and with no facts that he does not trust.

We propose the following procedure for realizing the trust process:

1. Include in the known facts base facts contained in the received information.
2. Remove one fact from either the trusted or the known facts base that has not been analyzed yet.
3. Apply the trust act to it.
4. If the fact tested is found trustful, then include it in the trusted facts base. If not, include it in the known facts base.
5. Go back to step 1 until all facts (either known or trusted) have been analyzed.
6. If no fact changed its status (from known to trusted or vice versa), the process ends; otherwise, restart the process from step 2.

This procedure is depicted in figure 1. Notice that the trust act uses only the set of trusted facts and the trust-points to decide the trustfulness of a fact.

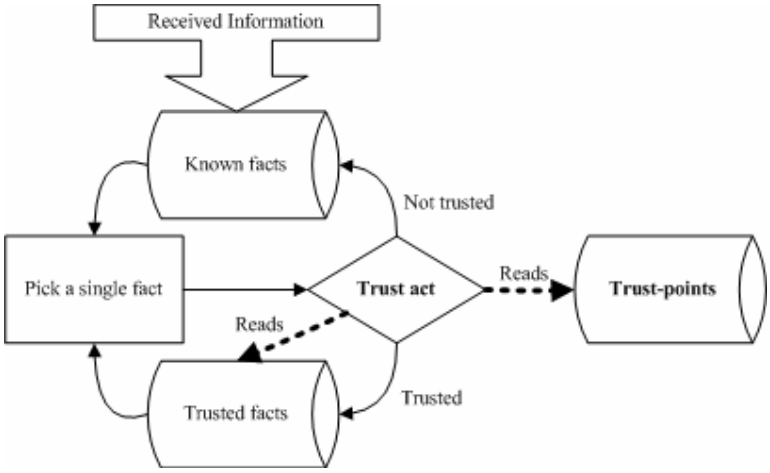


Fig. 1. Depiction of the trust process

An iterative procedure is used, as the conditions that each trust-point states to accept a set of facts (for a given provenance) are fulfilled depending on the presence (or absence) of other trusted facts. So, to trust a fact, it may be necessary that other facts have already been trusted, which also means that trusting a fact may lead other facts to be trusted. The same happens when a fact loses its trust status: other facts that depended on this one may also become untrusted.

The last step of the procedure is justified by the functional nature of the trust act: if there are no changes in the inputs, then nothing will change if it is applied again.

The dependence among facts shown above presents an interesting property of the model: the interplay between trust-points. The dependence that exists in practice among facts reveals the intrinsic dependence among trust-points. When a trust-point is added, it may implicitly enter in a chain of trust-points. These implicit chains of trust-points resemble the trust policies that are explicitly defined in other approaches

like [8]. We say that the model allows the implicit definition of arbitrarily complex trust policies through the use of its building blocks, the trust-points.

2.6 Trust Transitivity

One often-used property of the trust relationship is the transitivity [9, 10]. It is commonly used jointly with trust degrees to represent trust relationships as weighted graphs, where the weight is the degree of trust. Trust propagation algorithms are used to infer the degree of trust between unrelated nodes (that is, nodes with no direct edge between them).

Our approach does not yet use transitivity of trust in the model: it does not support trust-point inference through the use of trust relationships. In other words, trust-points are independent: the fact that one agent trusts another on some aspect does not influence the agent's trust on a third agent.

However, the model does provide transitivity *de facto* in a scenario of information sharing where agents use the proposed trust model and each one of them only shares information that they trust. To see how this happen, we must derive a trust graph, where each trust-point originates an edge from the trusting agent to the trusted agent (the provenance); this edge's "weight" is the subject matter of the trust-point. If we merge the trust graphs of several agents, one might find a path in which all edges have the same subject matter, e.g. John trusts Mary on finding papers, Mary trusts Daniel on finding papers and Daniel trusts Mark on finding papers. In the example, when Mark finds an interesting paper, it will eventually end up being trusted by all the agents in the path (see Fig. 2).

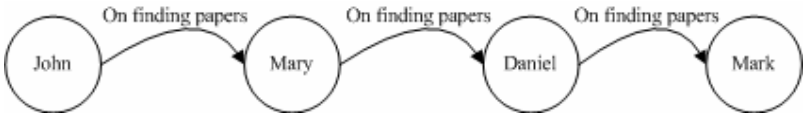


Fig. 2. *De facto* trust transitivity

2.7 Degrees of Trust

It is a common perception that trust has degrees: one can trust someone more than someone else [7]. In formal terms, there is an order on the relation of trusting agents and information sources. Some models try to capture this order through the assignment of trust ratings to trust relationships [10]. This assignment is made for some pairs of agent-source and then the remaining ratings are inferred [9].

Although the proposed model does not use trust degrees, it does capture the ordering of trust relationships based on the idea that to trust more one agent means to trust him over a larger scope of "things". A single trust-point's subject matter gives a set of trusted facts for some provenance. Given a trusting agent T_a, another agent A, we can group all trust-points of T_a that include A in their provenance. From this set of trust-points we can extract the set of subject matters on which T_a trusts A, which gives the *extent of trust of T_a with respect to A*. To visualize this, we can look again to the trust

graph originated by the trust-points: each pair of nodes can have one or more edges, one for each subject matter. The set of edges between two nodes gives the extent of the trust relationship, as exemplified in Fig. 3, which shows the extents of trust of John with respect to Mary (accepting web sites, Semantic Web papers and rock music) and Bob (accepting web sites and antivirus software). Formally, we can say that the trust predicate associated with a subject matter *entails* the set of facts that will be trusted. Then, we can say that the extent of trust of T_a with respect to A entails the set of all facts that T_a trusts when coming from A.

Now the problem is reduced to finding an order on the set of extents. A partial ordering on it is given by the entailment relation: an extent A is strictly greater than the extent B if A entails B and B does not entail A. Loosely speaking, an extent is greater than other if it *contains* the other extent. If there is no entailment relation between two extents, then it is not possible to order them.

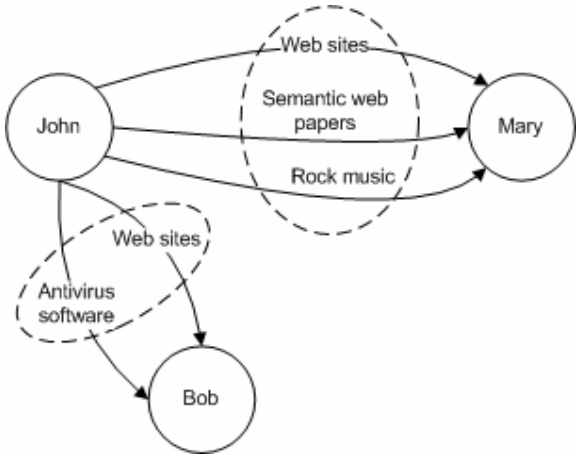


Fig. 3. John is the trusting agent. The dashed lines show the extents of trust of John with respect to Mary and Bob.

2.8 Applying the Model to the Motivating Scenario

The above model of trust can be applied to the scenario of interest with the following mappings:

- A fact is an RDF triple pertaining to a named graph [3]. We used named graphs in order to support attachment of provenance and contextual information to graphs (see below).
- The trusted facts are the accepted triples.
- The trusting agent is the information consumer.
- The circumstances are facts whose subject is the context of a triple, that is, its surrounding named graph. The provenance is a particular circumstance that will be treated separately, as justified before.

- The Semantic Web Publishing vocabulary [3] provides means to attach provenance information to graphs, establishing a relation between an agent and a graph. This relation can be of assertion, quoting, denial etc. So, a triple's provenance is the entity asserting this triple's graph.
- The subject matter *entails* a set of triples that the trusting agent relies on (for a particular provenance). The subject matter "receiving academic articles" will entail all triples that, in the domain theory of the agent, assert that something is an academic article. Notice that this relation can (and normally will) be described intensionally. In the preceding example, the entailment relation can be stated using an RDF property whose domain is the set of academic articles.
- The trust policy of an agent for receiving information is given by its set of trust-points. The trust process presented enforces this policy, separating reliable facts from facts that are just known.

3 Prototype Implementation

We tested the preceding ideas in a prototype solution aimed to partially implement the trust process presented above. We used named graphs to store triples and TriQL (the extension of RDQL for named graphs) to represent (and apply) trust predicates. Currently we do not implement neither composite nor negated trust predicates. We also do not support transparently blank nodes due to limitations of the underlying implementations used. Currently all blank nodes are transformed into fake URI nodes during the prototype execution.

The trust-points were expressed using a simple ontology, mirroring the model's structure: each trust-point is composed by one or more trust predicates, which may be predicates on the subject matter or on the provenance. We do not provide separate predicates for circumstances yet, although they can be implemented as subject matter predicates without loss of expressiveness. The elementary trust predicates are expressed as graph patterns and sentence patterns or URI, depending on the type of predicate (subject matter or provenance). Follows a sample of the trust-points and trust predicates we used in our tests. The trust-points are represented using N3 format.

```
# Describes the trust-point GoodSite

ex:goodSite a trust:TrustPoint;

      # Provenance must be a researcher
      trust:provenance trustpred:isResearcher;

      # Subject matter must be a web site
      trust:subjectMatter trustpred:isSite;

      # The site must be hosted on a university
      trust:subjectMatter trustpred:isHostedOnUniv .
```

```
# Describes the trust-point GoodPublJohn

ex:GoodPublJohn a trust:TrustPoint;

      # Provenance should be a John's colleague
      trust:provenance trustpred:isColleagueJohn;

      # Subject matter must be a publication and
      # must be cited by a good web site
      trust:subjectMatter trustpred:isPublication;
      trust:subjectMatter trustpred:citedByGoodSite .


# Describes the trust predicate IsRes

trustpred:isResearcher a trust:ElementaryPredicate;
      trust:graphPattern
            "?g (?GRAPH swp:assertedBy ?g .
                 ?g swp:authority ?ENTITY)
             (?ENTITY foaf:member ?u)
             (?u rdf:type ex:University)" .

# Describes the trust predicate isPubl

trustpred:isPublication a trust:ElementaryPredicate;

      # This predicate is described as a sentence
      # pattern. It will allow sentences whose
      # predicate is dc:type. This is the trusting
      # agent's view of what is a publication.
      trust:sentencePattern
            "?ANYTHING dc:type ?ANYTHING" .
```

The graph patterns are used to express conditions on the set of trusted facts, that is, what must be already known to trust a triple. They follow TriQL syntax and may use variables. Each trust-point is converted to a TriQL query, which is applied to the set of trusted facts. There are four special variables, whose values are bound before running the queries: GRAPH, SUBJ, PRED and OBJ, which are bound respectively to the graph name, the subject of the triple being tested, its predicate and its object. These variables allow the trust predicates to test relationships between the triple being analyzed with the trusted facts. The graph patterns of all trust predicates of a trust-point are concatenated to build the query.

The sentence patterns restrict the valid values on each of the triple's components. If the agent does not wish to restrict some of them, he may use the special variable ANYTHING. These restrictions appear in the query as conditions on the variable's values.

To deal with provenance information, the prototype recognizes the special variable ENTITY and allows the trust predicate to specify a concrete URI value for it. The trust predicate is responsible for providing a graph pattern that binds this variable to the node that represents the provenance of the triple being tested.

The trust-point *GoodSite* shown above is translated to the following TriQL query that, given a triple, rules it out if it does not match the conditions imposed by the trust-point.

```
SELECT * WHERE ?GRAPH (?SUBJ ?PRED ?OBJ)
                      (?SUBJ ex:hostedOn ?u)
                      (?u rdf:type ex:University)
                  ?g  (?GRAPH swp:assertedBy ?g .
                       ?g swp:authority ?ENTITY)
                      (?ENTITY foaf:member ?u)
                      (?u rdf:type ex:University)
           AND ?PRED eq rdf:type
           AND ?OBJ eq ex:Website
```

Once the trust-points are translated into queries, the prototype cycles through the set of triples not yet trusted, following the trust process defined earlier. Currently it operates on a static knowledge-base, that is, no new facts can be added during its execution. This restriction will be removed in future developments.

We tested the prototype with some hand-made sample data and it performed as expected, being capable of implement and enforce the trust policies expressed by the sample trust points presented. Follows the output of the prototype in one of the tests, showing which triples got trusted in which cycle. The name of the trust-point who allowed the inclusion of the fact is between angle brackets before each sentence.

```
Loading knowledge base...
Loading trust-points...
Running trust-point engine...

====> Cycle 1

+ {goodProvenance1} ex:JohnWarrant  swp:assertedBy
ex:JohnWarrant
+ {goodProvenance1} ex:AnnWarrant  swp:assertedBy
ex:AnnWarrant
+ {goodProvenance3} ex:JohnWarrant  swp:authority
ex:John
+ {goodProvenance3} ex:AnnWarrant  swp:authority
ex:Ann
+ {selfTrust} ex:JohnData  swp:assertedBy
ex:JohnWarrant
+ {selfTrust} ex:John  foaf:knows  ex:Ann
+ {selfTrust} ex:John  foaf:name  "John M."
+ {selfTrust} ex:puc  rdf:type  ex:University
+ {selfTrust} ex:Ann  foaf:member  ex:puc
+ {selfTrust} ex:swsite  ex:cites  ex:book
```

```
+ {hosting} ex:swsite  ex:hostedOn  ex:puc
+ {goodProvenance2} ex:AnnData  swp:assertedBy
ex:AnnWarrant

====> Cycle 2

+ {goodSite} ex:swsite  rdf:type  ex:Website

====> Cycle 3

+ {GoodPublJohn} ex:book  dc:type  ex:Book

====> Cycle 4

FINISHED
```

4 Conclusions

Our goal was to build a formalism to capture, represent and apply trust requirements of an agent in the scenario of Semantic Web data exchange, while preserving the real-world semantics of trust. This was done using the trust-point concept as a unit comprising all information needed to decide the trustfulness of received information. The composition of trust-points yields trust policies that can be realized using the trust process proposed. We presented a partial prototype implementation fulfilling this trust process, using TriQL queries to implement the trust policies.

Differently from Bizer's work [1], where each policy must specify all the conditions the triples must fulfill to be accepted, in the proposed model the trust policies can be built incrementally, as each trust-point can be specified independently from the others, while cooperating with them in each trust act. The addition of new trust-points enriches the resulting trust policies. In fact, these trust policies emerge from the interactions between the different trust-points. We believe this represents more realistically how trust acts occur in the real-world.

The next steps in this work include a deeper study of the proposed formalism in order to evaluate its properties (expressiveness and computational complexity, among others) and to see how it compares to other existing models. We also wish to complete the prototype's implementation and use it in more realistic scenarios, including Social Semantic Desktops and P2P networks.

5 References

1. Bizer, C., Cyganiak, R., Maresch, O., Gauss, T.: TriQL.P - Trust Policies Enabled Semantic Web Browser. http://www.wiwiss.fu-berlin.de/suhl/bizer/TriQLP/browser/
2. Guha, R.: Open Rating Systems. 1st Workshop on Friend of a Friend, Social Networking and the Semantic Web (2004).
3. Carroll, J. J., Bizer, C., Hayes, P., Stickler, P.: Named Graphs, Provenance and Trust. Technical report HPL-2004-57 (2004).

4. Gerck, E.: Toward Real-World Models of Trust: Reliance on Received Information. http://www.safevote.com/papers/trustdef.htm.
5. Tummarello, G., Morbidoni, C., Puliti, P., Piazza, F.: The DBin Semantic Web platform: an overview. http://www.instsec.org/2005ws/papers/tummarello.pdf.
6. Bizer, C., Oldakowski, R.: Using Context- and Content-Based Trust Policies on the Semantic Web. In: International World Wide Web Conference (2004).
7. Castelfranchi, C., Falcone, R.: Social Trust: A Cognitive Approach. In: Castelfranchi, C.; Yao-Hua Tan (Eds.): Trust and Deception in Virtual Societies. Springer-Verlag (2001).
8. Nejdl, W., Olmedilla, D., Winslett, M.: PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web. Workshop on Secure Data Management in a Connected World (SDM'04) in conjunction with 30th International Conference on Very Large Data Bases, Aug.-Sep. 2004, Toronto, Canada
9. Guha, R., Kumar, R., Raghavan, P., Tomkins, A.: Propagation of Trust and Distrust. In: International World Wide Web Conference (2004).
10. Golbeck, J., Parsia, B., Hendler, J.: Trust Networks on the Semantic Web. Proceedings of Cooperative Intelligent Agents (2003), Helsinki, Finland.

Specification of Policies for Automatic Negotiations of Web Services

Steffen Lamparter and Sudhir Agarwal

Institute of Applied Informatics and Formal Description Methods (AIFB),
University of Karlsruhe (TH), Germany
{lamparter,agarwal}@aifb.uni-karlsruhe.de

Abstract. Market mechanisms provide an efficient institution for allocating service offers and requests. In doing so, negotiations between the market participants play a crucial role. However, current policy languages are ill-suited to realize beneficial trade-offs within a negotiation, since they support only boolean decisions. Therefore, we suggest an approach where preferences are modeled as utility functions. We show, how such preferences can be specified with description logics to enable the use of existing inference engines for calculating the degree of policy satisfaction by offers/requests which can be considered in the negotiation process.

1 Introduction

Web services are self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces, e.g. WSDL. They allow flexible and dynamic software integration that is often referred to as the ”Find-Bind-Execute”-paradigm. Moreover, by using standard Internet technology, Web services facilitate cross-organizational transactions and thus outsourcing of software functionality to external service providers. When moving from distributed systems operating within one company to systems that involve different, independent companies, the ”Find-Bind-Execute”-schema describes nothing else than a B2B procurement process, where digital services such as information delivery or execution of calculations are purchased. Thus, service-oriented computing requires an infrastructure that provides a mechanism for coordinating between service requesters and providers. This coordination mechanism has to provide a platform where potential business partners can be discovered, prices can be ascertained, and contracts can be closed. A marketplace, where prices are determined by the interplay between supply and demand, can be regarded as a coordination mechanism that efficiently provides these functionalities [1].

1.1 Web Service Markets

Figure 1 brings together the phases that can be identified in an electronic market [2, 3] and the typical Web Service usage process which comprises the steps discovery, composition, negotiation, and finally contracting. In the *Matchmaking Phase* suitable services are discovered. Since a certain goal can not be accomplished only by a single service but also by a combination of services this phase also includes composition. After having determined those services that are able to achieve a certain goal an optimal assignment of service requests and offers with respect to the individual utility of the participants or

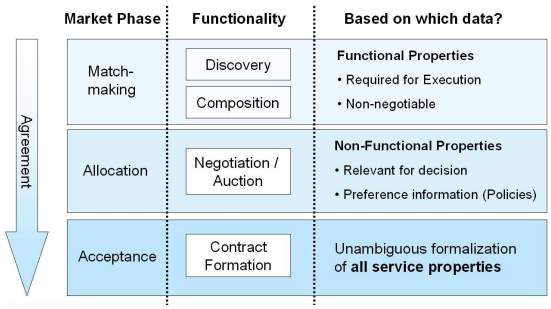


Fig. 1. Market phases and the Web Service usage process

to the overall welfare has to be found in the *Allocation Phase*. To achieve this, negotiations between the participants have to be carried out. For determining the allocation and price many different mechanisms are available ranging from simple selection approaches to complex negotiation or auction schemes. After this allocation, legally binding contracts are closed between the corresponding business partners in the *Contract Formation Phase*. These contracts have to be formalized in a machine-understandable way in order to allow automated execution and monitoring.

In each market phase different kind of information is required. *Functional properties*, required in the *Matchmaking Phase*, are those attributes that are mandatory to be able to invoke a service and to integrate the results, e.g. the input and output of a service. That means, that for functional properties no alternatives can be specified and thus negotiations about such properties are impossible. All discovered services fulfill the desired goal but may differ in their *non-functional properties* which are attributes that are not required to invoke the service nor to integrate the results, but they are the decisive factors for service selection and price determination. For example, price, payment method, security as well as trust attributes, and most notably quality of service attributes. Typically, for each non-functional attribute there are several alternatives that can be adopted depending on the preferences of the trading partners. Thus, a negotiation has to be carried out to agree on one of the alternatives. In order to be able to negotiate, preference information about the different alternatives is required. In case of an automatic negotiation it is not enough that preferences are in the user's mind, but they have to be formalized explicitly.

Here, the policies come into play. Policies allow to declaratively express preferences, i.e. which of the different alternatives a non-functional property may adopt. Thus, policies can be regarded as constraints or rules that restrict the decision space within the negotiation of agreements.

1.2 Some Motivating Scenarios

In this section some motivating examples are presented in order to illustrate why negotiations about non-functional properties are required. We come up with examples from the domains privacy and quality of service. However, the problems are the same for other non-functional properties.

Privacy. A Web service might support different privacy levels. For example, a provider either gives a guarantee to delete customer data straight after the business interaction was carried out or the provider stores customer data for further usage. In the latter case a discount on the service price is given to the customer, i.e. the customer could sell private data in exchange for a discount. Which of the alternatives is more preferable to the customer depends on how important data privacy as well as a cheap price is judged by the customer.

Quality of Service. A Web Service interaction involves several different quality of service criteria like response time, availability, etc. Typically, not all criteria are perfectly met by the service providers, rather each provider has his strengths and weaknesses. In order to decide which service suits best exact information about the requesters preferences are required, e.g. is a service with fast response time and bad availability better than a service with the converse properties. Moreover, one has to know if a $10 discount in price justifies a slower response time of 10s.

In each of this examples there is a trade-off between different service properties (e.g. quality vs. price, privacy vs. price, privacy vs. quality) which can only be resolved by making the different attributes comparable. This can be realized by assigning utility values to the different decision alternatives. Such cardinal preferences allow to decide whether a certain discount is high enough to compensate the loss in utility that results from a disadvantageous property value. Hence, negotiations between the parties may lead to service configurations that yield higher welfare for providers as well as requesters.

The paper is organized as follows. In section 2 a general policy framework is introduced and extended to enable the representation of fine-grained preferences. We show how preferences can be evaluated in a DL reasoner by mapping utility functions to an appropriate description logic. We conclude in section 4 after discussing related work in section 3.

2 Specifying Policies for Negotiation

In this section, a formalism is presented that allows formal specification of preferences and thus facilitates automatic decision making and negotiations. In section 2.1, a general framework for expressing policies is introduced. This framework is based on the foundational ontology DOLCE [4]. Foundational ontologies capture typical *ontology design patterns* (e.g. location in space and time). By providing a sound conceptual model with precise concept definitions they facilitate integration of different policy efforts (cf. [5]). In section 2.2 the description framework is extended to enable the representation of fine-grained preferences by means of description logics.

2.1 Policy Description Framework

In this section, the generic policy description framework introduced in [5] is refined. The framework provides a generic ontology for expressing policies. Ontologies formalize concepts and concept relationships very similar to conceptual database schemata or UML class diagrams [6]. However, ontologies typically feature logic-based representation languages. Those languages come with executable calculi that allow querying and

reasoning during run-time. Moreover, ontologies facilitate the conceptual integration of heterogeneous policy efforts by providing well-defined and machine understandable semantics.

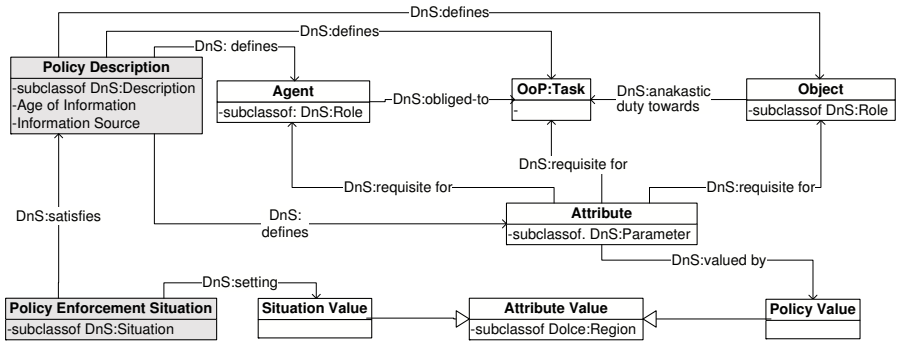


Fig. 2. Sketch of the Policy Description Framework.

In order to express policies we add a Core Policy Ontology to the DOLCE ontology stack. In the remaining part of this section we introduce the basic principles of DOLCE, present the design of the Core Policy Ontology, and show how the framework can be used to express concrete policies by means of an example.

DOLCE. The foundational ontology DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) provides the basis for the policy description framework used in this paper. Foundational ontologies are high-quality formalizations of domain independent concepts and associations that contain a rich axiomatization of their vocabulary. D&S (DnS) is an ontology module that extends DOLCE and introduces the basic distinction between descriptive (DnS : Description)[1] and ground entities (DnS : Situation). A Situation defines a state of affair (e.g. real settings in the world such as facts or cases), while a Description is a conceptualization which encompasses objects such as laws, plans, policies, etc. A detailed description of DOLCE and D&S can be found in [4] and [7], respectively. Moreover, in order to model workflow information as well as data the modules Ontology of Plans (OoP) and Ontology of Information Objects are introduced [7]. Descriptions contain Concepts such as FunctionalRoles, CourseofEvents, and Parameters. Ground entities in D&S are derived from DOLCE. FunctionalRoles are played − by Endurants, CoursesofEvents sequences Perdurants, Parameters are valued − by Regions.

Core Policy Ontology. In order to express policies we have to extend the basic vocabulary with policy specific concepts and relations, while reusing the foundational ontologies as far as possible. This core ontology contains the basic building blocks needed for modeling policies.

[1] Concepts of the ontology are written in sansserif. For concepts and relations that are directly contained in the corresponding ontology name spaces are omitted, for those derived from other modules the corresponding name space is mentioned explicitly.

Figure 2 sketches the Core Policy Ontology (CPO) in a simplified way. All concepts of the CPO are subclasses of DOLCE top-level concepts. A policy description consists of the concepts Agent, OoP : Task, Object, and Attribute. The entities Agent, OoP : Task, and Object allow to define the application area of the policy, while Attribute defines the property that is constrained by the policy. This could be, for instance, a constraint regarding the service, the agent that invokes the service, etc. The Attribute is DnS : valued − by an AttributeValue which is a Dolce : Region and specifies which attribute values are allowed according to the policy.

During run-time the policy has to be enforced by the system. This is done in a concrete PolicyEnforcementSituation which represents the current state of the system. In doing so, it has to be checked if the DnS : Concepts in the DnS : Description are DnS : classified by an entity in the DnS : Situation. If this is the case a DnS : satisfies relation is introduced between PolicyEnforcementSituation and PolicyDescription as specified in [8]. The actual attribute value in the situation (denoted by SituationValue) classifies the PolicyValue only in case the Dolce : Region defined in the SituationValue is contained in the Dolce : Region of the PolicyValue. In this case the policy is met.

Example. In the following we show how the framework introduced above is applied to specify concrete policies. Again we fall back on the privacy and quality of service domains. Consider a requester policy which says that a provider may store private data of the customer only for up to 14 days. This can be formalized by means of the Core Policy Ontology as show in figure 3. All concepts of the description are instantiated by domain specific entities. The policy will be applied if Web Service Providers (WSProvider) store PrivateData and permits this only for 14 days. Additionally, a quality of service policy is added, which specifies that the response time of the other party should be less than 5 seconds. Therefore, a additional Attribute ResponseTime is added. Both attributes are valuedby an IntegerRegion.

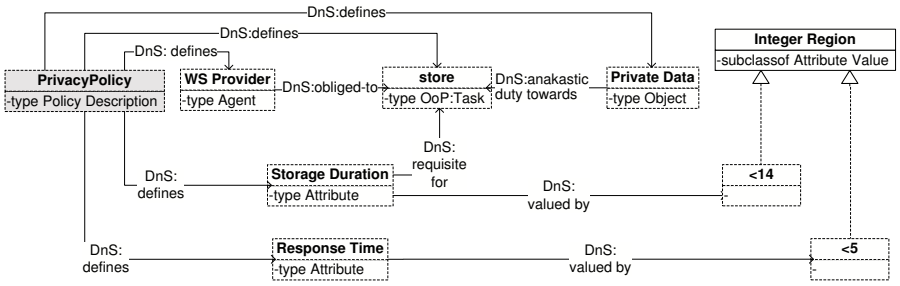


Fig. 3. Specification of a Privacy Policy.

2.2 Utility-based preference specification

In section 2 a generic framework for specifying policies is introduced. However, this framework is not expressive enough to capture fine-grained preferences as required for

supporting automatic negotiations. In the field of economics multi-attributive utility theory [9] is typically used to address these problems. It allows to handle trade-offs between alternatives and provides the right means for finding optimal service configurations. After introducing the basic idea behind utility theory, we show how such an utility approach can be integrated into our policy description model. To achieve this, utility functions are mapped to description logics. Further, we come up with an example to illustrate how fine-grained policies can be specified and rankings can be derived.

Utility Theory. In the context of utility theory a preference structure is defined by the complete, transitive, and reflexive relation $\succeq$. This means the property value $p_1 \in \mathcal{P}$ is preferred to $p_2 \in \mathcal{P}$ if $p_1 \succeq p_2$. The preference structure can be derived from the value function $v^i(p)$ of a user i.

$$\forall a, b \in \mathcal{P} : p_a \succeq p_b \Leftrightarrow v^i(a) \geq v^i(b)$$

The function $v^i(x)$ represents the utility defined by the relation $\succeq$ in a sense that the attribute values can be ranked by comparing the numeric values of the value function. Utility theory allows to decompose complex outcome spaces into utility functions composed of several lower-dimension functions. Thus, we can describe the preference structure for the attributes relevant to a specific service separately and then combine them to get the overall valuation. According to those definitions a user i specifies the utility function of the individual service properties X. Then, the overall valuation can be approximated by using the following additive value function.

$$V^i(x) = \sum_{j=1}^{n} \lambda_j^i v_j^i(x_j) \tag{1}$$

For the additive value function above we assume mutual preferential independence between the attributes [9]. Under this assumption we can easily aggregate the utility functions $v_j^i(x_j)$ of the individual attributes j to obtain the overall valuation of a service. Additive value functions are valid in many real world scenarios and might still provide a good approximation, even when mutual preferential independence does not hold exactly [10]. The weighting factor λ_j^i is normalized in the range $[0, 1]$ and allows to model the relative importance of an attribute j for a specific agent i.

Formal representation of preferences. In order to allow standard DL reasoners to make decisions based on the introduced utility approach, utility information has to be specified in a formal way. This can be done by modifying the concept PolicyValue of the Core Policy Ontology in a way that each property value in the set refers to a specific utility value. To allow for handling of discrete as well as continuous properties complex functions are required that map properties to utility values. In doing so, the satisfies-relation does no longer lead only to a pure boolean statement about the conformity of a Situation with respect to the PolicyDescription, but it leads to a statement about the degree of conformity. This is exactly the information that is required in order to automate negotiations. To facilitate the representation using description logics we restrict ourself to piecewise linear utility functions since such functions can be defined just by sets of points in $\mathbb{R}^2$. We use the description logic $\mathcal{ALC}(\mathcal{D} + \Sigma)$ with concrete domains and aggregates as proposed in [11]. The set of two points with adjacent x-coordinates

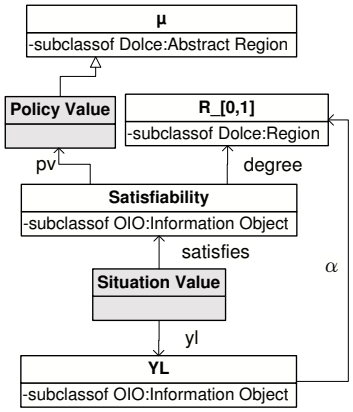


Fig. 4. Extended policy description framework.

can be interpreted as a straight line. For every line between (x_1, y_1) and (x_2, y_2) and a given x, we calculate an α as follows.

$$\alpha = \begin{cases} \frac{y_1 - y_2}{x_1 - x_2}(x - x_1) + y_1, \text{ if } x_1 \leq x \leq x_2 \\ 0, \text{ otherwise,} \end{cases} \tag{2}$$

We model this by defining a concept YL, to capture the αs as described above.

$$\mathsf{YL} \sqsubseteq \mathsf{OIO} : \mathsf{InformationObject} \sqcap \exists \alpha.\mathbb{R}_{[0,1]}, \tag{3}$$

where α is a functional role. This means for defining piecewise linear functions the semantics of PolicyValue has to be modified to a subclass of the Dolce : AbstractRegion μ that contains the set of points (x, y) which constitutes the utility function. Moreover, we define the relation $\exists$yl.YL from SituationValue to the Dolce : AbstractRegion YL. A SituationValue will be in as many relation instances of yl with instances of YL as there are lines in the utility function μ. The utility value of a SituationValue a according to μ is then just the sum of all such α over all the lines of μ. Further, we define a relation satisfies (specialization of OIO : realizes) from SituationValue to the OIO : InformationObject Satisfiability, which is defined as

$$\mathsf{Satisfiability} \sqsubseteq \mathsf{OIO} : \mathsf{InformationObject} \sqcap \exists \mathsf{pv}.\mathsf{PolicyValue} \sqcap \exists \mathsf{degree}.\mathbb{R}_{[0,1]}. \tag{4}$$

Now, the axiom

$$P_{=}(\mathsf{satisfies} \circ \mathsf{degree}, \sum(\mathsf{yl} \circ \alpha)), \tag{5}$$

where the predicate $P_{=}(x, y)$ is true iff $x = y$, ensures that the utility value of an individual a according to the function μ is equal to the sum of all α over all lines of μ. Based on this result we can calculated the weighted degree of satisfaction wds by means of the following formula:

$$P_{*}(\mathsf{wds} \circ \mathsf{degree}, \mathsf{satisfies} \circ \mathsf{degree}, \lambda_j^i), \tag{6}$$

The predicate $P_*(x, y)$ is true iff the condition $\mathsf{wds} * \mathsf{degree} = (\mathsf{satisfies} * \mathsf{degree}) * \mathsf{weight}$ holds. As already introduced, λ_j^i represents the relative importance of attribute j defined by user i.

Finally, the weighted degrees of satisfaction wds have to be aggregated in order to derive the overall degree of satisfaction. Analogously, this is done by the axiom

$$P_{=}(\mathsf{satisfies} \circ \mathsf{degree}, \sum a_j \circ \mathsf{wds} \circ \mathsf{degree}) \tag{7}$$

where a_j refers to the jth attribute.

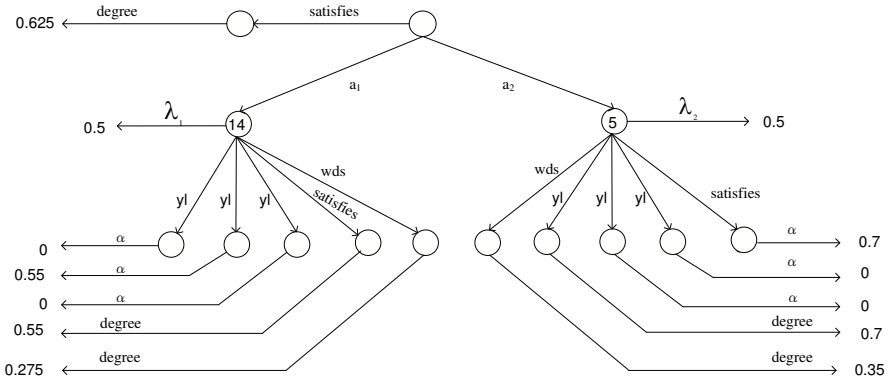


Fig. 5. Example

Example For this example the privacy policy used in section 2.1 is modified by introducing a piecewise linear function μ as follows: $\{(0, 1), (10, 0.75), (20, 0.25), (30, 0)\}$. That means the best alternative for the customer is realized when the provider does not store her private data. Consequently, the utility decreases with the number of days the private data is stored. After 10 days only 75% and after 20 days only one quarter of the overall utility remains. Beginning with a storage time of 30 days no utility can be derived from the property any more. The four points defined above result in a function containing three lines. This obviously leads to three relation instances yl in YL. For a service that stores data for 14 days the relation to all three instances of YL has to be calculated in order to derive the utility values α. Now, according to equation 6 the degree of satisfiability can be determined by aggregating the α-values. For our example, this calculation results in a degree of $0 + 0.55 + 0 = 0.55$. Analogously, the degree of satisfiability can be calculated for the attribute ResponseTime. We assume this results in a degree of 0.7 and that quality and privacy are equally important to our user ($\lambda_1 = \lambda_2 = 0.5$).

Now, the weighted degree of satisfaction wds for an attribute can be calculated by multiplying degree with the corresponding weighting factor. This results in a wds of $0.55 * 0.5 = 0.275$ for the privacy policy and $0.7 * 0.5 = 0.35$ for the quality policy. Consequently, the overall degree of satisfaction will be $0.275 + 0.35 = 0.625$.

3 Related Work

Many policy languages such as WS Policy, WS Security, EPAL, XACML, and others emerged in the Web Service community. Our work differs from these languages in that we base on a formal and extensible conceptual model. Furthermore, the WS* languages base on discrete reasoning or only vaguely define the semantics. Like our work, KAoS [12] and Rei [13] are also based on formal ontologies. In contrast to KAoS and Rei our work is currently restricted to obligations. Other modalities like permissions are not supported yet. However, both do not aim at unifying policy languages via foundational ontologies and apply a discrete reasoning approach that allows for boolean decisions only. For instance, those languages are suitable for deciding if a service is suitable according to specific policy, but make no statement about the degree of suitability. Furthermore, ontology-based policy languages often lack support for aggregation functions. This is tackled in our approach by relying on an expressive description logic ($\mathcal{ALC}(\mathcal{D}+\Sigma)$).

Moreover, in contrast to this work existing policy languages do not allow for expressing preference relations between different service configurations (e.g. between different privacy or quality levels) as well as weighting factor for the service properties. But this is necessary realizing beneficial trade-offs in a multi-attributive environment. However, some allow to assign priorities to individual policies or rules, which is not yet possible using our approach.

[14, 15] suggest to use utility functions in order to express policies and facilitate negotiation. However, they present no formal model for representing such utility information in a declarative, machine understandable, and interoperable way. But this is required to enable automatic negotiations in a distributed and heterogeneous environment. Therefore, we suggest an approach where utility information is represented by means of OWL-DL. This allows reasoning over preference information by means of standard inference engines.

Moreover, there are already existing approaches for policy based negotiation in the Semantic Web Service domain. Since deriving accurate as well as complete descriptions of Web Services is hardly manageable due to the information volume needed and the dynamic aspects that might require continuous updates of the service description, [16] introduces a contracting step where abstract service descriptions are concretized by means of individual negotiations. This procedure aims to find suitable services while keeping the descriptions simple and thus manageable. The work in this paper is complementary since we focus on the selection of the most suitable service while assuming that the set of suitable services are discovered already in the matchmaking phase before the negotiation. In [17] functional goals as well as policies are considered to find compatible services. In doing so delegation as well as trust negotiation play a crucial role, i.e. trust between two parties increases with each negotiation step. However, both approaches mentioned above allow only to derive a pure boolean statement about the compliance between policies. For selection as well as negotiation more fine-grained information about the degree of compliance might be necessary, e.g. in order to rank services or generate a counteroffer.

4 Conclusion

In this paper, we considered electronic markets as big picture and motivated the need of formal specification of policies in the allocation phase. We presented a technique for formally specifying user preferences for Web service properties and how ranking for Web services can be calculated based on such preferences. Our policy description framework is based on the foundational ontology DOLCE and thus facilitates easier integration of other policy specification languages.

In section 2.1, we have used the standard DOLCE satisfiability relation, which is in our opinion too weak. In section 2.2, we extend the satisfiability relation in a way such that one can talk about the degree of satisfiability. We have shown, how the degree of satisfiability can be calculated by a $\mathcal{ALC}(\mathcal{D}+\Sigma)$ reasoner. Note, that the description logic $\mathcal{ALC}(\mathcal{D}+\Sigma)$ is undecidable, whereas the description logic $\mathcal{ALC}(\mathcal{D})$ is decidable [18]. To be able to model the preferences with $\mathcal{ALC}(\mathcal{D})$, we only need to fix the maximum number of attributes (cf. equation (1)) and the maximum number of points in the utility function.

As discussed above, in order to enable agents to negotiate automatically without human intervention they require very detailed information about the preferences of users (e.g. utility functions for all attributes or attribute combinations, weights of the attributes). This leads to a considerable modeling effort, which obstructs the practical applicability. Thus, means have to be found to support and partly automate preference specification. Since our approach is based on utility theory we can rely on a substantial quantity of decision analysis and preference elicitation tools [19] that are already available in this context like the Analytic Hierarchy Process (AHP) [20] or a conjoint analysis.

Acknowledgements

This work was funded by the Federal Ministry of Education and Research (BMBF), the German Research Foundation (DFG), and the European Union in scope of the Internetökonomie project SESAM, the Graduate School Information Management and Market Engineering as well as the IST project SEKT.

References

1. Hurwicz, L.: The design of mechanisms for resource allocation. American Economic Review **69** (1973)
2. Lindemann, M.A., Schmid, B.F.: Elements of a reference model for electronic markets. In: HICSS '98: Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences-Volume 4. (1998)
3. Ströbel, M., Weinhard, C.: The montreal taxonomy for electronic negotiations. Group Decision and Negotiation **12** (2003) 143–164
4. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A., Schneider, L.: The WonderWeb library of foundational ontologies. WonderWeb Deliverable D17 (2002)
5. Lamparter, S., Eberhart, A., Oberle, D.: Approximating service utility from policies and value function patterns. In: 6th IEEE Int. Workshop on Policies for Distributed Systems and Networks, IEEE Computer Society (2005)
6. Staab, S., Studer, R.: Handbook on Ontologies. Springer Verlag, Heidelberg (2004)

7. Gangemi, A., Borgo, S., Catenacci, C., Lehmann, J.: Task taxonomies for knowledge content. Metokis deliverable d07 (2004)
8. Gangemi, A., Sagri, M.T., Tiscornia, D.: A constructive framework for legal ontologies. In Benjamins, R., Casanovas, P., Breuker, J., Gangemi, A., eds.: Law and the Semantic Web: Legal Ontologies, Methodologies, Legal Information Retrieval, and Applications. (2005)
9. Keeney, R.L., Raiffa, H.: Decisions with Multiple Objectives: Preferences and Value Tradeoffs. J. Wiley, New York (1976)
10. Russel, S., Norvig, P.: Artificial Intelligence - A Modern Approach. second edn. Prentice Hall Series in Artificiall Intelligence (2003)
11. Baader, F., Sattler, U.: Description logics with concrete domains and aggregation. In Prade, H., ed.: Proc. of the 13th European Conf. on AI (ECAI-98), John Wiley & Sons Ltd (1998)
12. Uszok, A., Bradshaw, J.M., Jeffers, R., Tate, A., Dalton, J.: Applying KAoS services to ensure policy compliance for semantic web services workflow composition and enactment. In: Int. Semantic Web Conf. (ISWC'04). (2004)
13. Kagal, L.: A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments. PhD thesis, University of Maryland, Baltimore MD 21250 (2004)
14. Kephart, J.O., Walsh, W.E.: An artificial intelligence perspective on autonomic computing policies. In: Proc. of 5th IEEE Int. Workshop on Policies for Distributed Systems and Networks. (2004)
15. Karp, A.H.: Representing utility for automated negotiation. Technical Report HPL-2003-153, HP Laboratories Palo Alto (2003)
16. Lara, R., Olmedilla, D.: Discovery and contracting of semantic web services. In: W3C Workshop on Frameworks for Semantic in Web Services, Innsbruck, Austria (2005)
17. Olmedilla, D., Lara, R., Polleres, A., Lausen, H.: Trust negotiation for semantic web services. In: 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC). Volume 3387 of Lecture Notes in Computer Science., San Diego, CA, USA, Springer (2004) 81–95
18. Baader, F., Hanschke, P.: A schema for integrating concrete domains into concept languages. In: Proc. of the 12th Int. Joint Conf. on AI (IJCAI-91), Sydney (1991)
19. Chen, L., Pu, P.: Survey of Preference Elicitation Methods. Technical report (2004)
20. Saaty, T.L.: How to make a decision: The analytic hierarchy process. European Journal of Operational Research **48** (1990) 9–26

Towards a Policy-Aware Web

Vladimir Kolovski[1], Yarden Katz[2], James Hendler[2],
Daniel Weitzner[3], and Tim Berners-Lee[3]
kolovski@cs.umd.edu, yarden@umd.edu, hendler@cs.umd.edu,
djw@w3.org, timbl@w3.org

[1] Maryland Information and Network Dynamics Laboratory Lab, University of Maryland , College Park , MD 20740
[2] Dept. of Computer Science, University of Maryland, College Park, MD
[3] CSAIL, Massachusetts Institute of Technology, Cambridge, MA

Abstract. In this paper, we argue that a new generation of Policy-Aware Web (PAW) technology can hold the key for providing open, distributed and scalable information access on the World Wide Web. Our approach provides for the publication of declarative access policies in a way that allows transparency for information sharing among parties. In addition, greater control over information release can be placed in the hands of the information owner, by employing rule-based discretionary access control. In this paper, we outline the kind of reasoning support which is needed to achieve these goals. Also, we present our initial steps in this direction. Our example application for this purpose is a calendar and photo sharing web site that uses a distributed policy framework (REIN) built on top of a rule-based reasoner (CWM).

1 Introduction/Motivation

While Semantic Web (SW) technologies continue to increase in popularity and scope, certain data owners are still reluctant to make their data public. The reluctance to share information prevents the full effects and benefits of Semantic Web data and tools. The main reason behind the reluctance is the lack of sufficiently sophisticated security and access control. Another issue is privacy; with so much information electronically available and increasingly sophisticated data aggregation techniques, individual privacy might be compromised and liberties put at risk if the information were to be made public with no qualifications. The problem is exacerbated when information must be shared between parties that do not have information sharing policies, or when the components of the data shared are not finely grained (in the cases where access control operates on the level of an entire website or data resource, for example.)

Consider the task of sharing photographs with friends. When sharing a picture with a friend or colleague offline, the context of the interaction and the nature of the information being shared are enough for both parties to understand the social rules governing how the information can be used. When we email pictures, many of the same social conventions are likely to apply. However, when

sharing pictures with informally-defined online communities and unknown audiences, problems soon emerge. The inability to write even very simple rules for specifying who can view this information on the Web leaves us with an inflexible set of choices. We must either share with everyone, share with no one, or engage in the arduous task of managing access via domain-specific security mechanisms (like IP address filtering, or distribution of user names and passwords.)

In this paper, we will argue that a new generation of Policy-Aware Web technology can hold the key for providing open, distributed and scalable information access on the World Wide Web. Our approach provides for the publication of declarative access policies in a way that allows significant transparency for sharing among partners without requiring pre-agreement. In addition, greater control over information release can be placed in the hands of the information owner, allowing discretionary access control to flourish. We will show that it is possible to deploy rules in a distributed and open system, and to produce and exchange proofs based on these rules in a scaleable way. These techniques, properly applied by taking crucial Web architecture issues into account, will extend Semantic Web technology to allow information resources on the World Wide Web to carry access policies that allow a wide dissemination of information without sacrificing individual privacy concerns.

In the next section, we describe in more detail the rule-based infrastructure that we are bulding. Section 3 discusses the kind of reasoning support needed for successful operation of our system. In Section 5, we describe a calendar/photo sharing web application that embodies the core ideas of PAW. Finally, we discuss the related work and future directions of our project.

2 Infrastructure

Most Web access today is performed using identity- and role-based approaches. A disadvantage of these approaches is that the roles (classes of users with specific access rights) have to be defined in advance. Setting up a temporary class for new, unforeseen access rights, in most implementations, difficult. Also, it is difficult to setup these access schemes in a fine-grained way because most web-based access generally works at the file-directory level. Our goal is to be able to describe policies at the level of individual URIs, grounding the system in the smallest externally nameable Web resources.

We are employing a rule-based approach, where a declarative set of rules is used to define fine-grained access to resources. In this framework, requesters are asked to provide a proof showing that they satisfy the policy encoded in the rules. The complexity of the required proof can vary depending on the application and domain.

There are two types of rule-based access mechanisms: Mandatory access control (MAC) and Discretionary access control (DAC). MAC systems usually do not allow the owner of the information to control protection decisions - the system is designed to enforce *a priori* protection decisions, such as enforcing the security policy over the wishes or intentions of the object owner.

Discretionary Access Control, on the other hand, allows the information owner to locally determine the access policy. The problem with DAC is that it generally requires the information requesters to prove that they have access, and for the objects in the system to have a finer-grained control of the information than in MAC systems. The system we're trying to build is of the discretionary, rule-based access type - thus, our work addresses the DAC challenges.

For an illustration of discretionary access control, consider the following example of a Girl Scout troop photo sharing. In this particular Girl Scout organization, pictures of the events can only be seen by Girl Scouts or their parents. Mary is a Girl Scout and her mother Jane wants to look at the pictures. In order to access these photos, Jane has to prove two things: she is a parent of Mary and that Mary is a girlscout. Our goal is to provide a general framework for providing those proofs. This framework will consist of a proof-exchange mechanism and also a way by which the system receiving the proof can check is correctness. On the Web, we cannot assume that every user will employ the same proof-checking software, so a set of standards is required be sure that all participants evaluate proofs on the semantic web in a consistent manner.

The central piece of the PAW project is the rules language that we are using for representing access policies and the inference engine that handles the proof checking. Since we are building on already existing Web standards, we need a rules language that can be serialized in a form where rules can be published, searched, browsed and shared using HTTP. Our language of choise is Notation 3 (N3), RDF-based rule language that was designed to be consistent with a number of Web Architecture principles. N3 is also designed to work closely with Closed World Machine (CWM), a forward-chaining reasoner that can be used for querying, checking, transforming and filtering information on the Web. Its core language is RDF, extended to include N3 rules. More detail and examples are available in the next section.

It is important to note one distinct feature of our approach. The current use of CWM requires the bulk of the reasoning to be done on the server's side. We are working on a system that is more scalable because CWM (or a CWM-like reasoner) will be used to generate proofs on the *client* side, so the only job of the server will be to check the proof to see if its grounded and consistent. Also, the authentication used by the client is not the key feature of our work (for example we might use transparent distributed authentication like openID [4]). The main contribution of our project will be combining the authentication, the proof generation on the client side and server-side proof checking to achieve a truly transparent, policy-aware web.

By shifting the burden of finding the access justification to the requesting party (and leaving only the task of checking the justification to the authorizing party), the resulting system (a) has a much smaller trusted computing base (only the part that verifies justifications) and (b) is much more transparent: any third party can audit that the justification is valid.

3 Reasoning Support for PAW

Reasoning is a critical issue when dealing with policies. Our recent work [11] shows the usefulness of having policies written in a formally defined language, OWL-DL in specific, for which there are reasoners available. The generic services of an OWL-DL reasoner were capable of capturing key tasks related to policies, such as policy containment ("If I meet policy X, do I also meet policy Y?") and detection of contradictory policies ("Is it impossible to meet policy X?"), among others.

Rule-based languages have also played a dominant role in reasoning on the Semantic Web, the primary example being N3. In the space of policies, the REIN language, based off of N3, was proposed in [10] as a generic framework for representing rule-based policies. In this section, we focus on the kind of inference licensed by languages like N3 and REIN. We sketch a brief outline of requirements from policy-aware inference engines that work with these languages, based on our preliminary work. Lastly we offer some directions in which future work may follow.

3.1 Rules, policies and proofs

Rules are desirable on the Semantic Web, and policies are no exception. The numerous proposals for rule languages offered recently suggest that rules are desirable due to their expressivity, and in some cases, because of their attractive computational properties. The latter is particularly important in PAW, where inference will be performed over a large number of policies, and will be addressed in a later section.

Once a policy is expressed as a set of rules, several challenges occur. A key challenge, and one which PAW reasoners must address, is *proof checking*. Abstractly, consider the following: agent B would like agent A to reveal certain information.[4] A has a publicly available policy (a collection of rules) P which mandates who is allowed to view the information B wishes to receive–B must meet this policy.

The simplistic proof checking scenario might then go as follows. B presents a proof Π (a sequence of assertions) to A, which is intended to establish B's eligibility to receive A's information. That is, the set of assertions $\pi_1, ..., \pi_{n-1} \in \Pi$ are supposed to justify the last line in Π ($\pi_n \in \Pi$), i.e., the assertion that B is authorized to view the information. A naive proof checker might check that $\{\pi_1, ..., \pi_{n-1}\} \cup P \vdash \pi_n$. The use of "proof" in this context of a forward-chaining is just a set of assertions, and nothing more. A proof will be a successful one (allow B access to information of A) if it (the set of facts) plus the shared policy rules between B and A entail B's granted authorization to access A's documents. Note that under the assumption that both A and B can read and process policies

[4] Note that we are assuming that these agents are acting on behalf of users. In other words, we assume that it is possible for an agent to provide information about the person for which it is performing a task if the person wishes to share this information.

written in the same language (say, REIN), there should be no disagreement about what statements are provable. Thus, determining a valid assertion in the deductive sense is trivial; simply run the rules by your own trusted reasoner and see whether the client's reported consequences follow. However, there can certainly be disagreement about the truth of the premises used in a proof, as we will now see. Suppose that A's specific policy is the following: an agent can access our information if (i) it is affiliated with the MIND lab institution, and (ii) the information is requested on a workday. Furthermore, one meets (i) just in case one's provided email address is a MIND lab email address, which take on a specific form. The "truth of the premise" disagreement is solved by having the client and server agree (via public key signing) on the grounded assertions, at which point it is sufficient to rely on deductive validity.

Policy P in N3

R_1: Conditions for being an authorized agent

```
{?agent policyP:hasMINDAcct ?email.
 ?agent policyP:requestsAccess ?request.
 ?request policyP:requestTime ?time.
 ?time a policyP:validRequestTime.}
  => {?agent a policyP:authorizedAgent.}
```

R_2 : Conditions for having the proper email credentials

```
{?agent foaf:mbox ?email.
 ?email string:matches ".*?@mindlab.umd.edu$".}
  => {?agent policyP:hasMINDAcct ?email}.
```

R_3 : Conditions for the times when a request can be processed

```
{?agent policyP:requestsAccess ?request.
 ?agent policy:requestTime ?time.
 "" time:localTime ?localTime.
 ?localTime time:dayOfWeek ?day.
 ?day math:greaterThan "0".
 ?day math:lessThan "6".
  => {?time a policyP:validRequestTime.}
```

Table 1. Policy P

In the case of policies, the naive approach is not sufficient. Certainly, there can be "proofs" presented to B which fail the naive proof checking test immediately. For example, if the triple `:AgentB foaf:mbox "bob@umbc.edu".` is part of Π, then rule R_2 will not fire, preventing us from concluding that B has the proper email address. In this case, the policy will not be satisfied. The same holds true for the case where the email address given is syntactically proper by our rule, i.e. ends with "mindlab.umd.edu", but simply does not exist.

One might object that in light of the requirement that policies be open, an agent can simply *falsify* the necessary triples satisfy the antecedents of every rule in the policy, thus tricking the naive proof checker. Let us consider such a case. Imagine B were a malicious agent, and used the valid address "bernardo@mindlab.umd.edu" without permission. Logically, this is sufficient to derive that B satisfied the email requirement. But this is clearly not enough; in a case like this, A must rely on an additional *callback confirmation* to B to guarantee the truth of the premise `:AgentB foaf:mbox "bernardo@mindlab.umd.edu"`. If B is not truly representing the individual to which this email address belongs, the callback will fail and the policy will be unsatisfied.

Notice that in both malicious scenarios, the work done to ensure the authenticity of the proof was split between the client and the server. Traditional authentication mechanisms would generally overload the server-side. For example, in the case of a falsified email, the traditional approach might require the server to contact an official server (the MIND lab server in our case) to verify the existence of the email address. The callback approach instead requires the client to do work in the process of a request (answering the callback), thus preventing potential denial-of-service attacks from the client, and relaxing the commitment to a trusted third-party server such as the university.

3.2 Engineering for scalability

For the goal of scalability, several insights from research into rule-based expert systems in the AI community are relevant. A primary example is the Rete algorithm, developed by Forgy [5], which allows for efficient processing of very large rule bases. While other approaches may be equally relevant, the use of Rete can prove useful in the context of rule-based policies for the following reasons: (i) policies can be quite complex (composed of many rules) while facts sent interactively by client may only trigger a small number of rules[5] and (ii) there will naturally be policies that are commonly used by a certain source (i.e. a company's policy to serve sensitive documents to its employees working from a remote location) and implementations of Rete lend themselves to optimization of such a use. For example, the Rete network, once constructed for a given large policy, can be either kept persistent memory, or stored locally and reloaded. The Rete network corresponding to a policy need not be rebuilt two separate client interactions.

Pychinko [13] is a forward chaining rule engine, written in Python, that implements Rete for such purposes. Its rules are expressed in the N3 language and its facts as RDF triples. The use of Rete allows it to scale far better than a similar and more popular rule engine, CWM [15], the engine supporting the REIN policy

[5] This is the case where the efficiency of the Rete algorithm shines: many rules and a newly added fact. A naive algorithm would be forced to check this newly added fact against all the rules for a possible match, while the Rete will channel the fact only to the relevant rules (i.e. rules that might be fired due to the addition of this new fact)

language, which uses a naive rule processing algorithm. It is important to note that while forward-chaining rule engines such as these add facts to the server's knowledge base (in order to determine whether's a clients authorization follows from its submitted proof), they do not in any way alter the state of the protocol interactions. The application of the reasoner is merely a server-side computation ordinarily performed in other stateless protocls.

4 Initial Results: Calendar/Photo Sharing Application

As an initial prototype, we have developed a calendar sharing application to illustrate how the rule-based policy infrastructure can be used. The application consists of a calendar view that displays events and photos associated with these events. Access to every event and photo is controlled by rules specified in N3. All of the calendar data is stored in an ontology and the policies that control access to it are in separate files. We are employing finest level of granularity - policies can be specified down to a single calendar event or a picture. REIN is used to connect the policies, the calendar data and the reasoner.

Whenever a user (who may or may not be authenticated) attempts to browse the calendar, the application first retrieves all of the events in the calendar view, filters them through the REIN policies associated with them and returns the accessible ones back to the user [6].

As an example of how policies are defined in the calendar application, consider the calendar sharing rules of a student, member of Mindswap research group at UMD). The student would like to specify that his public calendar can be seen by any member of Mindswap, but his work calendar can only be seen by his advisor.

In the first release, authentication is performed by an account-based mechanism where individuals in the ontology are mapped to user names. Our primary goal for the prototype was to use a distributed policy framework (REIN) as a resource manager to demonstrate discretionary access control on the web. A distributed authentication mechanism will be added in a future release.

5 Related Work

We are certainly not the first group to tackle access control on the web [7–9, 12, 14, 16]. In this section we will discuss two projects that have provided the most direct influence to our work.

Proof-carrying Authorization (PCA) and related distributed proof systems [2, 1] is an authorization framework that is based on a higher-order logic. We have built our work on top of theirs (ideas of a client generating and server checking proofs, integrating the proof exchange protocol into HTTP, etc.) The higher-order logic (AF logic) used to check the proofs is undecidable, though this problem is avoided by forcing clients to generate proofs on their own, using only

[6] A demo is available at http://www.policyawareweb.org/2005/calendar/

Any member of Mindswap can access events belonging to the public calendar
`@forAll :x, :y.`
`{ ?x a ont:User.`
`ont:Mindswap ont:hasMember ?x.`
`?y a ical:Vevent.`
`ont:PrivateCalendar ont:hasEvent ?y.`
`} => { ?x reine:canAccess ?y}.`

Any professor, member of Mindswap can access events belonging to the work calendar (telecons, etc.)
`@forAll :x, :y.`
`{ ?x a ont:Professor.`
`ont:Mindswap ont:hasMember ?x.`
`?y a ical:Vevent.`
`ont:WorkCalendar ont:hasEvent ?y.`
`\} => {?x reine:canAccess ?y}.`

Table 2. Calendar sharing policies in REIN P

a decidable subset of AF logic. Consequently, the authorizing server's task of proof-checking is reduced to a tractable type-checking problem - but this leads to large rate of increase of sizes of the client proof. We are currently in the process of defining our own logic for the task of representing policies, by formalizing key concepts in REIN and N3.

Bonatti et al. [3] discuss a uniform formal framework to formulate and reason about both service access and information disclosure constraints on the web. It introduces the ideas of servers publishing their policies as sets of rules and allowing usage of uncertified declarations in the client proof. Related to [3], PeerTrust [6] also deals with discretionary access control on the web using semantic web technologies. PeerTrust is a language for policies and trust negotiation, and this paper describes their implementation of implicit registration and authentication that runs over a Prolog engine. The work differs from ours in a couple of aspects. First, the authors make the assumptions that information owners are not willing to freely share their access control rules on the web, where one of the main postulates of the PAW project is transparency. Also, the trust negotiation protocol in [6] is keeping state, whereas our proof exchange protocol is stateless.

6 Conclusion and Future Work

In this paper, we have shown the need for a policy-aware infrastructure for the web, and shown our work towards this infrastructue. We have described our current work in developing a rule-based policy management system that can be deployed in an open and distributed manner on the World Wide Web. We have also shown in a demo application how it is possible to combine a Semantic Web rules language (N3), a theorem prover designed for the Web (CWM), and

a distributed policy management system (REIN) to provide discretionary access control for users. However this is still preliminary work and there are a lot of open issues, just to name a few:

- An important issue is the proof generation/checking on the client side. Future plans on this front include extending the Pychinko engine with proof checking capabilities that are sensitive to CWM's builtins.
- The party that requests access to data has to provide authentication as part of its proof. We have been experimenting with email callback mechanisms as means of proving identities, however this approach does not seem to generalize well. Other interesting mechanisms we are exploring are distributed authentication systems such as OpenID [4]
- In an open system such as the Web inconsistencies are inevitable - most logic-based systems built to date, however, are intolerant of inconsistencies. Developing a model where inconsistency can be tolerated, and kept from causing harm, is one of the key areas of research in our work.

References

1. L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access-control systems. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 81–95, Washington, DC, USA, 2005. IEEE Computer Society.
2. L. Bauer, M. A. Schneider, and E. W. Felten. A proof-carrying authorization system. Technical Report TR-638-01, Princeton University, 2001.
3. P. A. Bonatti and P. Samarati. A uniform framework for regulating service access and information release on the web. *J. Comput. Secur.*, 10(3):241–271, 2002.
4. B. Fitzpatrick. Openid:an actually distributed identity system, July 2005.
5. C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 12:17–37, 1982.
6. R. Gavriloaie, W. Nejdl, D. Olmedilla, K. Seamons, and M.Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *European Semantic Web Symposium*, May 2004.
7. S. Godik and T. Moses. Oasis extensible access control markup language (xacml) version 1.1. oasis committee specification, July 2003. http://www.oasis-open.org/committees/download.php/4103/cs-xacml-specification-1.1.doc.
8. JAAS. Java authentication and authorization service (jaas). http://java.sun.com/products/jaas/.
9. S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 474–485, New York, NY, USA, 1997. ACM Press.
10. L. Kagal and T. Berners-Lee. Rein : Where policies meet rules in the semantic web. Technical report, MIT, 2005.
11. V. Kolovski, B. Parsia, Y. Katz, and J. Hendler. Representing web service policies in owl-dl. In *International Semantic Web Conference (ISWC)*, 2005.
12. N. H. Minsky and V. Ungureanu. A mechanism for establishing policies for electronic commerce. In *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*, page 322, Washington, DC, USA, 1998. IEEE Computer Society.

13. B. Parsia, Y. Katz, and K. Clark. Pychinko: Rete-based CWM clone, October 2004. http://www.mindswap.org/~katz/pychinko/.
14. M. Roscheisen and T. Winograd. A communication agreement framework for access/action control. In *SP '96: Proceedings of the 1996 IEEE Symposium on Security and Privacy*, page 154, Washington, DC, USA, 1996. IEEE Computer Society.
15. W3C. cwm - a general purpose data processor for the semantic web, August 2004. http://www.w3.org/2000/10/swap/doc/cwm.html.
16. T. Yu, X. Ma, and M. Winslett. Prunes: an efficient and complete strategy for automated trust negotiation over the internet. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 210–219, New York, NY, USA, 2000. ACM Press.

Semantic Web Framework and Meta-Control Model to Enforce Context-Sensitive Policies

Jinghai Rao and Norman Sadeh

School of Computer Science, Carnegie Mellon University
5000 Forbes Avenue,
Pittsburgh, PA, 15213, USA
{sadeh; jinghai}@cs.cmu.edu

Abstract. Enforcing rich policies in open environments will increasingly require the ability to dynamically identify external sources of information necessary to enforce different policies. In this paper, we introduce a semantic web framework and a meta-control model for dynamically interleaving policy reasoning and external service discovery and access. Within this framework, external sources of information are wrapped as web services with rich semantic profiles allowing for the dynamic discovery and comparison of relevant sources of information. Each entity relies on one or more software agents responsible for enforcing relevant privacy and security policies in response to incoming requests. These agents implement meta-control strategies to dynamically interleave semantic web reasoning, service discovery and access. This research has been conducted in the context of *my*Campus, a pervasive computing environment aimed at enhancing everyday campus life at Carnegie Mellon University though the proposed framework extends to a number of other environments (e.g. virtual enterprises, coalition forces, homeland security). Preliminary empirical results appear rather promising.

1 Introduction

As Web applications aim for increasingly high levels of sophistication and automation, there will be a growing need for enforcing complex policies whose satisfaction is not tied to predefined sources of information. An example is enforcing context-sensitive security and privacy policies, whether in pervasive computing applications or in support of virtual enterprise scenarios, coalition force scenarios or interagency collaboration in a homeland security context.. Enforcing such policies in open environments is particularly challenging for several reasons:

- Sources of information available to enforce these policies may vary from one principal to another (e.g. different users may have different sources of location tracking information made available through different cell phone operators);
- Available sources of information for the same principal may vary over time (e.g. when a user is on company premises her location may be obtained from the wireless LAN location tracking functionality operated by her company, but, when she is not, this information can possibly be obtained via her cell phone operator);

– Available sources of information may not be known ahead of time (e.g. new location tracking functionality may be installed or the user may roam into a new area).

Enforcing context-sensitive policies in open domains requires the ability to opportunistically interleave policy reasoning with the dynamic identification, selection and access of relevant sources of contextual information. This requirement exceeds the capability of decentralized trust management infrastructures proposed so far and calls for privacy and security enforcing mechanisms capable of operating external services.

We introduce a semantic web framework and a meta-control model for dynamically interleaving policy reasoning and external service identification, selection and access. Within this framework, external sources of information are wrapped as web services with rich semantic profiles allowing for the dynamic discovery and comparison of relevant sources of information. In this paper, we look more particularly at the issue of enforcing privacy and security policies in pervasive computing environments. In this context, the owner of information sources relies on one or more software agents for enforcing relevant policies in response to incoming requests. These agents implement meta-control strategies to interleave policy enforcement, semantic web reasoning and service discovery and access. This paper introduces one particular type of agent we refer to as Information Disclosure Agents (IDA), who are responsible for enforcing two types of policies: access control policies and obfuscation policies. The latter are policies that manipulate the accuracy or inaccuracy with which information is released (see section 2 for more detail). The research reported here has been conducted in the context of *MyCampus*, a pervasive computing environment aimed at enhancing everyday campus life at Carnegie Mellon University [6, 11].

The work presented in this paper builds on concepts of decentralized trust management developed over the past decade (see [3] as well as more recent research such as [1, 2, 7]). Our own work in this area has involved the development of Semantic e-Wallets that enforce context-sensitive privacy and security policies in response to requests from context-aware applications implemented as intelligent agents [6, 10]. In this paper, we introduce a significantly more decentralized framework, where policies can be distributed among any number of agents and web services. Within this framework, our meta-control architecture interleaves semantic web reasoning and web service discovery in enforcing context-sensitive privacy and security policies.

The remainder of this paper is organized as follows. Section 2 introduces a software agent architecture for enforcing privacy and security policies. Section 3 details the meta-control model based on query status information. Section 4 discusses our service discovery model. Section 5 presents our current implementation and discusses initial empirical results. Concluding remarks are provided in Section 6. Additional details on the work described in this short paper, including a detailed description of the operation of our meta-control architecture can be found in [10].

2 Overall Approach and Architecture

We consider an environment where sources of information are all modeled as services that can be automatically discovered based on rich ontology-based service pro-

files advertised in service directories. Each service is applied to policies, which are represented as rules. In this paper we focus on access control policies and obfuscation policies enforced by *Information Disclosure Agents* (IDA), though the framework we present could readily be used to enforce a variety of other policies.

An IDA receives requests for information or service access. In processing the requests, it is responsible for enforcing access control and obfuscation polices specified by its owner. As it processes requests, the agent records status information that helps it monitor its own progress in enforcing its policies and in obtaining the necessary information to satisfy the request. Based on this updated *query status information*, a meta-control module ("meta-controller") dynamically orchestrates the operations of modules at its disposal to process queries (Fig. 1). As these modules report on the status of activities they have been tasked to perform, this information is processed by a housekeeping module responsible for updating query status information.

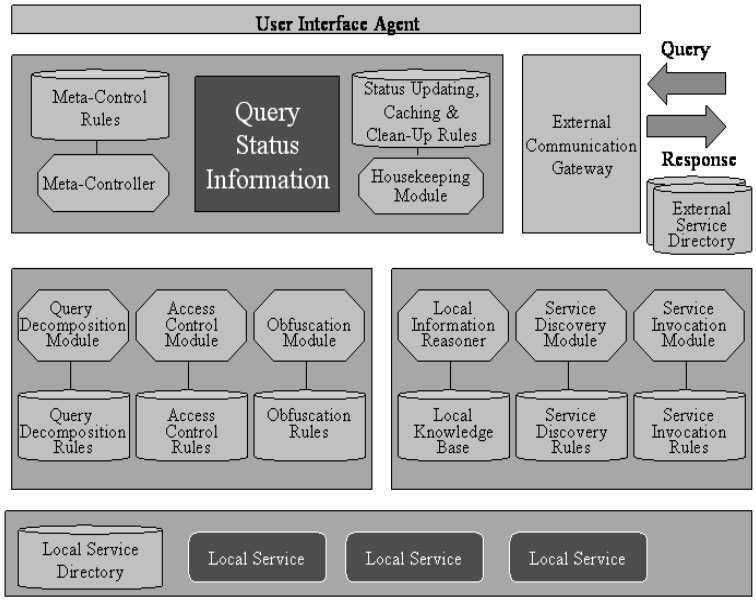


Fig. 1. Information Disclosure Agent: Overall Architecture

For obvious efficiency reasons, while an IDA consists of a number of logical modules, each operating according to a particular set of rules, it is typically implemented as a single reasoning engine. The following provides a brief description of each of the modules orchestrated by an IDA's meta-controller:

- *Query Decomposition Module* takes as input a particular query and breaks it down into elementary needs for information, which can each be thought of as subgoals or sub-queries. We refer to these as *Query Elements.*
- *Access Control Module* is responsible for determining whether a query or subquery is consistent with relevant access control policies – modeled as access control rules. While some policies can be checked just based on facts contained in the

agent's local knowledge base, many policies require obtaining information from a combination of both local and external sources. When this is the case, rather than immediately deciding whether or not to grant access to a query, the *Access Control Module* needs to request additional facts – also modeled as *Query Elements*.

- *Obfuscation Module* sanitizes information requested in a query according to relevant obfuscation policies – also modeled as rules. As it evaluates relevant obfuscation policies, this module too can post requests for additional *Query Elements.*
- *Local Information Reasoner* corresponds to domain knowledge (facts and rules) known locally to the IDA.
- *Service Discovery Module* helps the IDA identify potential sources of information to complement its local knowledge. External services can be identified through external service directories (whether public or not), by communicating via the agent's *External Communication Gateway*. The service identification rules directly map information needs on pre-specified services. We currently assume that all service directories rely on OWL-S to advertise service profiles (see Section 4).
- *Service Invocation Module* allows the agent to invoke relevant services. It is important to note that, in our architecture, each service can have its own IDA. As requests are sent to services, their IDAs may in turn respond with requests for additional information to enforce their own policies.
- *User Interface Agent:* The meta-controller treats its user as just another module who is modeled both as a potential source of domain knowledge (e.g. to acquire relevant contextual information) and a potential source of meta-control knowledge (e.g. if a particular query element proves too difficult to locate, the user may be asked whether to stop looking).

3 Query Status Model

An IDA's *Meta Controller* relies on meta-control rules to analyze query status information and determine which module(s) to activate next. Meta-control rules are modeled as if-then clauses, with Left Hand Sides (LHSs) specifying their premises and Right Hand Sides (RHSs) their conclusions. LHS elements refer to query status information, while RHS elements contain facts that result in module activations. Query status information helps keep track of how far along the IDA is in obtaining the information required by each query and in enforcing relevant policies. Query status information in the LHS of meta-control rules is expressed according to a taxonomy of predicates that helps the agent keep track of queries and query elements - e.g., whether a query has been or is being processed, what individual query elements it has given rise to, whether these elements have been cleared by relevant access control policies and sanitized according to relevant obfuscation control policies, etc. All status information is annotated with time stamps. In other words, query status information includes:

- **Status predicates** to describe the status of a query or query element
- **A query ID** or **query element ID** to which the predicate refers
- **A parent query ID** or **parent query element ID** to help keep track of dependencies (e.g. a query element may be needed to help check whether another query

element is consistent with a context-sensitive access control policy). These dependencies, if passed between IDA agents, can also help detect deadlocks (e.g. two IDA agents each waiting for information from the other to enforce their policies)

– **A time stamp** that describes when the status information was generated or updated. This information is critical when it comes to determining how much time has elapsed since a particular module or external service was invoked. It can help the agent look for alternative external services or decide when to prompt the user (e.g. to decide whether to wait any longer).

A list of query status predicates currently implemented can be found in [10]. In general, query status information is updated by asserting new facts (with old information being cleaned up by the IDA's housekeeping module). As query updates come in, they trigger one or more meta-control rules, which in turn result in additional query status information updates and the eventual activation of one or more of the IDA's modules. As already mentioned earlier, this meta-control architecture can also be used to model the user as a module that can be consulted by the meta-controller, e.g. to ask for a particular piece of domain knowledge or to decide whether or not to abandon a particular course of action such as looking for an external service capable of providing a particular query element.

The following example illustrates a meta-control rule. This rule indicates the status change after a service is invoked successfully to tell the value of the required query element. Once the service response is received, the old status "waiting-for-service-response" is cleaned, and the new status "element-available" is generated. The rule, expressed in CLIPS [4], is of the form:

```
?x <- (triple "Status#predicate" ?s1 "waiting-for-service-response")
?y <- (triple "Query#queryId" ?s1 ?service)
(triple "Status#predicate" ?s2 "service-response-available")
(triple "Query#queryId" ?s2 ?result)
=>
(retract ?x)
(retract ?y)
(assert (triple "Status#predicate" ?newstatus "element-available))
(assert (triple "Query#queryId" ?newstatus ?result))
```

4 The Service Discovery Model

A central element of our method is the ability of IDA agents to dynamically identify sources of information needed by query elements. Sources of information are modeled as semantic web services and may operate subject to their own access control and obfuscation policies enforced by their own IDA agents. Accordingly service invocation is itself implemented in the form of queries sent to a service's IDA agent. .

Each service (or source of information) is described by a *ServiceProfile* in OWL-S [9]. In general, a *ServiceProfile* consists of three parts: (1) information about the provider of the service, (2) information about the service's functionality and (3) information about non-functional attributes [12]. Functional attributes include the service's inputs, outputs, preconditions and effects. Non-functional attributes are other properties such as accuracy, quality of service, price, location, etc. An example of a

location tracking service operated on the premises of Company Y can be described as follows:

```
<profileHierarchy:InformationService rdf:ID="PositioningServ">
  <!-- reference to the service specification -->
  <service:presentedBy rdf:resource="&Serv;#PositioningServ"/>
  <profile:has_process rdf:resource="&Process;#PositionProc"/>
  <profile:serviceName Positioning_Service_in_Y />

  <!-- specification of quality rating for profile -->
  <profile:qualityRating>
    <profile:QualityRating rdf:ID="SERVQUAL">
      <profile:ratingName SERVQUAL />
      <profile:rating rdf:resource="&servqual;#Good"/>
    </profile:QualityRating>
  </profile:qualityRating>

  <profile:hasPrecondition rdf:resource="&Process;#LocateInCompanyY"/>
  <profile:hasOutput rdf:resource="&Process;#RoomNoOutput"/>
</profileHierarchy:InformationService>
```

When invoking a service it has identified, an IDA may opt to provide upfront all the input parameters required by that service or it may withhold one or more of these parameters. The latter option forces the service to request the missing input parameters from the IDA, thereby enabling the IDA to more fully determine whether the invoked service meets its policies. This option is however more computation and communication intensive.

Service outputs are represented as OWL classes, which play the role of a typing mechanism for concepts and resources. Using OWL also allows for some measure of semantic inference as part of the service discovery process. If an agent requires a service that produces as output a contextual attribute of a specific type, then all services that output the value of that attribute as a subtype are potential matches.

Service preconditions and effects are also used for service matching. For instance, the positioning service above has a precondition specifying that it is only available on company Y's premises.

5 Current Implementation: Evaluation and Discussion

Our policy enforcing agents are currently implemented in JESS, a high-performance rule-based engine in Java [5]. Domain knowledge, including service profiles, queries, access control policies and obfuscation policies are expressed in OWL [6]. As already indicated earlier, we use ROWL to define rules. XSLT transformations are used to translate OWL facts and ROWL rules into CLIPS, the rule language supported by JESS. Currently all information exchange between agents is done in the clear and without digital signatures. In the future, we plan to use SSL or some equivalent protocol for information exchange and without digital signatures. In the future, we plan to use SSL or some equivalent protocol for all information exchange. This will include signing all queries and responses.

We have evaluated our solution on an IBM laptop with a 1.80GHz Pentium M CPU and 1.50GB of RAM. The laptop was running Windows XP Professional OS, Java SDK 1.4.1 and Jess 7.0. As part of the evaluation, we implemented the example

introduced in Section 4 and 6, using a light-weight rule/fact set. The set included 22 rules and 178 facts and features a single semantic service directory with 50 services, each represented by 5 to 10 Jess rules. A breakdown of the CPU times required to process Bob's query is provided in the table below. For each module the table provides a cumulative CPU time, namely the sum of the CPU times of all invocations of that module in processing the query.

Module	CPU time in millisecond
Meta-Controller	28
Access-Controller	32
Local-KB	49
Service discovery / invocation	72
Total	181

While these results provide just one data point and only evaluate a subset of our functionality, they seem to suggest that our solution can be viewed as practical in at least some simple settings. It should be noted that our solution is not JESS-specific and could be implemented in other rule languages.

6 Concluding Remarks

In this paper, we presented a semantic web framework for dynamically interleaving policy reasoning and external service discovery and access. Within this framework, external sources of information are wrapped as web services with rich semantic profiles allowing for the dynamic discovery and comparison of relevant sources of information. Each entity (e.g. user, sensor, application, or organization) relies on one or more software agents responsible for enforcing relevant privacy and security policies in response to incoming requests. These agents implement meta-control strategies to dynamically interleave semantic web reasoning and service discovery and access. These meta-control strategies can also be extended to treat the user as another source of information, e.g. to confirm whether a given fact holds or to provide meta-control guidance such as deciding when to abandon trying to determine whether a policy is satisfied.

The Information Disclosure Agent presented in this paper is just one instantiation of our more general concept of Policy Enforcing Agents (PEAs)[10]. Other policies (e.g. information collection policies, notification preference policies) will typically rely on slightly different sets of modules and different meta-control strategies, yet they could all be implemented using the same meta-control architecture and many of the same principles presented in this paper. In general, PEAs rely on a taxonomy of query information status predicates to monitor their own progress in processing incoming queries and enforcing relevant security and privacy policies. Preliminary evaluation of an early implementation of our framework seems encouraging. At the same time, it is easy to see that the generality of our framework also gives rise to a number of challenging issues. Future work will focus on further evaluating and refining the scalability of our framework, evaluating tradeoffs between the expressiveness of privacy and security policies we allow and associated computational and communication requirements. Other issues of particular interest include studying opportuni-

ties for concurrency (e.g. simultaneously accessing multiple web services), dealing with real-time meta-control issues (e.g. deciding when to give up or when to look for additional sources of information/web services), breaking deadlocks [8], and integrating the user as a source of information.

References

[1] L. Bauer, M.A. Schneider and E.W. Felten. "A General and Flexible Access Control System for the Web", In Proceedings of the 11th USENIX Security Symposium, August 2002.

[2] L.Bauer, S. Garriss, J. McCune, M.K. Reiter, J. Rouse, and P Rutenbar, "Device-Enabled Authorization in the Grey System", Submitted to USENIX Security 2005. Also available as Technical Report CMU-CS-05-111, Carnegie Mellon University, February 2005.

[3] M. Blaze, J. Feigenbaum, an J. Lacy. "Decentralized Trust Management". Proc. IEEE Conference on Security and Privacy. Oakland, CA. May 1996.

[4] CLIPS. http://www.ghg.net/clips/CLIPS.html.

[5] E. Friedman-Hill. Jess in Action: Java Rule-based Systems, Manning Publications Company, June 2003, ISBN 1930110898, http://herzberg.ca.sandia.gov/jess/

[6] F. Gandon, and N. Sadeh. Semantic web technologies to reconcile privacy and context awareness. Web Semantics Journal, 1(3), 2004.

[7] T. van der Horst, T. Sundelin, K. E. Seamons, and C. D. Knutson. Mobile Trust Negotiation: Authentication and Authorization in Dynamic Mobile Networks. Eighth IFIP Conference on Communications and Multimedia Security, Lake Windermere, England, 2004

[8] T. Leithead, W. Nejdl, D. Olmedilla, K. Seamons, M. Winslett, T. Yu, and C. Zhang, How to Exploit Ontologies in Trust Negotiation. Workshop on Trust, Security, and Reputation on the Semantic Web, part of ISWC04, Hiroshima, Japan, November 2004.

[9] OWL-S: Semantic Markup for Web Services, W3C Submission Member Submission, November 2004. http://www.w3.org/Submission/OWL-S

[10] J. Rao and N.M. Sadeh. Interleaving Semantic Web Reasoning and Service Discovery to Enforce Context-Sensitive Security and Privacy Policies. Carnegie Mellon University Technical Report (CMU-ISRI-05-113), July 2005. http://www-2.cs.cmu.edu/~sadeh/Publications/More%20Complete%20List/techreport%20%20july%2027%202005.pdf

[11] N.M. Sadeh, F. Gandon, and Oh Byung Kwon. Ambient Intelligence: The MyCampus Experience. Carnegie Mellon University Technical Report (CMU-ISRI-05-123). June 2005.

[12] J. O'Sullivan, D. Edmond, and A.T. Hofstede. What's in a service? Towards accurate description of non-functional service properties. Distributedand Parallel Databases, 12:117.133, 2002.

Acknowledgements

The work reported herein has been supported in part under DARPA contract F30602-02-2-0035 ("DAML initiative") and in part under ARO research grant D20D19-02-1-0389 ("Perpetually Available and Secure Information Systems") to Carnegie Mellon University's CyLab. Additional support has been provided by IBM, HP, Symbol, Boeing, Amazon, Fujitsu, the EU IST Program (SWAP project), and the ROC's Institute for Information Industry.

Towards Integrated Specification and Analysis of Machine-Readable Policies Using Maude[1]

Rukman Senanayake, Grit Denker, and *Jon Pearce

SRI International, Menlo Park, CA 94025, *San Jose State University, CA 95112
rukman@csl.sri.com, Grit.Denker@sri.com, *pearce@cs.sjsu.edu

1. Introduction and Objectives

A policy is defined as "a plan or course of action, as of a government, political party, or business, intended to influence and determine decisions, actions, and other matters." The behavior of many computerized systems and applications is determined by policies. For such systems it is useful not only to specify policies in a machine-readable way, as provided by semantic markup languages such as OWL[2] and SWRL[3], but also to analyze whether a policy meets the intention of the policy designer. The latter can be achieved by translating policies into an existing specification and analysis framework and providing check lists, such as reachability of certain system states while complying with the policy or consistency checks among sets of policies, to increase the trust of the policy designer that the formalized policy captures her intention.

We propose to investigate the use of the rule-based framework Maude[4] for the analysis of policies. The advantage of existing rule-based formal frameworks is that they usually already support a variety of analysis tasks that can be adapted to the task at hand. We believe that a rule-based system is adequate since many policies satisfy the following two characteristics: 1. Often natural language policy uses "*If-Then-Else*" statements and, thus, can be naturally formalized as rules. For example, a return policy for an online purchase service may state "If an item is returned within 30 days, then full refund is given." 2. Often policies define constraints on the process or behavior of a system that implements the policy.

For example, the policy "First it is determined whether an item is returnable, next the exact refund amount is calculated" defines constraints in the sense that a certain sequence of state transitions or actions must be followed. Thus many policies can be formalized as a collection of rules and often policies abstractly define the behavior of a system in terms of a state transition system where rules govern how a state changes. This is also the case in one of our projects, the DARPA XG project[5], in which policies define the behavior of wireless radios. In the XG project policies are defined using OWL extended by a rule mechanism similar to SWRL.

---

[1] Supported by the Defense Advanced Research Projects Agency through Air Force Research Laboratory under Contracts FA8750-05-C-0230 and F30602-00-C-0168.

[2] Web Ontology Language: http://www.w3.org/2001/sw/WebOnt/

[3] A Semantic Web Rule Language Combining OWL and RuleML: http://www.daml.org/2003/11/swrl/

[4] The Maude System: http://maude.csl.sri.com

[5] DARPA Next Generation (XG) project: http://www.darpa.mil/ato/programs/xg/

Thus, in this paper we investigate by means of example the adequacy of Maude to express OWL/SWRL policies and use the built-in analysis techniques for Maude. As shown in this paper, our approach seems feasible, and future work must investigate the existence of a formal translation from OWL/SWRL into Maude to show that inference in OWL and SWRL can be reduced to inference in Maude.

In the remainder of this paper we will introduce a small example that we will use for illustration purposes (Section 2), briefly overview the Maude framework (Section 3), apply the framework to the example (Section 4), illustrate the translation of SWRL policies to Maude (Section 5), explain the various analysis tests that can be executed in Maude on the policy (Section 6), and give a brief overview of related work and close with a brief summary in Section 7.

2. Policy Specification and Semantics

2.1. A Policy Example

We selected a real-world policy as our primary example, namely, the return policy of Amazon. Amazon uses a 'wizard-like' approach to guide a user through the return process for an item sold to a customer. The policy as a whole can be described by a collection of rules and a set of states through which the process transitions. The rules define transitions between states, and the user needs to provide certain information, such as "whether the item was opened or not" to determine the initial state of the system. Table 1 summarizes some of the rules applicable to Amazon's return policy.

Rule	Description
1	Partial refund if item is returned after 30 days
2	Item is nonreturnable if stated in Product Description Page (PDP)
3	Partial refund if item is a book and has signs of usage
4	If there was a shipping error, Amazon pays return cost

Table 1. Example Rules of Amazon's Return Policy.

Translating the Amazon return policy into a state transition system, we get seven states and transitions as depicted in Figure 1.

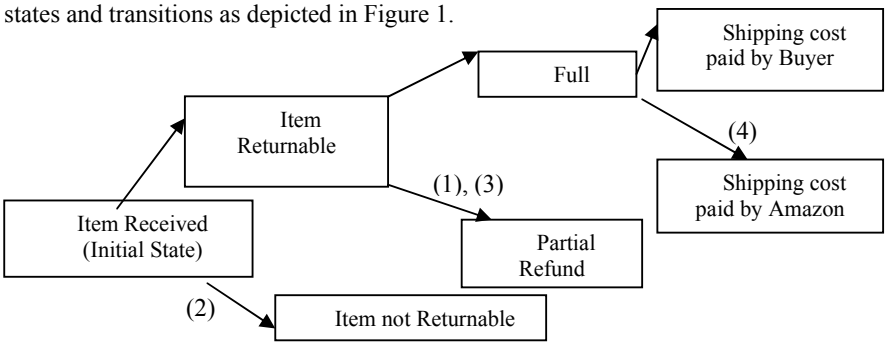


Fig. 1. State Transition System for Amazon's Return Policy. Labels on arrows refer to rules that enable state transition. Unlabeled arrows are default transitions.

2.2. Policy Specification

The return policy was specified using the OWL and SWRL semantic markup languages. OWL was used to define the attributes, classes, and instances that collectively define a state space necessary for the return policy, and SWRL was used to define the rules of the policy.

We used the Protégé Ontology editor to encode the policies, as defined in Table 2. The class taxonomy contains classes such as 'Manual' which is a subclass of 'Book' which in turn is a subclass of 'Item'. The class 'Boo' has datatype properties 'isUsed' and so on. The rule base for the policy was specified using SWRL.

Rule	SWRL Encoding of the Rule
2	Item(?x) ∧ hasPDP(?x, "Can not be returned") → isReturnable(?x, false)
3	Book(?x) ∧ isUsed(?x, true) → hasRefundRatio(?x, 2)
4	Item(?x) ∧ hasCondition(?x, "Shipping Error") ∧ hasSeller(?x, ?y) → hasShippingCostPayee(?x, ?y)

Table 2. Policy Rules Defined Using SWRL.

3. The Maude System

Maude [CEL+96] is a multiparadigm executable specification language based on rewriting logic [Mes92, Mes00]. Maude sources, executables for several platforms, the manual, a primer, cases studies, and papers are available from the Maude Web site at http://maude.csl.sri.com.

We briefly summarize the syntax of Maude that is used in our case study. Maude specifies systems as collections of modules. System modules are rewrite theories specifying concurrent systems; they are declared with the syntax *mod...endm*. Immediately after the module's keyword, the *name* of the module is given. After this, a list of imported modules can be added. One can also declare *sorts, subsorts,* and *operators*.

Operators are introduced with the *op* keyword followed by the operator name, the argument, and result sorts. An operator may have mixfix syntax, with the name containing '_'s (underscores) marking the argument positions.

Equational axioms are introduced with the keyword *eq* or *ceq* (for conditional equations) followed by the two terms being declared equal separated by the equality sign *'='*. Rewrite rules are introduced with the keyword *rl* or *crl* (for conditional rules) followed by an optional rule label, and terms corresponding to the premises and conclusion of the rule separated by the rewrite sign *'=>'*. Variables appearing in axioms or rules (and commands) may be declared globally using keyword *var* or *vars,* or "inline" using the variable name and its sort separated by a colon; for

example, *n:Nat* is a variable named *n* of sort *Nat*. Rewrite rules are not allowed in functional modules.

We model the ontology described by OWL and SWRL specifications in Maude by using Maude's notation and conventions for concurrent objects. A snapshot of a system's state can be thought of as a collection of objects (a configuration). The multiset union operator for configurations is denoted with empty syntax (juxtaposition) and (by definition of multiset) is associative and commutative.

An object has the form *< O : C | att-1, ... , att-n >* where *O* is an object identifier (sort Oid), *C* is a class identifier (sort Cid), and *att-1…att-n* are attributes (sort Attribute). This notation is part of a module called CONFIGURATION, which is part of the standard Maude library.

A typical system configuration will consist of several objects. The dynamic behavior of a concurrent object system is then axiomatized by specifying rewrite rules for each class that determine, for example, how objects can evolve from one state to another through rule application.

4. Example Policy in Maude

An abbreviated version of the generated Maude module is given in Table 3.

(1)	mod ATTRIBUTES is protecting STRING . protecting BOOL . op hasOriginalCondition:_ : String -> Attribute . op isReturnable:_ : Bool -> Attribute . op hasCondition:_ : String -> Attribute . endm	The module ATTRIBUTES defines the collection of datatype properties in the OWL ontology
(2)	mod OWLONTOLOGY is inc CONFIGURATION . inc ATTRIBUTES .	The module OWLONTOLOGY defines classes and rules. The 'inc' statements are somewhat analogous to importing of URIs that define namespaces.
(3)	sort Book . op Book : -> Book . sort Item . op Item : -> Item . sort swrlAtom . op swrlAtom : -> swrlAtom .	Each class in the OWL ontology is defined as a sort in the Maude module. This includes the defining of a constructor with the same name as the class name using the *'op'* keyword.
(4)	subsort String < Oid . subsort String < Cid . subsort swrlAtom < Cid . subsort Item < Cid . subsort Book < Item . subsort Manual < Book .	The subsort definitions reflect the correct class taxonomy within Maude. All class names and object names are strings. All other classes are subclasses of the top-level class

	subsort Novell < Book .	Cid or one of its subclasses.
(5)	var atts : AttributeSet . var oid : Oid . var I1 : Int . var book : Book . var item : Item .	All variables used in the rewrite rules (see (6)) are defined.
(6)	rl [rule2] : < oid : item | hasPDP: "Can not be returned", isReturnable: B1, atts > => < oid : item | hasPDP: "Can not be returned", isReturnable: false, atts > .	Rewrite rule definitions consist of keyword *'rl'* (or *crl*), an optional name ([rule1]), the start state, and the terminating state. The example rule implements Rule 1.
(7)	op startState : -> Configuration . eq startState = < "Les Miserables" : Book | isReturnable: true, hasCondition: "Used", hasRefundRatio: 0, hasPDP: "All time favorite" > .	The initial state of the ontology is defined by the operator 'startState'. It contains instances of classes (one or more) together with their attribute values.

Table 3. Partial Maude Module for Amazon's Return Policy.

5. Translating Semantic Markup Specifications to Maude

5.1. Translating OWL Specifications to Maude

Translating the Class Taxonomy

Because of the object-oriented nature of the OWL ontology, we chose the object-oriented specification style of Maude to stay as close as possible in our translation to the original OWL/SWRL policy. The OWL-to-Maude translation process preserves the class hierarchy and this is done using the 'subsort' definitions (see section (4) of Table 3). Using the *'subsort'* keyword of Maude is semantically comparable to the OWL class taxonomy.

Some additional definitions are incorporated because of requirements within Maude, namely, the subsort definitions *'subsort String < Cid'* and *'subsort String < Oid'*. This is to support using strings as class and object names. Another important definition is *'subsort Item < Cid'*, which is analogous to the fact that every OWL named class is a subclass of the OWL:thing class (since the sort 'Cid' corresponds to the top-level class 'thing' in OWL).

Translating OWL Individuals

The collection of OWL individuals defined in the ontology is equivalent to the start state of the Maude system module. The collection of OWL individuals is retrieved and converted into a list of object definitions based on the attributes and sorts defined in the Maude model (see section (7) of Table 3). This collection of objects is the start state for the Maude rewrite logic, and is an instance of the Configuration sort. Since the Configuration sort is a multiset of objects and messages this preserves the object-oriented semantics of the OWL ontology.

5.2. Translating SWRL Specifications to Maude

The SWRL rule base is translated into a Maude rule collection. A Maude rule has a start-state definition and an end-state definition. These definitions are simply

collections of objects with certain attributes, and a rule as a whole depicts how the values of the attributes change when the rule is applied.

For example, Rule 4 of Table 2 is translated into a Maude rule by the following technique:

i. The SWRL class membership predicate is identified for the relevant class of objects that the rule affects. In Rule 4 this is equivalent to 'Item(?x)' and the Maude representation of this would be 'oid : item' where 'oid' is a variable (typically of type String) used as an object identifier.

ii. The collection of attributes in the antecedent and the consequent of the SWRL rule are retrieved and translated into equivalent attribute definitions of Maude. For example, 'hasSeller(?x,?y)' is written in Maude as 'hasSeller: Y' where 'Y' is a variable of a suitable type (typically String).

iii. The Maude rule is generated by converting the antecedent of the SWRL rule to the start state and the consequent to the end state.

The resulting Maude rule follows:

Rule 4	rl [Rule-5] < oid : item | hasCondition: "Shipping Error", hasSeller: S1, hasShippingCostPayee: S2, atts > => < oid : item | hasCondition: "Shipping Error", hasSeller: S1, hasShippingCostPayee: S1, atts >

The rule says that an item with condition "Shipping Error" can be rewritten to a state where the payee for the shipping cost is set to the seller (independent of the values in the hasShippingCostPayee before). In the above Maude rule, 'atts' is a variable representing a set of attributes a class may have. This is a simple technique used in Maude to avoid listing the entire set of attributes for every state, since 'atts' represents any subset of the attribute collection of a class, so that only the attributes affected by the current rule need to be listed.

6. Analyzing Policies Using Maude

The dynamic behavior of a concurrent object system is axiomatized by specifying rewrite rules for each type of state transition that can take place in the system. For example, a rewrite rule may define the transition into a state where an item is not returnable.

Maude [CEL+96] is not just a language. It also has an interpreter, making it an executable, multiparadigm specification language. The Maude interpreter is very efficient, allowing prototyping of quite complex test cases. In addition to modeling capabilities, Maude provides efficient built-in search and model checking capabilities. Maude is reflective [Cla98], providing a meta-level module that reflects both its syntax and semantics. Using reflection, the user can program special-purpose execution and search strategies, module transformations, analyses, and user interfaces. Maude has several built-in analysis capabilities. For example, the user can define an initial system state and use Maude to apply rules to find a final state. This is called "default execution", since one possible sequence of rules is applied. Obviously, a final state is found only for rules that describe finite-state systems or, for an infinite-state

system, when the number of rules applied is restricted by the user. Besides default execution, Maude supports state search in at least two ways.
(1) A user can define an initial system state (which can be represented in a symbolic way) and use a search strategy in Maude to determine all possible combinations of rule applications and the resulting states. If this analysis results in more than one final state for a given set of rules, it means that the system specified by the rules is not deterministic and that the order in which policy rules are applied matters. This is usually not the intent of policy designers, and in particular in a situation where several policies are combined, it is important to know whether there are such "side effects" due to rule ordering. This test can be generalized to test policy consistency. We define a strategy that finds sequences of rule applications that result in contradictory statements (e.g., refundRatio=partial and refundRatio=full).
(2) Maude also supports a search function for Linear Temporal Logic (LTL) formulas. The user can define a state for which he would like to know whether the state is reachable from a given initial state with the set of rules. Maude will find a path, if it exists. This type of analysis is useful to check the adequacy of policies with regard to the user's intuition. For example, a user might expect for a given situation a certain policy decision or outcome by applying the rules. She can check this by running the LTL model checker that is part of Maude. If the desired and expected outcome is not achieved, then the rule base does not correctly reflect the intention of the user. More generally, the user might want to test what are all possible reachable states (if a finite state system is described by the set of rules). In addition to analysis, Maude can serve as an execution engine for rules or services since any Maude specification can be efficiently executed.

7. Related Work and Concluding Remarks

Semantic Web Service (SWS) approaches (e.g., OWL-S or WSML) describe services in semantically meaningful and machine-readable ways, and thus overcome the issues of informal descriptions or natural language policies that describe a Web service's behavior. For example, the process specification of a service may say that one cannot use the service without successfully completing a login phase or that a service will always go through an order confirmation phase before charging a credit card. Our approach could be used to formalize such process constraints on a high abstraction level, and define a "refinement" or "implementation" relationship between high-level policies and a service model. In this way, one could define a semantics for service models using rewriting logic. Other approaches to give semantics to OWL-S processes are [NM02] and [AHS02]. [NM02] gives semantics to some parts of the process model in terms of Petri Nets, and [AHS02] gives an operational semantics. A Maude formalization of the OWL-S process model in terms of rules describing the state transitions would provide both a denotational and an executable semantics, since Maude specifications are efficiently executable [CDE+03] and possess a well-founded semantics based on rewriting logic [Mes92].

Recent work on machine-readable policy and rule languages (e.g., SWRL, RuleML, SWSL, Rei [KFJ05] and KAoS [UBJ+03]) enables the formal definition of policies and rules. Some of the languages have tools for analysis of policies, such as checking whether a given action or instance matches a policy or whether policies are

consistent. Rei and KAoS make use of existing technology to achieve their reasoning capabilities. Rei uses prolog-style reasoning and KAoS uses existing Description Logic reasoning. In a manner somewhat similar to the Rei and KAoS approaches, we are interested in investigating how the rule-based Maude technology can be used to help with policy and service specification and analysis.

We designed and implemented a mapping from example policies expressed in SWRL to executable Maude specifications. We support the analysis of machine-readable policies in an integrated fashion from the policy editor, using Maude's built-in analysis capabilities. This encourages us to further investigate how the Maude environment and its theoretical underpinning (i.e., rewriting logic) could be of use as a framework for OWL/SWRL policies.

References

[AHS02] A. Ankolekar, F. Huch, and K. Sycara. Concurrent Execution Semantics for DAML-S with Subtypes, First Int. Semantic Web Conference (ISWC), Sardinia (Italy), June, 2002.

[CDE+03] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, and C. Talcott. The Maude 2.0 System. In Nieuwenhus (ed). Rewriting Techniques and Applications. Springer, LNCS 2706, pp 77-87, 2003.

[CEL+96] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude, In J. Meseguer (ed), Rewriting Logic and Its Applications, First Int. Workshop, Asilomar, CA, pp 65-89, 1996.

[Cla98] M. Clavel. Reflection in General Logics, Rewriting Logic, and Maude. Ph.D. Dissertation, University of Navarre, 1998.

[KFJ05] L. Kagal, T. Finin, and A. Joshi. Rei: A Policy Specification Language. See http://rei.umbc.edu/.

[Mes92] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency, Theoretical Computer Science. 96(1):73-155, 1992.

[Mes00] J. Meseguer. Rewriting Logic and Maude: A Wide-spectrum Semantic Framework for Object-based Distributed Systems. In S. Smith and T. Talcott (eds), Formal Methods for Open Object-based Distributed Systems, FMOODS 2000, pp 89-117. Kluwer, 2000.

[NM02] S. Narayanan and S. McIlraith. Simulation, Verification and Automated Composition of Web Services, Proc. of the Eleventh International World Wide Web Conference (WWW-11), Honolulu, May 2002.

[Protégé01] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, and M. A. Musen. Creating Semantic Web Contents with Protege-2000, *IEEE Intelligent Systems* 16(2):60-71, 2001.

[UBJ+03] A. Uszok, J. Bradshaw, J. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement, Proc. IEEE Workshop on Policy, 2003.

An Integration of Reputation-based and Policy-based Trust Management

⋆ Piero Bonatti[1], Claudiu Duma[2], Daniel Olmedilla[3], and Nahid Shahmehri[2]

[1] Università di Napoli Federico II, Napoli, Italy
bonatti@na.infn.it
[2] Department of Computer and Information Science, Linköpings universitet
{cladu,nahsh}@ida.liu.se
[3] L3S Research Center and University of Hannover, Hanover, Germany
olmedilla@l3s.de

Abstract. Trust management is currently being tackled from two different perspectives: a "strong and crisp" approach, where decisions are founded on logical rules and verifiable properties encoded in digital credentials, and a "soft and social" approach, based on reputation measures gathered and shared by a distributed community. We analyze the differences between the two models of trust and argue that an integrated approach would improve significantly trust management systems. We support our claim with real world scenarios and illustrate how the two models are integrated in PROTUNE, the core policy specification language of the network of excellence REWERSE.

1 Introduction

Trust management has been an important research line in the development of modern open distributed and decentralized systems. Trust has been studied in the context of decentralized access control [5, 16], public key certification [4, 9], and reputation systems for P2P networks [2, 14, 10].

There exist currently two different major approaches for managing trust: policy-based and reputation-based trust management. The two approaches have been developed within the context of different environments and targeting different requirements. On the one hand, policy-based trust relies on objective "strong security" mechanisms such as signed certificates and trusted certification authorities (CA hereafter) in order to regulate the access of users to services. Moreover, the access decision is usually based on mechanisms with well defined semantics (e.g., logic programming) providing strong verification and analysis support. The result of such a policy-based trust management approach usually consists of a binary decision according to which the requester is trusted or not, and thus the service (or resource) is allowed or denied. On the other hand, reputation-based trust relies on a "soft computational" approach to the problem of trust. In this case, trust is typically computed from local experiences together with the feedback given by other entities in the network (e.g., users who have used services of that provider). For instance, in eBay buyers and sellers rate each other after each transaction. The ratings pertaining to a certain seller (or buyer) are aggregated by the eBay's reputation system into a number reflecting

⋆ In alphabetical order. This work is partially supported by the Network of Excellence REWERSE, IST-506779, http://rewerse.net.

seller (or buyer) trustworthiness as seen by the eBay community. The reputation-based approach has been favored for environments, such as Peer-to-Peer or Semantic Web, where the existence of certifying authorities could not be always assumed but where a large pool of individual user ratings was usually available.

The two trust management approaches address the same problem - establishing trust among interacting parties in distributed and decentralized systems. However, they assume different settings. While the policy based approach has been developed within the context of structured organizational environments, the reputation systems have been proposed to address the unstructured user community. Consequently, they assume different sources for trust (CAs and community opinion, respectively) and accordingly employ different mechanisms. Due to this, in the past years, researchers have targeted scenarios focusing on requirements which they could address with only one of these approaches. However, real life scenarios are not split in a way that they can just fit one of these approaches and in many cases, a mixed approach is required. For example, users might be interested in knowing whether a provider has a certificate from a CA but also in experiences other users had in the past while performing transactions with it. In addition, a seller might be interested in protecting an item on sale in different ways depending on the value of the item: based on reputation if the price is of a few euros (e.g., a T-shirt) or based on policies if it is of thousands (e.g., requiring a credit card for a flight).

Therefore, in this paper we propose the integration of policy based and reputation based approaches into a versatile trust management language capable of addressing both the structured organizational environments as well as the unstructured user communities. By combining the two different approaches, our integrated trust mechanism enhances the properties of the existing trust management tools.

2 Policy based vs. Reputation based Trust Management

The term *trust management*, introduced in [5] as "a unified approach to specifying and interpreting security policies, credentials, and relationships which allow direct authorization of security-critical actions", has been given later a broader definition, not limited to authorizations [12]: "Trust management is the activity of collecting, encoding, analyzing and presenting evidence relating to competence, honesty, security or dependability with the purpose of making assessments and decisions regarding trust relationships". Two main approaches are currently available for managing trust: policy-based and reputation-based trust management.

2.1 Policy-based Trust Management

This approach has been proposed in the context of open and distributed services architectures [6, 15, 11, 7] as a solution to the problem of authorization and access control in open systems. The focus here is on trust management mechanisms employing different policy languages and engines for specifying and reasoning on rules for trust establishment. The goal is to determine whether or not an unknown user can be trusted, based on a set of credentials and a set of policies.

In addition, it is possible to formalize trust and risk within rule-based policy languages [18, 13] in terms of logical formulae that may occur in rule bodies.

Currently, policy-based trust is typically involved in access control decisions. Declarative policies are very well suited to specifying access control conditions that are eventually meant to yield a boolean decision (the requested resource is either granted or denied). Systems enforcing policy based trust typically use languages with well-defined semantics and make decisions based on "non-subjective" attributes (e.g., requester's age or address) which might be certified by certification authorities (e.g., via digital credentials). In general, policy-based trust is intended for systems with strong protection requirements, for systems whose behavior is guided by complex rules and/or must be easily changeable, as well as for systems where the nature of the information used in the authorization process is exact.

2.2 Reputation-based Trust Management

This approach has emerged in the context of electronic commerce systems, e.g. eBay. In distributed settings, reputation-based approaches have been proposed for managing trust in public key certificates, in P2P systems, mobile ad-hoc networks, and, very recently, in the Semantic Web. The focus here is on trust computation models capable to estimate the degree of trust that can be invested in a certain party based on the history of its past behavior.

The main issues characterizing the reputation systems are the trust metric (how to model and compute the trust) and the management of reputation data (how to securely and efficiently retrieve the data required by the trust computation) [3].

Marsh [17] made one of the early attempts at formalizing trust using simple trust metrics based on linear equations. This model has been further extended by Abdul-Rahman and Hailes to address reputation-based trust in virtual communities [1]. A number of reputation mechanisms for P2P systems, such as [3, 14, 10], followed similar trust and reputation models.

Typically, reputation-based trust is used in distributed networks where a system only has a limited view of the information in the whole network. New trust relationships are inferred based on the available information (following the idea of exploiting world's information). In these scenarios, the available information is based on the recommendations and the experiences of other users, and it is typically not signed by certification authorities but (possibly) self-signed by the source of the statement. This approach supports trust estimates with a wide, continuum range and allows the propagation of trust (e.g., transitive propagation) along the network as well as weighting of values (e.g., fresher information vs. older information).

3 Integrated View of Trust Management

As described in previuos sections, policy-based and reputation-based trust management address the same problem but from different perspectives. However, these points of view are not always just black or white and in many cases it would be desirable to combine them. In this section we propose an approach in which both of them can be integrated, based on the policy language PROTUNE [7].

First, reputation-based trust can be formalized by relations between *trustors*, *trustees*, *actions*, and *trust levels* [18]. For instance, a fact like

$$\texttt{trust}(P, S, \texttt{diagnosis}(\texttt{viral}), 80{-}100)$$

would model the fact that patient P trusts specialist S on diagnosis of viral diseases with an estimated confidence level belonging to the interval $80 - 100$.

Such trust statements can be the basis for trust propagation (e.g. via rules such as "trust X as a bike mechanic if X is trusted as a car mechanic"), and for access control decisions such as

$$\texttt{allow}(\texttt{download}(\texttt{contents/pre\_release})) \leftarrow$$
$$\texttt{user}(X),$$
$$\texttt{trust}(\texttt{self}, X, \texttt{download}(\texttt{contents/pre\_release}), 90{-}100)\,.$$

Such decisions may consider a notion of *risk*, as in

$$\texttt{trust}(\texttt{ProgramX}, \texttt{Server}, \texttt{storeData}(\texttt{Server}), 80{-}100) \leftarrow$$
$$\texttt{Server.owner:CoXYZ},$$
$$\texttt{risk}(\texttt{fail}(\texttt{Server}), 0{-}0.1)\,.$$

These examples (taken from [18]) show how trust and recomendations can be modelled and applied through a small set of predicates. The problem is: How should the basic *facts* about trust and risk be gathered and maintained?

In some cases, such facts can be defined by standard policy rules, for example:

$$\texttt{trust}(A, B, \texttt{download}(\texttt{file}), 80{-}100) \leftarrow$$
$$\texttt{credential}(X, \texttt{VISA}),$$
$$X.\texttt{type} : \texttt{credit\_card},\ X.\texttt{owner} : B\,.$$

However, the main current approaches are based on numerical models (see [8] for an extensive illustration of the main approaches) and ad-hoc algorithms for gathering, processing, and propagating historical data about past interactions and the resulting trust measures. In perspective, it may be possible to apply probabilistic, possibilistic or annotated logics to handle such numbers, but so far there is no clear indication that this is the right direction, nor any hint on how to do it.

In many approaches, the trust relationships we used as facts (not the inferred ones) are computed automatically based on experience and on the declarations of other users, using a numerical model. On the contrary, in policy based trust, all trust relationship are declared manually (e.g. an entity trusts another entity explicitly creating a statement in FOAF).

We argue that policy based decisions can be enhanced by numerical-based ones and viceversa. For example, in a policy where we protect our credit card, we could think of a policy like the following:

$$\texttt{allow}(\texttt{visaCard}) \leftarrow$$
$$\texttt{credential}(\texttt{member}(Requester), \texttt{bbb}),$$
$$\texttt{trust}(\texttt{self}, Requester, \texttt{buying}, X),\ X > 0.8.$$

specifying that we will give our credit card only to entities that are certified by the Better Business Bureau (company that certifies that a company behaves according to the policies it published) and only if the server has a good reputation (this value is extracted from our personal experiences or inferred by using a reputation based algorithm on the community).

Further difficulties are: (i) data are application dependent, as well as the procedures for obtaining them; (ii) trust is a dynamic concept, i.e., it changes over time.

The above difficulties suggest a modular approach, namely, the computation and distribution of the basic facts on reputation and risk are delegated to suitable external packages. The results of their processing can be imported via HERMES-like [19] predicates such as

$$\texttt{in}(\texttt{trust}(X, Y, A, L),\ \texttt{reputation\_pckg} : \texttt{eval\_trust}()))$$

(more details available in [7]). In the above examples the functions `eval_trust()` wrap queries to the underlying reputation management algorithms, whatever they are. The wrapper collects and return the results of those subsystems as a set of terms matching the first argument of the `in` predicate. Then non-rule-based reputation and risk models can be integrated in policies without any ad-hoc language primitives. Moreover, the semantics of the `in` predicate depends on a time dependent state [7, 19], and this makes it possible to address the dynamic aspects of reputation.

Another advantage of this approach is that a single policy may simultaneously apply different approaches to reputation simply by invoking different packages and combining their results with suitable rules. This kind of flexibility is particularly important in a stage where it is not yet clear which of the competing models of reputation-based trust will become widely accepted, and which application domains they will prove to be good for. It is also possible to change the number and type of parameters of the `trust` and `risk` predicates, if needed by a particular reputation model.

This flexible architecture is compatible both with on-demand trust computation and with proactive propagation of trust evaluation, as reputation packages may receive asynchronous messages from other peers, concerning warnings and reputation evaluations.

3.1 An Application Scenario: Electronic Business

Transaction policies must handle expenses of all magnitudes, from micropayments (e.g. a few cents for a song downloaded to your iPod) to credit card payments of a thousand euros (e.g. for a plane ticket) or even more. The cost of the traded goods or services typically contributes to determining the risk associated to the transaction and hence the trust needed for performing it. For instance, for micro-payments of a few euros or cents, a seller could just check the reputation of the buyer within the community. If the buyer's reputation is high, the risk that he or she would not pay is very low, and thus the transaction can be conducted with a simple check. On the contrary, if a buyer's reputation is low or the amount of money involved in the transaction is high, risk is higher and thus the seller may require stronger guarantees, such as a verified credit card number to ensure that the buyer can and will pay.

The buyer's point of view is dual. If the amount of the transaction is high, the buyer may require strong and objective guarantees that the seller will deliver the goods and that the credit card will not be misused. For example, the buyer may require a secure connection, BBB (Better Business Bureau) certificates, blacklist checks, etc. In addition, the buyer may consider the seller's reputation in the community to increase the chances of successful transaction completion and privacy protection.

4 Conclusions

In this paper we have identified the advantages and limitations of policy-based and reputation-based trust management and described how the two approaches can improve each other. The need for an integrated approach has been motivated with real world scenarios. We proposed an integrated trust management approach that combines rule-based and credential-based trust with numerical trust estimates based on a large number of sources (e.g., user community). Our formalization privileges flexibility and extendibility as a design goal. The extension of the traditional crisp, boolean policies with a continuum range

of trust levels, and the extension of numerical trust models with well defined trust combination and propagation rules, yield a versatile trust management framework capable of addressing the complexity and the variety of semantic web scenarios, involving both structured organizational environments and unstructured user communities.

References

1. A. Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. In *Proceedings of 33rd Hawaii International Conference on System Sciences*, 2000.
2. K. Aberer. P-grid: A self-organizing access structure for p2p information systems. In *Proceedings of Ninth International Conference on Cooperative Information Systems*, 2001.
3. K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *Proc. of 10th International Conference on Information and Knowledge Management*, 2001.
4. T. Beth, M. Borcherding, and B. Klein. Valuation of trust in open networks. In *Proc. of the 3rd European Symposium on Research in Computer Security*. Springer-Verlag, 1994.
5. M. Blaze, J. Feigenbaum, and J.Lacy. Decentralized trust management. In *Proceedings of IEEE Conference on Security and Privacy*, 1996.
6. P. Bonatti and P. Samarati. Regulating service access and information release on the web. In *Proc. of the 7th ACM conference on computer and communications security*, 2000.
7. P. A. Bonatti and D. Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *IEEE 6th International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 14–23, Stockholm, Sweden, jun 2005. IEEE Computer Society.
8. P. A. Bonatti, N. Shahmehri, C. Duma, D. Olmedilla, W. Nejdl, M. Baldoni, C. Baroglio, A. Martelli, V. Patti, P. Coraggio, G. Antoniou, J. Peer, and N. E. Fuchs. Rule-based policy specification: State of the art and future work. Report I2:D1, EU NoE REWERSE, sep 2004.
9. G. Caronni. Walking the web of trust. In *Proceedings of 9th IEEE International Workshops on Enabling Technologies (WETICE)*, pages 153–158, June 2000.
10. C. Duma, N. Shahmehri, and G. Caronni. Dynamic trust metrics for peer-to-peer systems. In *Proc. of 2nd IEEE Workshop on P2P Data Management, Security and Trust*, August 2005.
11. R. Gavriloaie, W. Nejdl, D. Olmedilla, K. E. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *1st European Semantic Web Symposium (ESWS 2004)*, pages 342–356, Heraklion, Crete, Greece, may 2004. Springer.
12. T. Grandison. *Trust Management for Internet Applications*. PhD thesis, Imperial College London, 2003.
13. T. Grandison and M. Sloman. Specifying and analysing trust for internet applications. In *Towards The Knowledge Society: eCommerce, eBusiness, and eGovernment, The Second IFIP Conference on E-Commerce, E-Business, E-Government (I3E 2002)*, IFIP Conference Proceedings, pages 145–157, Lisbon, Portugal, oct 2002. Kluwer.
14. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. Eigenrep: Reputation management in p2p networks. In *Proc. of 12th International WWW Conference*, pages 640–651, 2003.
15. N. Li and J. Mitchell. RT: A Role-based Trust-management Framework. In *DARPA Information Survivability Conference and Exposition (DISCEX)*, Washington, D.C., Apr. 2003.
16. N. Li and J. C. Mitchell. Datalog with Constraints: A Foundation for Trust-management Languages. In *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages (PADL 2003)*, pages 58–73, January 2003.
17. S. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, Uni. of Stirling, 1994.
18. S. Staab, B. K. Bhargava, L. Lilien, A. Rosenthal, M. Winslett, M. Sloman, T. S. Dillon, E. Chang, F. K. Hussain, W. Nejdl, D. Olmedilla, and V. Kashyap. The pudding of trust. *IEEE Intelligent Systems*, 19(5):74–88, 2004.
19. V. Subrahmanian, S. Adali, A. Brink, J. Lu, A. Rajput, T. Rogers, R. Ross, and C. Ward. *HERMES: Heterogeneous reasoning and mediator system*. 1995.

Semantic Policy-based Security Framework for Business Processes

Dong Huang[1,2]

[1]Institute for Algorithms and Cognitive Systems (IAKS)
University of Karlsruhe (TH)
D-76131 Karlsruhe, Germany
dong.huang@ira.uka.de
[2]Siemens AG, Corporate Technology
D-81730 Munich, Germany

Abstract. Web service composition and workflow language enable the definition and execution of business process in various application domains. Security is now a major concern for us to implement business process in the context of web service. Meanwhile policy-based approach is becoming popular for the dynamic specification and regulation of web service constraints. We are going to propose a security framework for business process and use policy language enriched with semantics to represent the security concerns and requirements. Furthermore, the challenges will be listed to guide future research.

1 Introduction

Business Processes describe the interaction and collaboration between multiple parties working towards a common objective or a special function. Each party in the Business Process provides its service interface to be accessed and defines by itself how this interface can be invoked. The introduction of Web Services has provided a new way to conduct the business. For example, in the populate travel agent scenario, a travel agency offers its service for booking a travel package by combining several elementary web services such as flight and hotel reservation.

Web Services composition is currently defined by two largely complementary initiatives for developing business processes. The terms *orchestration* and *choreography* have been widely used to describe business interaction protocol. Orchestration describes the business logic and how web service can interact with each other from the perspective of a single endpoint. Business Process Execution Language for Web Service (BPEL or WS-BPEL 2.0)[1] is an orchestration language that is widely used in industry to define the business process and the execution order. Choreography is associated with globally visibly message exchange and is more collaborative in nature than orchestration. Web Service Choreography Description Language (WS-CDL)[10] is a draft document of W3C and introduces a description language which fixes the rule of the interaction between the parts involved in the system. In this paper we focus on the orchestration approach and use BPEL to describe the related business processes.

Security is one of the major concerns when developing business processes. We distinguish two levels of security requirement: *Task Level* and *Process Level*.

- **Task Level Security**. Business Task describes what is to be done in the business model. In the context of web services, a business task is represented by a web service that fulfils the specification of task. Security requirements in this level include basic aspects such as Authentication, Authorization (Access Control), Non-reputation, Data Integrity and Confidentiality. Web service can protect SOAP messages sent over insecure transports by embedding security headers. The WS-Security standard[11] defines how such headers may include signatures, cipher texts and security tokens. There are several emerging specification of web service security such as WS-Policy, WS-Trust, WS-Privacy, and WS-Federation, covering various facets of security in the context of web service. They are built on the top of WS-Security and define enhancements to provide security protection to web service endpoints and the data communication between them.
- **Process Level Security**. Business process defines how business tasks interact and collaborate. Security requirements in this level are normally defined by Business Rules. A business rule is a statement that defines or constrains some aspects of the business. Business rules are usually expressed as *constraints* or in the form **if** *condition* **then** *action*. Business rules provide a means to express and specify high-level security constraints in the form of policy, which are separated logically and physically from the other components through out business processes. Security concerns arisen from business rules concentrate on the critical constraints in the business model and other aspects, such as those for Six Sigma and Sarbanes-Oxley legislation compliance.

WS-Security and other emerging specifications provide the basic security functionalities, but they do not offer enough support for process level security in web service composition. The initial way to solve the process level security is to integrate business rules into BPEL process manually. Business rules are integrated with process by adding activities, which are used to model the consumption and production of messages, tasks, data or goods. But it is not easy for the developer or administrator to handle the complex rules and deal with the impact of dynamically changing of business rules.

In this paper, we propose a semantic policy-based approach to secure the web service composition for business processes. Section 2 includes related approaches and Section 3 gives an overview of our proposed security framework. Section 5 describes the challenges for the work.

2 Related Work

In the project SECTINO[3][1], a system architecture for local and global workflow system is proposed based on the XACML and SAML. Security concerns are

[1] http://qe-informatik.uibk.ac.at

defined in OCL(Object Constraint Language) with model-driven UML tools. SECTINO employs a static specification and enforement of security policies in web services composition. XACML is good for specifying policy in a specified domain. But it is not semantic rich enough for cross-organitaional orchestration and high-level security requirments.

AO4BPEL[4][2] proposes an aspect-oriented extension to BPEL. It uses aspects-oriented concept to modularize cross-cutting concerns like security and performance in business processes. Although the AO4BPEL framework offers the modularity and dynamic adaptability to the web service composition, it lacks semantic description of security aspects, business processes and business rules. This make conflicts detection and policy negotiation infeasible for securing the web service composition. The adoption of a semantic web language can overcome this limitation with the help of a common ontology basis.

There are a lot of research works and industry standards on using semantic and non-semantic policy for security. Ponder[5], XACML[15] and WS-Policy[3] are typically non-semantic policy framework. KAoS[2], Rei[9] and SWRL[8] are approaches that enriched with semantics using RDF[13] and OWL[14] as standards for policy specification. A comparative analysis between semantic and non-semantic language is made by [7] to show the advantages of semantic policy approach. After comparing these semantic policy languages [7], Rei and SWRL seems to have sufficient capability to represent the security requirements in the context of business process.

All semantic description should base on the same knowledge base. Security restrictions have to be expressed in underlying knowledge representation formalism for an ontological description of policies. A generic policy description framework based on three ontology layer is defined in [12]. The three ontology layers are: a domain-independent upper-level ontology, a Core Legal Ontology and a Core Policy Ontology. The first two components are off-the-shelf ontologies that are used as modeling basis for the construction of domain specific ontologies. In [6], security ontologies are defined in DAML+OWL[4] that allow the annotation of web service with respect to various security related notions such as access control, data integrity and others.

3 Design of the Framework

A service platform to deploy web service composition whose interaction and security are specified and governed by policy need to address the following challenges:

- Policy languages used in the system should be well-defined, flexible enough to allow new policy information to be expressed and extensible enough to add new policy type. Different policy languages from different domains should also be able to interoperate [7].

[2] http://www.st.informatik.tu-darmstadt.de/static/pages/projects/AO4BPEL
[3] http://www-128.ibm.com/developerworks/library/specification/ws-polfram/
[4] http://www.daml.org

– Effective policy combined and created from policies should be able to negotiate during runtime. Changes in a policy should be reflected in the runtime logic[16] and conflicts arisen should be to be detected and resolved on the fly.

For the task level security we tend to use the WS-Policy framework, because WS-Policy has already been well developed and addressed all the necessary security aspects on the task level. For the process level security Rei and SWRL are suitable, because the business rules usually represented as constraints or *if-then* form can be efficiently expressed by logical functions of these semantic policy languages. Business rules are usually defined by different parties and distributed through out the network, so a policy langugage with rich semantics can also help the interoperation and combination of these business rules. Even though different semantic/non-semantic policies can be used to represent security concerns at both task and process level, policies with semantics built on common security ontology are more general and flexible.

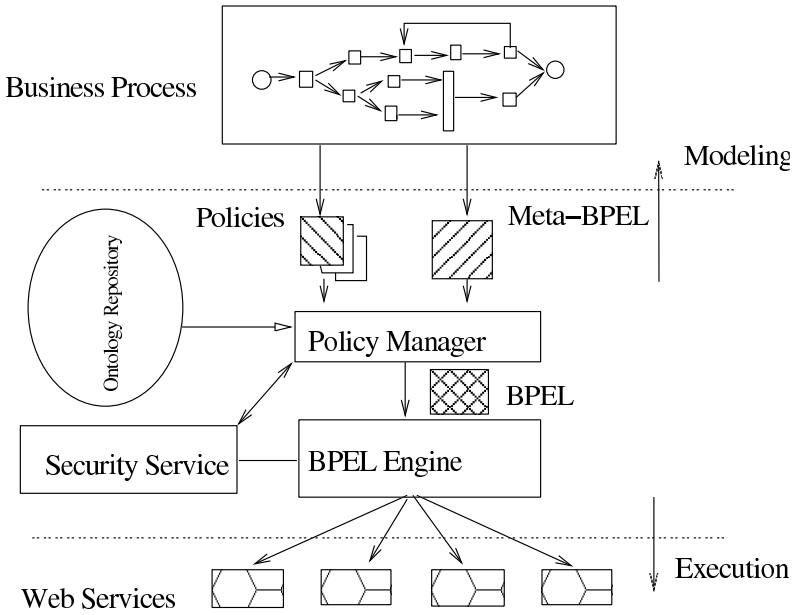


Fig. 1. Architecture of the security framework

Fig. 1 shows the architecture of the semantic policy-based framework for business process. The output of the business process modelling and model-driven security requirements analysis on the top layer are a set of security policies and the meta business process, which describes abstract process with functional tasks. All security concerns arisen from different domains and reasons, such as legal problems, privacy and changes of business rules, will be covered in the

security policies. These policies would be specified and annotated with semantics based on the **Ontology Repository**. Two kinds of ontology in the repository are:

- Business Ontology describes the concepts and relations related to the current business process.
- Security Ontology illustrates the relations among security concepts like authorization and authentication.

The **Policy Manager** gets the meta process definition and policies as input. Formally described policies can be checked for compatibility via matching. Description Logic will be used to conduct the matching phase and make the policy negotiation and conflicts detection possible. Then the Policy Manager creates as output the BPEL process definition, in which semantic policies that represent the business rules and other security requirements are integrated. New tasks or activities, which access the **Security Service** to get the necessary security token for SOAP message or invoke encryption and signature methods, are inserted into the meta BPEL to create the new BPEL file. Policy changes can be deployed and take effect on the fly without stopping the process by using aspect-oriented extension to BPEL like AO4BPEL.

4 Conclusion and Future Work

In this paper we addressed our ongoing research about a semantic policy-based security framework for business processes. We have distinguished all security concerns and requirements into two levels: Task and Process Level. The architecture of security framework is designed to support runtime policy management and enforcement. Security policies are built on the top of ontology to enrich representation of security concerns and enable reasoning for conflict detection and policy negotiations. The challenges and issues, which deserve future research, will be the following:

- Model-Driven Security Modelling. There are ongoing standardization effort for business process modelling from both OMG[5] and BPMI[6]. Security as an important concern is still not well specified to incorporate with the business process modelling standards from OMG and BPMI.
- How to define and translate security concerns to semantic policies? Ontology and rule language, such as SWRL, should be used to represent declarative policy in the future work.
- How to enforce policies on the business process during the runtime? The aspect-oriented approaches can modularize the crosscutting concerns like security and should be implemented on the process level to enforce the policies dynamically.

[5] http://www.omg.org
[6] http://www.bpmi.org

5 Acknowledgment

The authors would like to thank the team at Siemens CT IC 3 for their guidance and support whilst conducting this research. In particular, we thank Jorge Cuellar for his valuable contributions towards this work.

References

1. Arkin, A., Askary, S., Bloch, B., and Curbera, F. Web services business process execution language version 2.0. Tech. rep., OASIS, December 2004.
2. Bradshaw, J., and Uszok, A. Representation and reasoning for daml-based policy and domain services in kaos and nomads. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems* (New York, NY, USA, 2003), ACM Press, pp. 835–842.
3. Breu, R., and Hafner, M. Sectino:inter-organizational workflow security in e-government, 2004.
4. Charfi, A., and Mezini, M. Aspect-oriented web service composition with ao4bpel. In *ECOWS* (2004), vol. 3250 of *LNCS*, Springer, pp. 168–182.
5. Damianou, N., Dulay, N., Lupu, E., and Sloman, M. Ponder:a language for specifying security and management policies for distributed systems. Tech. rep., Imperial College, October 2000.
6. Denker, G., Kagal, L., Finin, T., Sycara, K., and Paoucci, M. Security for daml web services: Annotation and matchmaking. In *Second International Semantic Web Conference* (September 2003).
7. Felix Clemente, G. P. Representing security policies in web information systems. In *Proceedings of WWW 2005* (May 2005).
8. Horrocks, I., and Patel-Schneider, P. F. Swrl: A semantic web rule language combining owl and ruleml. Tech. rep., The Rule Markup Initiative, May 2004.
9. Kagal, L., Finin, T., and Joshi, A. A policy language for a pervasive computing environment. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks* (2003).
10. Kavantzas, N., Burdett, D., and Ritzinger, G. Web services choreography description language version 1.0. Tech. rep., December 2004.
11. Kelvin Lawrence, Chris Kaler, S. A. Web services security. Tech. rep., OASIS, 2004.
12. Lamparter, S., Eberhart, A., and Oberle, D. Approximating service utility from policies and value function patterns. In *Proc. of the 6th IEEE Workshop on Policies for Distributed Systems and Networks* (JUN 2005), IEEE Computer Society.
13. Manola, F., and Miller, E. Rdf primer. Tech. rep., W3C Recommendation, February 2004.
14. McGuinness, D. L., and van Harmelen, F. Owl web ontology language overview. Tech. rep., W3C Recommendation, February 2004.
15. Moses, T. extensible access control markup language (xacml) version 2.0 3. OASIS Standard, Feb 2005.
16. Mukhi, N. K., and Plebani, P. Supporting policy-driven behaviors in web services: experiences and issues. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing* (New York, NY, USA, 2004), ACM Press, pp. 322–328.

RBAC Policy Engineering with Patterns

Taufiq Rochaeli* and Claudia Eckert

Departement of Computer Science
Darmstadt University of Technology
{rochaeli,eckert}@sec.informatik.tu-darmstadt.de

Abstract. We present a RBAC policy engineering approach that supports administrators to specify RBAC policies with the help of experts' knowledge, which is documented using the pattern paradigm. These patterns are formalised in Web Ontology Language (OWL) that enables machine interpretation of experts' knowledge and reasoning about the RBAC policy. Thus, administrators could specify RBAC policies by choosing the patterns matching their scenario and asserting instances without knowing the complex RBAC policy specification.

1 Introduction

Role-Based Access Control (RBAC) [1] offers a better manageability than traditional Mandatory Access Control (MAC) or Discretionary Access Control (DAC) to fulfil organisational security policies. However, in an organisation with complex scenarios, the specification of access control policies using role based model may also be overwhelming for security administrators. They need domain as well as security experts' knowledge to accomplish this task. This knowledge provides security administrators with well-proven empirical solutions to access control policies specification within a certain domain (i.e. hospital, government, etc.), which is strongly affected by social and/or economical factors. Furthermore, increasing complexity of the scenario also poses another inconsistency problem, which is caused by the manual specification of RBAC policies. For this reason, a tool support to generate RBAC policies specification is required.

We developed a role engineering approach, which follows the scenario-oriented requirements engineering approach [2] that assists security administrators to specify the RBAC policies with the help of experts' knowledge. As argued above, the experts' knowledge on specifying RBAC policies with consideration on several factors in a certain domain will greatly helps security administrators to specify RBAC policies.

To document the experts' knowledge, we follow the structure proposed in pattern paradigm originated from the architecture field [3]. This paradigm has been widely used in several fields in computer science [4, 5]. Despite of the improvement in their engineering processes, pattern paradigm still has a major

* partially supported by the German Ministry of Education and Research (BMBF) under SicAri Project.

drawback: novice users find it difficult to interpret and to apply the patterns. By encoding experts' knowledge in specifying RBAC policies in semantic language, i.e. OWL, we explore the possibilities of automatic generation of RBAC policies by using reasoning services provided by description logic-based knowledge representation system. With this framework, security administrators only need to interpret their scenario, search for patterns that match the scenario and assert instances in the scenario into the knowledge representation system. Hence, the complexity of RBAC policies specification could be concealed by the abstraction of more understandable concepts of the scenario.

This paper is organised as follows. Section 2 begins with the background of this work and subsequently presents the definition of RBAC policy patterns in description logic notation[6]. Section 3 outlines the generation process of RBAC policies by using the reasoning services of knowledge representation system. Section 4 discusses some related works in pattern paradigm and role engineering. Finally, section 5 presents the conclusion.

2 Definition of RBAC Policy Pattern

This section briefly describes the background of pattern paradigm and continues with the motivation that drives the adaptation of pattern paradigm in RBAC policies specification process. It explains the similarities between construction design and RBAC policies specification and continues with the definition of RBAC policies pattern.

To design a building, architects should consider many non-technical human-related factors, for example, social and/or psychology factors. This is necessary, due to the fact that people live and interact in the building and have their own needs. Therefore, architects should design constructions, which fulfil the needs emerging from these factors.

Christopher Alexander proposed *pattern* method in his work [3], which structurally captures experts' knowledge. Each pattern has three general parts, a **context**, a **problem** and a **solution**. The context describes the environment, in which the pattern shall apply. It covers both temporal and spatial aspects of the environment that represent a scenario. In this context, there exist forces[1] that need to be resolved. These forces characterise the problem in the context. Finally, the solution proposes a configuration or a design which should resolves the existing forces in the context. Collected patterns do not exist independently; they have one or more relationships with other patterns. These relationships reflect the conflicts, compatibilities and dependencies between patterns applied within a context.

The pattern approach in software engineering [4] and security engineering [5], which captures the solutions considering different non-technical factors, motivates this work to adapt the pattern approach in RBAC policy specification. Thus, RBAC policy pattern shall capture this expertise knowledge, which has

[1] In our work, we interpret forces as requirements and problems.

the solution to specify access control policies in a certain situation by considering various non-technical factors, even the subtle one.

A pattern consists of a context with problem and a solution: $\mathsf{Pattern} \equiv \exists \mathit{hasCtx}.\mathsf{Context} \sqcap \exists \mathit{hasSln}.\mathsf{Solution}$. In the next two subsections we define the main parts of RBAC policy pattern, which are described using description logic notation.

2.1 Context and Problem

The context of RBAC policy pattern represents a novel description of business process scenario. It has SubjectInTaskClass and ObjectClass as main concepts, which are interpreted as a class of subject performing a certain task, and as a class of resources, respectively. Optionally, the context may also have concepts and concepts relationships of the Event-controlled Process Chain [7] such as Event and *controlFlow*, which links Event with SubjectInTaskClass.

The critical aspect of context of business process is the information that flows from subjects to objects and vice versa. Information flow is needed in order to perform the business process. On the other hand, a possible unintended information flow could also pose a security threat to the business process. We identify two main classes of problem, which arise in this context. They are *workflow requirements* and *security threats*. These problems are denoted by concept relationships between SubjectInTaskClass and ObjectClass. A workflow requirement relationship means that a subject needs an access to objects in order to perform the business process. The workflow relationship is further classified into *hasWflowReqInput* and *hasWflowReqOutput*, which represent the input and output information flow needed in business process between the subjects in a task and the objects. A security threat relationship means that a subject could perform an attack to objects, which can cause any harm to the business process. The security attack relationship is also classified into two classes, *hasSecThreatInput* and *hasSecThreatOutput*, which represent the input and output information flow between the subjects in a task and the objects posing security threats.

Fig. 1 shows an excerpt of transaction pattern context adapted from the reference model of an industrial business process in [8] with its problem. In this context, a malicious worker in WarehouseManagement task could issue a fictive transaction, so that he can steal some goods in the warehouse. This threat is represented by *createFictiveTransactionIn*. Note that, we do not define problem as separate concept, because the problem is already represented by concept relationships.

It is also possible to define a context having conflicting forces, i.e., a task class requires a write access to a database, but it also poses a security threat when it write to the database. To prevent this definition, both axioms

$\neg(\exists \mathit{hasTask}.\exists \mathit{hasSecThreatInput}.\mathsf{ObjectClass}) \sqsupseteq \exists \mathit{hasTask}.\exists \mathit{hasWflowReqInput}.\mathsf{ObjectClass}$

and

$\neg(\exists \mathit{hasTask}.\exists \mathit{hasSecThreatOutput}.\mathsf{ObjectClass}) \sqsupseteq \exists \mathit{hasTask}.\exists \mathit{hasWflowReqOutput}.\mathsf{ObjectClass}$

should be defined.

$$
\begin{aligned}
\mathsf{TransactionCtx} &\equiv \exists \mathit{hasTask}.\mathsf{TransactionMonitoring} \sqcap \exists \mathit{hasTask}.\mathsf{WarehouseManagement} \sqcap \ \ldots \\
&\quad\ \exists \mathit{hasObject}.\mathsf{Transaction} \sqcap \exists \mathit{hasObject}.\mathsf{DunningLetter} \sqcap \\
&\quad\ \exists \mathit{hasObject}.\mathsf{DeliveryNote} \sqcap \exists \mathit{hasObject}.\mathsf{Inventory} \sqcap \ \ldots \\
\mathit{createFictiveTransactionIn} &\sqsubseteq \mathit{hasSecThreatOutput} \\
\mathsf{WarehouseManagement} &\equiv \exists \mathit{createFictiveTransactionIn}.\mathsf{Transaction} \sqcap \exists \mathit{hasWflowReqInput}.\mathsf{Transaction} \sqcap \ldots \\
\mathsf{TransactionMonitoring} &\equiv \exists \mathit{hasWflowReqInput}.\mathsf{Transaction} \sqcap \exists \mathit{hasWflowReqInput}.\mathsf{DunningLetter} \sqcap \ldots
\end{aligned}
$$

Fig. 1. Context and problem of *Transaction* pattern

2.2 Solution

The solution part of a RBAC policy pattern specifies authorisation policies of a context that fulfil the requirements within this context. Currently, the solution only proposes the core RBAC and hierarchical RBAC specification of the RBAC INCITS standard [1].

A solution defines permission and role concepts, which have different interpretation than the concepts introduced in RBAC model. We interpret *permission* as a class of subjects which have a certain permission. For example, $\mathsf{P1} \equiv \exists \mathit{read}.\mathsf{DatabaseX}$ is a class of subjects which are permitted to read database X. A *role* is interpreted as a class of subjects which have required permissions to perform a task. A role concept is constructed by intersection of different permissions. With our definition, $\mathsf{R2} \sqsubseteq \mathsf{R1}$ is interpreted by DL reasoning service as: (1) all permissions of R2 are included in R1, (2) all users in R2 are also members of R1. Thus, role hierarchy relationship $\mathsf{R2} \preceq \mathsf{R1}$ could be represented by inclusion $\mathsf{R2} \sqsubseteq \mathsf{R1}$. In case of redundant definition of permissions among roles, role hierarchy could be automatically detected and built by using classification service provided by reasoning engine.

From the context, the administrator already knows the membership of subjects in task classes. We extend the interpretation of SubjectInTaskClass to a class of subjects performing a certain task and having the necessary role(s). Therefore, the solution defines SubjectInTaskClass as intersection of roles. This kind of definition implicitely defines $\mathsf{SubjectInTaskClass} \sqsubseteq \mathsf{Role}$, which ensures that every subjects having task membership are always assigned to roles.

Fig. 2 shows an excerpt of transaction pattern solution.

$$
\begin{aligned}
\mathsf{TransactionSln} &\equiv \exists \mathit{defines}.\mathsf{ManufacturingRole} \sqcap \exists \mathit{defines}.\mathsf{ShippingDeptRole} \sqcap \ldots \\
\mathsf{ManufacturingRole} &\equiv \exists \mathit{get}.\mathsf{DunningLetter} \sqcap \exists \mathit{put}.\mathsf{DunningLetter} \sqcap \exists \mathit{get}.\mathsf{Transaction} \\
\mathsf{ShippingDeptRole} &\equiv \exists \mathit{get}.\mathsf{Transaction} \sqcap \exists \mathit{put}.\mathsf{DeliveryNote} \sqcap \exists \mathit{put}.\mathsf{Inventory} \\
\mathsf{TransactionMonitoring} &\equiv \mathsf{ManufacturingRole} \\
\mathsf{WarehouseManagement} &\equiv \mathsf{ShippingDeptRole}
\end{aligned}
$$

Fig. 2. Solution of *Transaction* pattern

2.3 Relationship between RBAC Policy Patterns

The relationships between patterns distinguish pattern from template. They represent compatibility between patterns and guide the pattern application process. The compatibility between patterns is represented by *refinement* and *dependency* relationships. The *conflict* relationship should guide the pattern user to avoid simultaneous use of these patterns, which have conflicting requirements between pattern contexts. In this paper, we only focus on *conflict* relationship. A pattern can conflict with another pattern, if and only if, the contexts of these patterns have any task class, which has a security threat relationship in one context, and also has a workflow requirement relationship in another context to the same object class. This relationship is formally defined in fig. 3.

$$
\begin{aligned}
confl \equiv\ & hasCtx \circ \\
& ((hasTask \circ hasSecThreatInput \circ hasObject^{-} \sqcap hasObject \circ hasWorkflowReqInput^{-} \circ hasTask^{-}) \sqcup \\
& (hasTask \circ hasSecThreatOutput \circ hasObject^{-} \sqcap hasObject \circ hasWorkflowReqOutput^{-} \circ hasTask^{-})) \circ \\
& hasCtx^{-}
\end{aligned}
$$

Fig. 3. Conflict relationship

3 Generating RBAC policies

In order to generate RBAC policies, security administrator should first interpret his scenario. Next, he chooses patterns starting from the most general one to the detailed one. The selection criterion is that the context of patterns should (partially) match the concrete scenario and (partially) fulfil the requirements of scenario. For each selected pattern, concept instances of pattern, concept and solution and concept relationships should be asserted.

After all requirements of scenario have been met by the patterns, he asserts the instances of scenario matching the context concepts into the knowledge representation system. Subject and role assignments are be defined by retrieving instances of roles. Role and permission assignments are defined by retrieving concept descendants of role.

4 Related Works

Previous work on formalisation of patterns [9, 5] propose formalisation of patterns based on first-order predicate logic and frame-logic, respectively. The first work only defines the formalisation of pattern *solution* without its related *context* and *problem*. It also lacks of definition of pattern relationship. The latter work only defines the formalisation of pattern *structure* and its relationships. In comparison of previous works to our case, we encode the context, problem and the solution in semantic language based on description logic. Therefore, machine interpretation of context, problem and solution is possible. However, in our current

work, searching and selecting suitable patterns is still done by involving human intelligence, which interprets the context of pattern. Since the description logic notation of context is still hard to understand, each pattern also provides the description of its context as plain text. In the role engineering area, Neumann et. al. present in [10] a method to derive role from scenario. Our approach differs in a way that the expertise knowledge in a certain domain is documented using pattern paradigm instead of catalog.

5 Conclusion

In our approach, the involvement of the policy designer in the RBAC policies specification task is reduced only to scenario identification, patterns selection and instances assertion. Thus, the possibility of human error in RBAC policy specification that could lead into inconsistent and redundant specification could be avoided.

Acknowledgements

The authors thank anonymous reviewers for helpful comments.

References

1. American National Standards Institute: ANSI Standard 359-2004: Role Based Access Control (2004)
2. Alistair G. Sutcliffe, Neil A.M. Maiden, Shailey Minocha, Darrel Manuel: Supporting scenario-based requirements engineering. IEEE Transactions on Software Engineering **24** (1998) 1072 – 1088
3. Alexander, C.: The Timeless Way of Building. Oxford University Press (1979)
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Addison Wesley (1995)
5. Schumacher, M.: Security Engineering with Patterns - Origins, Theoretical Model, and New Applications. Volume 2754 of LNCS. Springer (2003)
6. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook. Cambridge University Press (2004)
7. (G. Keller, M. Nüttgens, A.-W. Scheer)
8. A. -W. Scheer: Wirtschaftsinformatik: Referenzmodelle für industrielle Geschäftsprozesse(in German). Springer (1998)
9. Alencar, P., Cowan, D., Dong, J., Lucena, C.: A pattern-based approach to structural design composition. In: Twenty-Third Annual International Computer Software and Applications Conference, IEEE Press (1999)
10. Neumann, G., Strembeck, M.: A scenario-driven role engineering process for functional rbac roles. In: SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies, New York, NY, USA, ACM Press (2002) 33–42