An Eclipse plug-in for specifying security policies in modern information systems

Simone Mutti, Mario Arrigoni Neri, Stefano Paraboschi *

Dipartimento di Ingegneria dell'Informazione e Metodi Matematici
Universitá degli Studi di Bergamo, 24044 Dalmine BG, Italy
{simone.mutti,mario.arrigonineri,parabosc}@unibg.it

Abstract. In this paper we want to show how the Eclipse platform can offer very interesting support and be an adequate infrastructure in the realization of a software environment for the design and development of security policies. The efficient and effective design of security policies for modern information systems is a crucial open problem, important and at the same time quite difficult. The lifecycle of a company-wide security system is complex. Its correct management requires conceptual depth, with the ability to describe the security configuration at different abstraction levels, from a representation that is near to the way security requirements arise from business applications, to the concrete configuration of the security components that characterizes the implementation of the system. Also, information systems present a large variety of security-enabled tools and devices, each one tailored to a specific task and operating on a specific representation of a system subset, and a correct security management environment has to be able to cover as many of them as possible. Ideally, an integrated security management environment should provide interoperability among all the security tools and assist the user in such a complex job. In such a scenario, Eclipse is able, with its open source technology and its flexible architecture, to support the realization of such systems, facilitating the integration among the separate models required to describe the security configuration of a heterogeneous system. For instance, formal ontologies can be easily integrated and they can be used for the definition of the involved models, with clear benefits in terms of automatization of model definition and control.

1 Introduction

As applications become more sophisticated, extensive and operate in an increasingly open and integrated environment, an urgent need grows of flexible and automated tools for the management of information security aspects. The proper management of security requirements is an important open problem in this environment.

* The work in this paper has been supported by the PoSecCo project (project no. 257129), co-funded by the European Community under the Information and Communication Technologies (ICT) theme of the 7th Framework Programme for R&D (FP7).

Among the many topics investigated by access control research in recent years, two themes are particularly relevant to this scenario. The first is the investigation on new access control models able to meet the policy needs of real world application domains. This work has led to several successful and now well established models, such as the RBAC96 model [1], the NIST Standard RBAC model [2] and the RT model [3]. In a parallel line of investigation, researchers have developed a variety of policy languages for access control, in most cases with a direct integration with the languages and models of the modern Web scenario. These include industry standards such as XACML [4], but also academic efforts ranging from more practical implemented languages such as Ponder [5] to theoretical languages like the one proposed by Jajodia et al. in [6] and finally to Semantic Web based languages such as Rei [7] and KAoS [8]. Policy languages based on Semantic Web technologies allow policies to be described over heterogeneous domain data and promote a common understanding among participants, who might not use the same information model. This paper is motivated by the consideration that these two parallel efforts - access control models and Web-conscious policy languages - need to develop a synergy in order to enable the development of security infrastructures with verifiable security properties for emerging open and dynamic environments.

An abstract policy language without ties to a concrete model gives the designer considerable freedom, but it may also offer limited guidance and insufficient precision in the description. Conversely, an excessively concrete model may not have the flexibility needed to correctly express the policy of a given system or may force the omission of important aspects. For instance the NIST Standard RBAC model only supports the definition of the specific constraints of static and dynamic separation of duties. No attention is paid to the representation of other properties of the security policy. A policy language built on the basis of the NIST Standard RBAC model can be successfully applied to the management of security policies of real systems only if it supports the specification of a large variety of additional details that are not captured directly in the model. Our goal is to create an environment for the design and management of security policies that follows the approach proposed by the European project PoSecCo (`http://www.posecco.eu`). PoSecCo plans to represent the security policies at three distinct levels of abstracion: Business, IT and Landscape. The design of the security policy should start with the definition of the security requirements at the Business level and then proceed to the IT level and finally the Landscape level, maintaining a traceable link between the elements at the different levels. These connections are crucial to support policy inspection and maintenance activities. We have a specific interest in the IT level description of security policies. Our first goal is to contribute to the definition of the IT level policy model, providing a description that is based on the representation of the common features of an Information Technology (IT) infrastructure. To achieve our goal we need models that are able to represent extended system features, to provide a precise description of the expected system behavior. Another goal is to increase knowledge on the security configuration in the model relying on

Semantic Web technology, and specifically on an enrichment of the model with the use of OWL (Ontology Web Language) [9].

The use of OWL within the definition of the model will bring significant benefits, especially in terms of partial automatization of the process of model verification and refinement. In fact, thanks to the use of OWL, model implementation can be done in part automatically. To introduce these automated processes, the model relies on ontological argumentation (reasoning), a powerful tool to derive new information and make an extensive verification of the model correctness.

This paper presents the main design principles that have been used in the construction of an Eclipse plug-in that implements the aspects seen before. The motivation underlying the implementation of the plug-in is that Eclipse is the most used open source development framework, offering a high degree of flexibility and supporting the extention of its functionality through the implementation of plug-ins. Moreover, Eclipse can provide a common repository model as well a common plugin interface to facilitate the integration between this and other plugins, which will be used to manage business level and landscape level descriptions of the security policy. Finally, the Eclipse platform encourages the reuse of the functions of other plug-ins and modules of the Eclipse framework, speeding up and improving the quality of the development process.

The paper is organized as follows. Section 2 presents the main features of the IT-level model used for the description of security policies. Section 3 briefly illustrates the role of Semantic Web technology and OWL. Section 4 discusses the design principles that have been followed in the design of the Eclipse plugin for PoSecCo. Section 5 shows the interface the plugin offers to the security designer. Finally, Section 6 draws some concluding remarks.

2 Access control IT-level metamodel

The metamodel that is being used in PoSecCo to describe security policies at IT-level consists of four related conceptual blocks, describing the main aspects that characterize access control and authentication services.

- The *profiling* metamodel is used to describe the organization by grouping users into groups and allowing them to activate roles.
- The *authorization* metamodel describes a taxonomy of authorizations and some basic properties of authorizations themselves.
- The *privilege* metamodel is used to describe the detailed structure of privileges and works as a fundamental link to the software and hardware infrastructure, involving the actual landscape entities as target.
- The *network* metamodel is used to describe the network layer.

The profiling metamodel is composed of many classes. The *Principal* class represents generic user profiles or groups and is specialized into two subclasses: *Identity* and *Role*. Identities are static hierarchical user classifications. In general, a principal can correspond to any user or to any specific organization unit that

corresponds to a group of identities. *GroupIdentity* is a specialization of the Identity concept and denotes (directly or indirectly) a set of lower level identities. *SingleIdentity* corresponds to an Identity that represents a single user in the system. *Role* is the core concept of the submodel. The concept of role offers a variety of interpretations, adaptable to the specific features of the access control models implemented in the landscape.

The authorization metamodel contains several classes. The *PolicySet* class represents the aggregation of policies. The *Policy* class consists of a set of authorizations. It is associated with a target that specifies restrictions valid over all the authorizations in the policy. The *Target* class permits to specify for each policy a set of restrictions that have to be applied to all the authorizations in the policy. The core concept of this part is the *Authorization* class. There are two different types of authorizations: *Role Authorization* and *System Authorization*. A Role Authorization represents the authorization to activate a role, that is, to enable all the privileges granted to such a role. System authorizations correspond to the classical authorizations/rules of most access control models.

The main classes that appear in the Privilege metamodel are: *Privilege*, *Action* and *Resource*. The *Privilege* class represents the right of performing some action and its subclass *ResourcePrivilege* represents privileges that are associated with some specific resource. Another class is *Action*, which describes a generic business or technical action. The structure of the action is strictly related to the characteristics of the underlying access control system. In fact, the *Action* class is related with the *System* class that represents the system where the action can be performed. The Action is also related with the *SystemAction* class that represents the specific action of a system. Another class in the privilege metamodel is *Resource*, which rapresents the entity on which a privilege can be assigned. There is a large variety of resources to manage in the PoSecCo environment. The main distinction we can introduce in resources is between *Static* and *Dynamic* resources. A *Static* resource is an information resource, like a file or a folder in a file system. The static resource entity has a specialization: *ProtectedDataObject* represents an abstraction of a protected resource. The *ProtectedDataObject* has two specializations: *EncryptedDataObject* and *SignedDataObject*. *EncryptedDataObject* represents an encrypted resource. *SignedDataObject* represents a resource whose content has been signed with a digital signature. A *Dynamic* resource corresponds to a target of an authorization that is responsible for the execution of a specific function. The dynamic resource entity has two specializations, *Service* and *ServiceInterface*. A service represents a complete set of service invocations, responsible for the execution of a specific application. A Service must expose at least one ServiceInterface. A ServiceInterface is an atomic service invocation provided by a service.

The network metamodel is composed by: *NetworkElement*, an abstract class that describes elements in a computer network. NetworkElement has two specializations: *Node* and *NetworkConnection*. A Node is an element of the network that contains a lot of resources. *NetworkConnection* is a data transport entity

that transports data from one Node to another Node at the same layer. Node has a specialization, *Link*. A Link is a connection between nodes.

3 Semantic Web and OWL

The term *Semantic Web* refers to both a vision and a set of technologies. The vision is articulated, in particular by the World Wide Web Consortium (W3C), as an extension to the existing web in which knowledge and data could be published in a form easy for computers to understand and reason with. Doing so would support more sophisticated software systems that share knowledge, information and data on the Web just as people do by publishing text and multimedia. Under the stewardship of the W3C, a set of languages, protocols and technologies have been developed to partially realize this vision, to enable exploration and experimentation and to support the evolution of the concepts and technology. The current set of W3C standards are based on RDF [10], a language that provides a basic capability of specifying graphs with a simple interpretation as a *semantic network* and serializing them in XML and several other popular Web systems (e.g., JSON). Since it is a graph-based representation, RDF data are often reduced to a set of triples where each represents an edge in the graph or, alternatively, a binary predicate. The Web Ontology Language (OWL) [9] is a family of knowledge representation languages based on Description Logic (DL) [11] with a representation in RDF. OWL supports the specification and use of ontologies that consist of terms representing individuals, classes of individuals, properties, and axioms that assert constraints over them.

The use of OWL to define policies has several very important advantages that become critical in distributed environments involving coordination across multiple organizations. First, most policy languages define constraints over classes of targets, objects, actions and other constraints (e.g., location). A substantial part of the development of a policy is often devoted to the precise specification of these classes. This is especially important if the policy is shared between multiple organizations that must adhere to or enforce the policy even though they have their own native schemas or data models for the domain in question. The second advantage is that OWL's grounding in logic facilitates the translation of policies expressed in OWL to other formalisms, either for analysis or for execution.

4 Eclipse and PoSecCo

Eclipse (`http://www.eclipse.org`) can be considered as a technology, a development platform, and a family of tools. From an infrastructural point of view, Eclipse is a collection of both plug-ins and access points for plug-ins. To take advantage of modern development techniques and to enhance PEPP (PoSecCo Eclipse Policy Plug-in) extensibility, we chose the Eclipse framework as our development platform and decided to implement the proposed metamodel as an Eclipse plug-in.

4.1 Integration between Eclipse and PoSecCo

In Figure 1 we can see how the Eclipse architecture allows us to have a vertical integration of the various layers that make up the architecture of PoSecCo. Within each block we can find various layers that manage particular aspects of the security policies. These blocks can be easily integrated within Eclipse, in agreement with the philosophy that considers Eclipse as a collection of both plug-ins and access points of plug-ins.

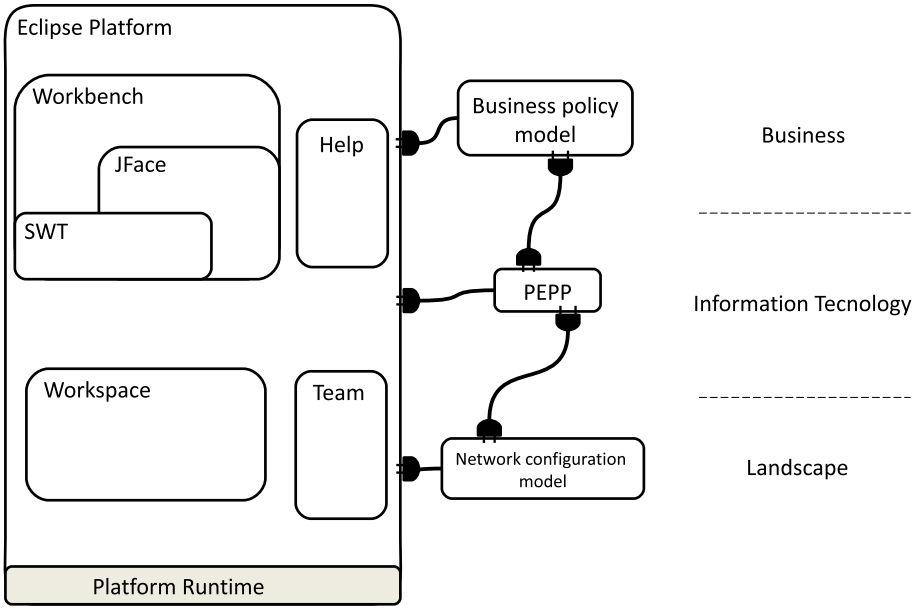

Fig. 1. Integration of the different components

4.2 PoSecCo Eclipse Policy Plug-in (PEPP)

We introduce here the PEPP (PoSecCo Eclipse Policy Plug-in) component, described in Figure 2. This is the core of the system, responsible for managing the IT level representation of the security policy.

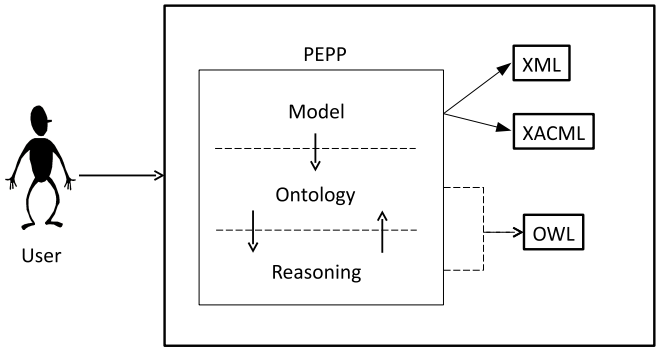

Fig. 2. Abstract architecture of the plug-in.

In particular, this architecture allows us to have two representations of the access control model defined above, a *model* representation understandable by the user (via form-based interface) and a representation understandable by the machine through *formal ontology and reasoning*. This allows us to use the power of reasoning because, especially in model checking, these controls can be performed automatically by the machine.

XML and XACML The plug-in provides support to the user during data entry. User is guided to describe consistent models due to ontology constraints. The obtained model is stored inside Eclipse project as an *XML* file that is valid w.r.t. a *PEPP* specific language. When the data entry is complete the user can export the created model in some task oriented formats, including *OWL* and *XACML*. The plug-in allows this choice according to the Eclipse philosophy. We want to maintain some flexibility as well as also an high level of integration with other modules that use the XML language or other tools that use XACML format. The use of the OWL format is instead used internally to the plug-in for reaoning related tasks.

OWL and OWL API The ontology is implemented within the plug-in through the latest version of the OWL API. OWL API has been designed to meet the needs of people developing OWL based applications, OWL editors and OWL reasoners. It is a high level API that is closely aligned with the OWL 2 specification. It includes first class change support, general purpose reasoner interfaces, validators for the various OWL profiles, and support for parsing and serializing ontologies in a variety of syntaxes. The API also has a very flexible design that allows third parties to provide alternative implementations for all the major components.

Reasoning A key part of working with OWL ontologies is reasoning. Reasoners are used to check the consistency of ontologies, check to see whether the signa-

ture of an ontology contains unsatisfiable classes, compute class and property hierarchies, and check to see if axioms are entailed by an ontology. The OWL API has various interfaces to support the interaction with OWL reasoners.

A reasoner is a key component for working with OWL ontologies. In fact, virtually all querying of an OWL ontology (and its imports closure) should be done using a reasoner. This is because knowledge in an ontology might not be explicit and a reasoner is required to deduce implicit knowledge so that the correct query results are obtained. The OWL API has various interfaces to support interacting with OWL reasoners. In PEPP we use HermiT (`http://www.hermit-reasoner.com/`) as reasoner. HermiT is the first publicly-available OWL reasoner based on a "hypertableau" calculus, which provides more efficient reasoning than most other algorithms. Ontologies that previously required minutes or hours to classify can often by classified in seconds by HermiT. This performance gain allows HermiT to classify a number of ontologies that had previously considered too complex.

For more specific controls, such as check of correctness of the policy that are beyond Description Logics expressivity, some specific inference rules must be introduced. *SWRL* (Semantic Web Rule Language) [12] is a proposal for a Semantic Web rules-language, combining sublanguages of the OWL Web Ontology Language (OWL DL and Lite) with those of the Rule Markup Language. Rules are in the form of an implication between an antecedent (body) and consequent (head). The intended meaning is consistent with the classical semantics that characterizes most rule paradigms: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. Thanks to the introduction of these rules, the verification process of the model can be automated. The use of SWRL offers interesting opportunities in the PoSecCo context and is well supported by the reasoners that can be easily integrated in Eclipse.

5 PEPP

PEPP fully supports all the phases of the process depicted in Figure 2. The PEPP plugin is composed of several modules:

- a main window to display and edit policy description files. The plugin assigns a different tab panel in the window to every top concept from the meta-model. This set of tabs provides the user with a high level guide through the model, always offering direct access to the main components of the model. The selection of a tab open a form that permits to enter values for each of the properties of the corresponding entity. Instances of each class can be described in a homogeneous way by the same form used for entering its properties. The plug-in has many forms (see Figure 4), in particular there is a form for each class of the model described in Section 2. These forms allow to maintain a high usability of the plug-in.
- a navigation tool, that organizes resources in a finer taxonomy according to the specific top level concept selected by the user in the main window.

The classification proposed to the user is driven by the ontology itself. This makes the tool quite flexible and automatically adaptable to changes in the ontology. The tool can be applied to any fragment of the metamodel.

– a set of task buttons, published in the Eclipse main toolbar, that offer direct access to functions like the creation of the OWL ontology starting from an instance of the security policy or the activation of the reasoning-based checking.

As a first task, the plug-in requires to create an empty project. Once the project has been created, the user must create the file that will contain the policies. This is possible in a way consistent with the practices of Eclipse plugins, by right-clicking the project within the Package Explorer view and choosing new file in the category PoSecCo within the menu *new*(see Figure 3).

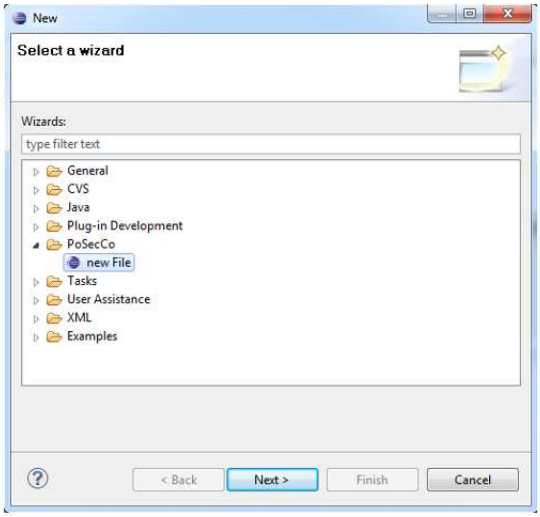

Fig. 3. The window used for the generation of a project.

At this point the plug-in will provide a window for choosing the name of the new file and the workspace where the file will be stored. When the file is created, the user can insert the data about the policy that he wants to create. Then, the user will typically insert the elements of the model using the forms associated with the entities in the tabs.

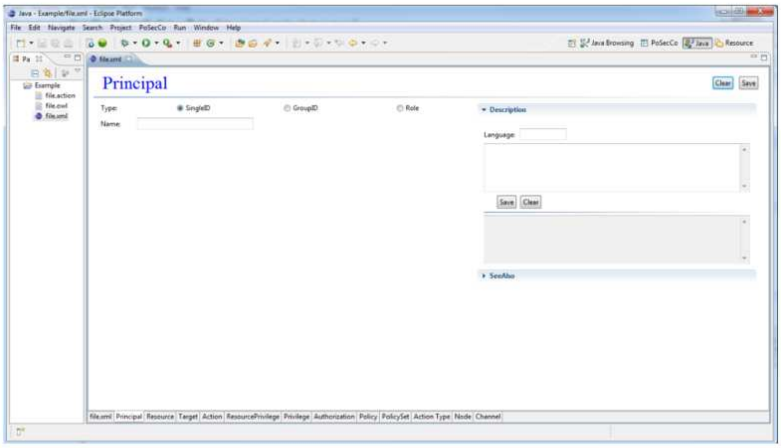

Fig. 4. Appearance of the plug-in.

To help the user during the creation and editing of data, a view has been introduced that allows the user to navigate between the *entities* entered above (see Figure 5).

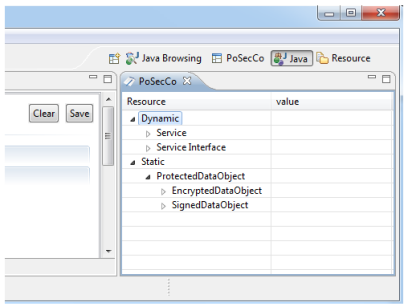

Fig. 5. The tree view of the instance of the model.

When the policy creation phase is complete, the user can use the plug-in not only to export the data into an XML file, but it can also create the ontology, expressed in an OWL file, according to the pattern seen in Section 2. This feature of the plug-in allows the user to obtain a representation of the security policy that can be processed by a standard OWL reasoner. In Figure 6 we can see both aspects described above.

Fig. 6. The buttons supporting the generation of the OWL model and the invocation of the reasoner.

The integration of reasoning was relatively direct, thanks to the flexible architecture that characterizes Eclipse. We created a view that shows the messages returned by HermiT, to provide feedback to the user and inform him of possible conflicts within the model. Stricter integrations that offer a greater level of interaction and control on the execution of the reasoner are planned for the next versions of the plugin. We are confindent that the Eclipse architecture will allow us to manage this further integration, something that would be harder if using closed-source or stricter frameworks.

6 Conclusions and Future Work

In this paper, we briefly described the structure of the PoSecCo plugin for Eclipse PEPP. We described the relationship between the PoSecCo security model and OWL and showed that the plugin can produce a representation of the PoSecCo model in OWL. We believe that a tool like the one presented in this paper can offer great support in the development of security policies and configurations. The use of the tool offers the designer the possibility to efficiently describe large systems and rely on well understood and verifiable security properties for open, dynamic environments, which require coordination across multiple organizations and integration of different data formats. The support provided by the Eclipse framework in the construction of such a system is essential, because Eclipse provides a powerful environment for tool integration and already makes available a large number of functions that are commonly required in the construction of such systems. In particular, it can integrate and manage plug-ins developed by other project partners and provide a unique environment that can manage all phases at all layers. Future work will be devoted to extend the Eclipse plug-in with richer functionality and modules of the Eclipse framework, further improving and speeding up the development process. In particular, we will create a module for exporting policies in XACML format and expand the checks carried out by the SWRL rules. We already know that the Eclipse framework has been the foundation for the construction of design environments for a variety of programming languages and methodologies. The work described in this paper testifies that Eclipse can also have an important role in the construction of modern environments for the management of security policies in large and integrated information systems.

References

1. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-Based Access Control Models. IEEE Computer **29** (1996) 38–47
2. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur. **4** (2001) 224–274
3. Li, N., Mitchell, J.C., Winsborough, W.H.: Design of a Role-Based Trust-Management Framework. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy, Washington, DC, USA, IEEE Computer Society (2002) 114–128
4. Anderson, A.: eXtensible Access Control Markup Language (XACML). Identity (2006) `http://www.oasis-open.org/committees/xacml/`.
5. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder Policy Specification Language. In: Proceedings of the International Workshop on Policies for Distributed Systems and Networks. POLICY '01, London, UK, Springer-Verlag (2001) 18–38
6. Jajodia, S., Samarati, P., Subrahmanian, V.S.: A Logical Language for Expressing Authorizations. In: Proceedings of the 1997 IEEE Symposium on Security and Privacy. SP '97, Washington, DC, USA, IEEE Computer Society (1997) 31–41
7. Kagal, L., Finin, T., Joshi, A.: A Policy Language for a Pervasive Computing Environment. In: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks. POLICY '03, Washington, DC, USA, IEEE Computer Society (2003) 63–74
8. 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. In Fensel, D., Sycara, K.P., Mylopoulos, J., eds.: International Semantic Web Conference, Springer (2003) 419–437
9. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference. Technical report, W3C, http://www.w3.org/TR/owl-ref/ (2004)
10. Lassila, O., Swick, R.R.: Resource Description Framework (RDF). Model and Syntax Specification. (1999)
11. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation, and Applications. In Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: Description Logic Handbook, Cambridge University Press (2003)
12. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A semantic web rule language combining OWL and ruleML. W3C member submission, World Wide Web Consortium, http://www.w3.org/Submission/SWRL/ (2004)