Towards Intelligent Web Services: The Web Service Modeling Ontology (WSMO)*

Cristina Feier[1], Dumitru Roman[1], Axel Polleres[1], John Domingue[2], Michael Stollberg[1], and Dieter Fensel[1]

[1] Digital Enterprise Research Institute, Institut für Informatik, Universität Innsbruck,AT
{firstname.lastname}@deri.org
[2] Knowledge Media Institute, The Open University, Milton Keynes, UK
J.B.Domingue@open.ac.uk

Abstract. The Semantic Web and the Semantic Web Services build a natural application area for Intelligent Agents, namely querying and reasoning about structured knowledge and semantic descriptions of services and their interfaces on the Web. This paper provides an overview of the Web Service Modeling Ontology, a conceptual framework for the semantical description of Web services.

1 Introduction

The Semantic Web [1] approach refers to the idea of making the overwhelming amount of data on the Web machine-processable. This shall be achieved by annotating web content with consensual formalizations of the knowledge published, often referred to under the common term of *ontologies*. Language proposals for describing ontologies include the Resource Description Framework (RDF) [2], RDF Schema [3], the Web Ontology Language (OWL) [4] and the Web Service Modeling Language (WSML) family of languages [5]. As indicated by the name of the latter, the current Web is not only a repository for static data, but furthermore offers interfaces to Web-accessible services, ranging from simple dynamically generated pages for pure information provision to more complex services for purchasing books, booking trips or trading with other internet-users over commercial or private marketplaces.

The next step after making the data on the Web machine-processable is facilitating the direct interaction of applications, i.e. services, over the Web. Making this vision real should not solely be viewed in the context of the Web as such, but has high potential benefits in the areas of Enterprise Application Integration and Business-to-Business Integration, being the two most prosperous application areas of current Information Technology. Current technologies around SOAP [6], WSDL [7] and UDDI [8], often subsumed under the term "Web services" only partly solve this integration problem by providing a common protocol (SOAP), interface description (WSDL) and directory (UDDI), but operating at a purely syntactic level.

* The work is funded by the European Commission under the projects DIP, Knowledge Web, InfraWebs, SEKT, SWWS, ASG and Esperonto; by Science Foundation Ireland under the DERI-Lion project; by the FIT-IT under the projects RW and TSC.

The goal of what is called Semantic Web services (SWS) [9] is the fruitful combination of Semantic Web technology and Web services. By using ontologies as the semantic data model for Web Service technologies Web Services have machine-processable annotations just as static data on the Web. Semantically enhanced information processing empowered by logical inference eventually shall allow the development of high quality techniques for automated discovery, composition, and execution of Services on the Web, stepping towards seamless integration of applications and data on the Web.

Two relevant initiatives have to be considered in the context of Semantic Web services. Chronologically, the first one is OWL-S [10], an upper level ontology for describing Web services, specified using OWL. We critizized several deficiencies in this model [11], and propose a new framework for Semantic Web Services by the Web Service Modeling Ontology (WSMO) [12] which refines and extends the Web Service Modeling Framework (WSMF) [13] to a meta-ontology for Semantic Web services. WSMF defines a rich conceptual model for the development and the description of Web services based on two main requirements: maximal decoupling and strong mediation.

WSMO is accompanied by a formal language, the Web Service Modeling Language (WSML), that allows one to write annotations of Web services according to the conceptual model. Also an execution environment (WSMX) [14] for the dynamic discovery, selection, mediation, invocation, and inter-operation of Semantic Web services based on the WSMO specification is under development.

This paper gives an overview of WSMO. In Section 2 the design principles of WSMO are presented. The basic concepts of WSMO are introduced in Section 3. Section 4 describes in more detail the WSMO elements, explaining what is needed for defining each of them and exemplifying their definition by making use of a scenario from the domain of e-tourism. Some conclusions are drawn in Section 5.

2 WSMO Design Principles

Since Semantic Web services aim at turning the Internet from an information repository for human consumption into a world-wide system for distributed Web computing by combining Semantic Web technologies and Web services, WSMO, as any other framework for Semantic Web services description needs to integrate the *basic Web design principles*, the *Semantic Web design principles*, as well as *design principles for distributed, serviceoriented computing for the Web*. This section enumerates and discusses the design principles of WSMO.

Web Compliance: WSMO inherits the concept of URI (Universal Resource Identifier) for unique identification of resources as the essential design principle of the Web. Moreover, WSMO adopts the concept of Namespaces for denoting consistent information spaces, supports XMLand, other W3C Web technology recommendations, as well as the decentralization of resources.

Ontology-Based: ontologies are used as the data model throughout WSMO, meaning that all resource descriptions as well as all data interchanged during service usage are based on ontologies. The extensive usage of ontologies allows semantically enhanced information processing as well as support for interoperability.

Strict Decoupling: each WSMO resource is specified independently, without regard to possible usage or interactions with other resources. This complies with the open and distributed nature of the Web.

Centrality of Mediation: mediation addresses the handling of heterogeneities that naturally arise in open environments. As a complementary design principle to strict decoupling, WSMO recognizes the importance of mediation for the successful deployment of Web services by making mediation a first class component of the framework.

Ontological Role Separation: User requests are formulated independently of (in a different context than) the available Web services. The underlying epistemology of WSMO differentiates between the desires of clients and available Web services.

Execution Semantics: In order to verify the WSMO specification, the formal execution semantics of reference implementations like WSMX as well as other WSMO-enabled systems provide the technical realization of WSMO.

Service versus Web service: A Web service is a computational entity which is able (by invocation) to achieve a goal. A service in contrast is the actual value provided by this invocation [15, 16]. Thus, WSMO does not specify services, but Web services, which are actually means to buy and search services.

3 WSMO Basic Concepts

WSMO defines the modeling elements for describing Semantic Web services based on the conceptual grounding set up in the Web Service Modeling Framework (WSMF) [13], wherein four main components are defined: ontologies, Web services, goals, and mediators. WSMO inherits these four top elements, further refining and extending them.

Ontologies represent a key element in WSMO since they provide (domain specific) terminologies for describing the other elements. They serve a twofold purpose: defining the formal semantics of the information, and linking machine and human terminologies.

Web services connect computers and devices using the standard Web-based protocols to exchange data and combine data in new ways. Their distribution over the Web confers them the advantage of platform independence. Each Web service represents an atomic piece of functionality that can be reused to build more complex ones. Web services are described in WSMO from three different perspectives: non-functional properties, functionality and behavior.

Goals specify objectives that a client might have when consulting a Web service, i.e. functionalities that a Web service should provide from the user perspective. The coexistence of goals and Web services as non-overlapping entities ensures the decoupling between request and Web service. This kind of stating problems, in which the requester formulates objectives without regard to Web services for resolution is known as *the goal-driven approach*, derived from *the AI rational agent approach*.

Mediators describe elements that aim to overcome the mismatches that appear between the different components that build up a WSMO description. The existence of mediators allows one to link possibly heterogeneous resources. They resolve incompatibilities that arise at different levels:

- data level - mediating between different used terminologies, more specifically solving the problem of ontology integration.

– process level - mediating between heterogeneous communication patterns. This kind of heterogeneity appears during the communication between Web services.

4 WSMO Modeling Elements

This section explains the WSMO modeling elements, describing their purposes and illustrating how they can be specified. A scenario from the domain of e-tourism is introduced in the first subsection as support for examples of modeling different WSMO elements.

4.1 E-tourism Scenario: Traveling from Innsbruck to Venice

In order to exemplify the modeling of WSMO elements a scenario from the domain of e-tourism is considered: an agent wants to buy a ticket to travel from Innsbruck to Venice on a certain date. The goal that specifies the intent of buying a ticket for a trip from Innsbruck to Venice is abstracted from this agent desire. A hypothetical Web Service called the ”Book Ticket Web Service” is considered for achieving the goal. This Web service allows to search and buy tickets for itineraries starting in Austria. The only accepted payment method is credit card. For the execution of the transaction, the credit card must be a valid PlasticBuy or GoldCard (two fictitious credit card brands).

4.2 Ontologies

WSMO specifies the following constituents as part of the description of an ontology: *non-functional properties*, *imported ontologies*, *used mediators*, *concepts*, *relations*, *functions*, *axioms*, and *instances*.

As an example for the header of an ontology, we present the header of the ”Trip Reservation Ontology”, used for specifying the ”Book Ticket Web Service” and the goal mentioned in the scenario. This ontology defines the necessary terminology for describing trip and reservation related information.

```
namespace {_"http://example.org/tripReservationOntology#",
  dc     _"http://purl.org/dc/elements/1.1#",
  loc    _"http://example.org/locationOntology#",
  po     _"http://example.org/purchaseOntology#",
  foaf   _"http://xmlns.com/foaf/0.1/",
  wsml   _"http://www.wsmo.org/wsml/wsml-syntax#",
  prs    _"http://example.org/owlPersonMediator#"
}
ontology  _"http://example.org/tripReservationOntology"
  nonFunctionalProperties
    dc#title hasValue "Trip Reservation Ontology"
    dc#creator hasValue _"http://example.org/foaf#deri"
    dc#format hasValue "text/x-wsml"
  endNonFunctionalProperties
  importsOntology{ _"http://example.org/locationOntology",
                   _"http://example.org/purchaseOntology"}
  usesMediator _"http://example.org/owlPersonMediator"
```

A namespace declaration can appear at the beginning of each WSML file. Such a declaration may comprise the default namespace and abbreviations for other used

namespaces. The listing above contains the namespace declaration from the top of the WSML file where the ”Trip Reservation Ontology” is specified. For simplicity, we will omit the namespace declarations for the other listings presented in this paper.

Non-functional properties can be specified for every WSMO element and they describe information that do not affect the element functionality like title, creator, etc. An ontology can *import other ontologies* either directly, when no conflicts need to be resolved, or indirectly, by using *ooMediators*, in case of data heterogeneities. The ”Trip Reservation Ontology”, imports two ontologies without any mediation and one OWL ontology by using one oomediator.

The basic blocks of an ontology are concepts, relations, functions, instances, and axioms. The examples provided in the next paragraphs for each of these components are taken from the ”Trip Reservation Ontology”, except for the relation example, which is taken part from the ”Purchase Ontology”.

Concepts are defined by their subsumption hierarchy and their attributes, including range specification. The range of the attributes can be a datatype or another concept. There are two kinds of attribute definitions: constraining definitions declared using the keyword `ofType` and inferring definitions declared using the keyword `impliesType`. The extension of a concept can be defined or restricted by one or more logical expressions embedded in axioms. The corresponding axioms should be referred in the `dc#relation` non-functional property of the concept. In the example below, the concept `tripFromAustria` is subsumed by the concept `trip` and its definition is completed with an axiom, `tripFromAustriaDef`, that specifies that a necessary condition for an individual to be an instance of this concept is to have as value of its `origin` attribute the location Austria.

```
concept trip
      origin impliesType loc#location
      destination impliesType loc#location
      departure ofType _date
      arrival ofType _date
concept tripFromAustria subConceptOf trip
      nonFunctionalProperties
        dc#relation hasValue tripFromAustriaDef
      endNonFunctionalProperties
axiom tripFromAustriaDef
      definedBy
        forall {?x ,?origin}
          (?x memberOf tripFromAustria
           implies
           ?x[origin hasValue ?origin] and
           ?origin[loc#locatedIn hasValue loc#austria]).
```

Relations describe interdependencies between a set of parameters. A relation declaration comprises the identifier of the relation and optionally: its arity, its superrelations, the domain of its parameters, and a set of non-functional properties. Like for concepts, axioms can be used for defining/constraining the relation extension.

Below is a relation that has a single argument, a credit card, and that holds when the credit card is valid. Accompanying this relation is an axiom that compares the credit card expiry date with the current date for establishing its validity.

```
relation validCreditCard(ofType creditCard)
      nonFunctionalProperties
        dc#relation hasValue ValidCreditCardDef
      endNonFunctionalProperties
axiom ValidCreditCardDef definedBy
      forall {?x, ?y}  (
```

```
        validCreditCard(?x)  impliedBy
        ?x[expiryDate hasValue ?y] memberOf creditCard and
        neg (wsml#dateLessThan(?y, wsml#currentDate())).
```

Functions are a special type of relations that have a unary range beside the set of parameters. Besides the typical declaration for a relation, when declaring a function one must include an axiom that states the functional dependency. The function `ticketPrice` is modeled as a relation with three parameters: the first one is a ticket, the second one is a currency, and the third one is the result returned by the function: the price of the ticket in the given currency.

```
relation ticketPrice (ofType ticket, ofType po#currency, ofType _integer)
      nonFunctionalProperties
        dc#relation hasValue {FunctionalDependencyTripPrice}
      endNonFunctionalProperties
axiom FunctionalDependencyTicketPrice
      definedBy
        !- ticketPrice(?x,?y,?z1) and ticketPrice(?x,?y,?z2) and ?z1 != ?z2.
```

Instances are embodiments of concepts or relations, being defined either explicitly, by specifying concrete values for attributes or parameters or by a link to an instance store. Below there are two instance declarations, one being an instance of the concept `trip`, and the other an instance of the relation `ticketPrice`.

```
instance tripInnVen memberOf trip
   origin hasValue loc#innsbruck
   destination hasValue loc#venice
   departure hasValue  _date(2005,11,22)
   arrival hasValue  _date(2005,11,22)
relationInstance ticketPrice(ticketInnVen, po#euro, 120)
```

Axioms are specified as logical expressions and help to formalize domain specific knowledge. We already presented some examples of axioms.

4.3 Web Services

A Web service is defined by *non-functional properties*, *imported ontologies*, *used mediators*, *one capability*, and *one or multiple interfaces*. Below is the declaration of the Book Ticket Web Service:

```
webService _"http://example.org/bookTicketWebService"
  importsOntology _"http://example.org/tripReservationOntology"
  capability BookTicketCapability
  interface BookTicketInterface
```

The *capability* of a Web service defines its functionality. It comprises the next elements: *non-functional properties*, *imported ontologies*, *used mediators*, *shared variables*, *precondition*, *postcondition*, *assumption*, and *effect*.

For declaring the basic blocks of the Web service capability, one should take a closer look at what the Web service offers to a client (*postcondition*), when some conditions are met in the information space (*precondition*), and how the execution of the Web service changes the world (*effect*), given that some conditions over the world state are met before execution (*assumption*). Preconditions, assumptions, postconditions and effects are expressed through a set of axioms. A set of *shared variables* can be declared, which

are implicitly all-quantified, and whose scope is the whole Web service capability. Informally, the logical interpretation of a Web service capability is: for any values taken by the shared variables, the precondition and the assumption implies the postcondition and the effect.

When its execution is successful, the ”Book Ticket Web Service” has as result a reservation that includes the reservation holder and a ticket for the desired trip (postcondition) if there is a reservation request for a trip with its starting point in Austria for a certain person (precondition) and if the credit card intended to be used for paying is a valid one and its type is either PlasticBuy or GoldenCard (assumption). As a consequence of the execution of the Web service, the price of the ticket will be deducted from the credit card (effect).

```
capability BookTicketCapability
    sharedVariables {?creditCard, ?initialBalance, ?trip,
                    ?reservationHolder, ?ticket}
    precondition
       definedBy
          ?reservationRequest[
               reservationItem hasValue ?trip,
               reservationHolder hasValue ?reservationHolder
           ] memberOf tr#reservationRequest and
           ?trip memberOf tr#tripFromAustria and
           ?creditCard[balance hasValue ?initialBalance
                       ] memberOf po#creditCard.
    assumption
       definedBy
         po#validCreditCard(?creditCard)and
         (?creditCard[type hasValue "PlasticBuy"] or
          ?creditCard[type hasValue "GoldenCard"]).
    postcondition
       definedBy
         ?reservation memberOf tr#reservation[
           reservationItem hasValue ?ticket,
           reservationHolder hasValue ?reservationHolder]and
        ?ticket[trip hasValue ?trip] memberOf tr#ticket.
    effect
       definedBy
          ticketPrice(?ticket, "euro", ?ticketPrice)and
          ?finalBalance= (?initialBalance - ?ticketPrice)and
          ?creditCard[po#balance  hasValue  ?finalBalance] .
```

The *interface* of a Web service describes its behavior. It has two distinct components: *choreography*, the description of the communication pattern that allows one to consume the functionality of the Web service, and *orchestration*, the description of how the overall functionality of the Web service is achieved by means of cooperation of different Web service providers.

The underlying model for describing choreographies and orchestrations is based on the Abstract State Machines methodology [17]. Thus, both choreography and orchestration descriptions have two parts: the *state* and the *guarded transitions*. The states of a choreography/orchestration are represented by an ontology whose content is dynamical (instances can be created and/or their values can be changed on-the-fly), while guarded transitions are if-then rules that specify transitions between states. The state signature is the non-dynamical part of the ontology. Due to space constraints we give here only an informal description (graphical and textual) of the interface of the ”Book Ticket Web Service”.

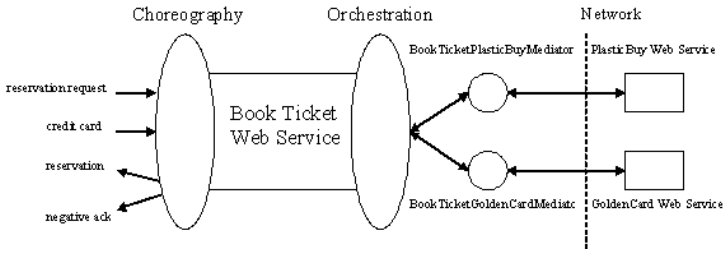

Fig. 1. The ”Book Ticket Web Service” interface

Three transition rules that describe the choreography of the service: the first one checks whether there is a reservation request for a trip that starts in Austria and a ticket for the desired trip in the Web service instance store. If this is the case, it creates a temporary reservation for that ticket. The next step is to wait for a credit card information. If it points to a valid credit card, a reservation is created (the second transition rule), otherwise a negative acknowledgement is created (the third transition rule).

As already said, the ”Book Ticket Web Service” accepts for payment PlasticBuy or GoldenCard credit cards. Depending on the type of the received credit card, the ”Book Ticket Web Service” must interact with the corresponding payment service in order for the payment to be done. Thus, two transition rules are necessary for describing the orchestration of the service, one for each possible case. As can be seen in figure 1, *wwmediators* are used to link the orchestration of the ”Book Ticket Web Service” with each of the corresponding payment services.

4.4 Goals

A goal in WSMO is described by *non-functional properties*, *imported ontologies*, *used mediators*, *requested capability* and *requested interface*. Besides ontologies, other goals can be reused within the definition of a goal via *ggMediators*.

The main element that appears in the declaration of a goal is the *requested capability*, which specifies the functionality required from a Web service. It is declared in the same way as a Web service capability. The user has also the possibility to specify the desired way of interacting with the Web service by declaring the *requested interface*.

Our scenario is that a user wants to buy a ticket from Innsbruck to Venice on a certain date. A goal is abstracted from this user desire that does not include the desired date for the trip, because this information is not considered relevant in the Web service Discovery phase. This goal is described in the listing below. In this case, the only element of the requested capability the user is interested in, is its postcondition.

```
goal _"http://example.org/havingAReservationInnsbruckVenice"
  importsOntology {
    _"http://example.org/tripReservationOntology",
    _"http://www.wsmo.org/ontologies/locationOntology"}
  capability
    postcondition
      definedBy
        ?reservation[
```

```
    reservationHolder hasValue ?reservationHolder,
    item hasValue ?ticket] memberOf tr#reservation and
?ticket[trip hasValue ?trip] memberOf tr#ticket and
?trip[origin hasValue loc#innsbruck,
      destination hasValue loc#venice] memberOf tr#trip.
```

4.5 Mediators

In the general case WSMO defines mediators by means of *non-functional properties*, *imported ontologies*, *source component*, *target component* and *mediation service*, where source and target component can be a mediator, a Web service, an ontology or a goal. Currently the specification counts with four different types of mediators:

- *ooMediators*: import the target ontology into the source ontology by resolving all the representation mismatches between the source and the target. The *oomediator* `owlPersonMediator` is used by the "Trip Reservation Ontology" for importing the "OWL Person Ontology"[3]:

```
ooMediator "http://example.org/owlPersonMediator"
  source _"http://daml.umbc.edu/ontologies/ittalks/person/"
  target _"http://example.org/tripReservationOntology"
  usesService  _"http://example.org/OWL2WSML"
```

- *ggMediators*: connect goals that are in a relation of refinement and resolve mismatches between those;
- *wgMediators*: link Web services to goals and resolve mismatches. Such a mediator connects the "Book Ticket Web Service" with the goal described in subsection 4.4:

```
wgMediator _"http://example.org/wgMed"
  source  _"http://example.org/BookTicketWebService"
  target _"http://example.org/havingAReservationInnsbruckVenice"
```

- *wwMediators*: connect several Web services for collaboration. The wwmediator `BookTicketPlasticBuyMediator` makes possible the interaction between the "Book Ticket Web Service" and the "PlasticBuy Web Service".

```
wwMediator _"http://example.org/BookTicketPlasticBuyMediator"
  source  _"http://example.org/BookTicketWebService"
  target _"http://example.org/PlasticBuyWebService"
```

5 Conclusions

This paper briefly introduced WSMO, one of the most salient efforts in the domain of Semantic Web services. Semantic Web Services are a key application area for Intelligent Agent Systems because of the necessity for semantic descriptions frameworks as a basis for such Intelligent Systems. Thus, WSMO and its formalization WSML provide the infrastructure for such systems.

Several implementations are already available or under development. The first version of the Web Service Execution Environment (WSMX) has been available since June 2004 at the SourceForge portal. Apart from WSMX which marks a reference architecture and implementation for a WSMO compliant execution environment there already exist several other ready-to-use implementations and tools for WSMO. The Internet Reasoning Service (*IRS-III*) is an infrastructure for publishing, locating, executing

[3] "http://daml.umbc.edu/ontologies/ittalks/person/"

and composing Semantic Web services, organized according to the WSMO framework. *WSMO Studio* is a Semantic Web Service editor compliant with WSMO. The WSMO Studio will be available as a set of Eclipse plug-ins that will allow easy reusability and extension from 3rd parties. wsmo4j is a Java API and a reference implementation for building Semantic Web services applications compliant with WSMO. Like WSMX, it is also being developed as an Open Source project. The Semantic Web Fred [18] combines combines agent technology with WSMO, in order to provide advanced support for Semantic Web applications.

As future work, we plan to further investigate the way in which agent-based solutions can be reused for achieving the automation of discovery, composition, and execution of Web services in the context of WSMO.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American **284** (2001)
2. Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and abstract syntax. Recommendation 10 February 2004, W3C (2004)
3. Brickley, D., Guha, R.V., eds.: RDF Vocabulary Description Language 1.0: RDF Schema. (2004) W3C Recommendation 10 February 2004.
4. Dean, M., Schreiber, G., eds.: OWL Web Ontology Language Reference. (2004) W3C Recommendation 10 February 2004.
5. WSML working group: WSML homepage (since 2004) http://www.wsmo.org/wsml/.
6. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.J., Nielsen, H., eds.: SOAP Version 1.2. (2003) W3C Recommendation 24 June 2003.
7. Chinnici, R., Gudgin, M., Moreau, J.J., Schlimmer, J., Weerawarana(eds), S.: WSDL. Working draft, W3C (2004) Available from http://www.w3.org/TR/wsdl20.
8. Clement, L., Hately, A., von Riegen, C., Rogers(eds), T.: UDDI Version 3. Uddi spec technical committee draft, OASIS (2004) Available from http://uddi.org/pubs/uddi_v3.htm.
9. McIlraith, S., Son, T.C., Zeng, H.: Semantic Web Services. IEEE Intelligent Systems, Special Issue on the Semantic Web **16** (2001) 46–53
10. Martin, D., ed.: OWL-S 1.1 Release. (2004) http://www.daml.org/services/owl-s/1.1/.
11. Lara, R., Roman, D., Polleres, A., Fensel, D.: A Conceptual Comparison of WSMO and OWL-S. In: Proc. of the European Conf. on Web Services. (2004)
12. WSMO working group: WSMO homepage (since 2004) http://www.wsmo.org/.
13. Fensel, D., Bussler, C., Ding, Y., Omelayenko, B.: The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications **1** (2002)
14. WSMX working group: WSMX homepage (since 2004) http://www.wsmx.org/.
15. Baida, Z., Gordijn, J., Omelayenko, B., Akkermans, H.: A Shared Service Terminology for Online Service Provisioning. In: Proc. of the 6th Int. Conf. on Electronic Commerce. (2004)
16. Preist, C.: A Conceptual Architecture for Semantic Web Services. In: Proc. of the Int. Semantic Web Conf. (ISWC 2004). (2004)
17. Gurevich, Y. In: Evolving algebras 1993: Lipari guide. Oxford University Press, Inc. (1995) 9–36
18. Stollberg, M., Roman, D., Toma, I., Keller, U., Herzog, R., Zugmann, P., Fensel, D.: Semantic Web Fred - Automated Goal Resolution on the Semantic Web. In: Proc. of the 38th Hawaii Int. Conf. on System Science. (2005)