ODEGSG Framework, Knowledge-Based Annotation and Design of Grid Services

Carole Goble[1], Asunción Gómez-Pérez[2], Rafael González-Cabero[2], and María S. Pérez-Hernández[3]

[1] Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK
[2] Ontology Engineering Group, Universidad Politécnica de Madrid, Campus de Montegancedo s/n, 28660 Boadilla del Monte, Madrid, Spain
[3] DATSI, Facultad de Informática, Campus de Montegancedo s/n, Universidad Politécnica de Madrid, 28660 Boadilla del Monte, Madrid, Spain
{asun, rgonza, mperez}@fi.upm.es

Abstract. The convergence of the Semantic Web and Grid technologies has resulted in the Semantic Grid. The great effort devoted in by the Semantic Web community to achieve the semantic markup of Web services (what we call Semantic Web Services) has yielded many markup technologies and initiatives, from which the Semantic Grid technology should benefit as, in recent years, it has become Web service-oriented. Keeping this fact in mind, our first premise in this work is to reuse the ODESWS Framework for the Knowledge-based markup of Grid services. Initially ODESWS was developed to enable users to annotate, design, discover and compose Semantic Web Services at the Knowledge Level. But at present, if we want to reuse it for annotating Grid services, we should carry out a detailed study of the characteristics of Web services and Grid services and thus, we will learn where they differ and why. Only when this analysis is performed should we know how to extend our theoretical framework for describing Grid services. Finally, we present the ODESGS Framework, which is the result of having applied the extensions identified to the aforementioned Semantic Web Services description framework.

1 Introduction

The Semantic Grid is the result of the convergence of the Semantic Web and the Grid technologies. Its definition of is created by modifying the Semantic Web definition given in [1]. The Semantic Grid is defined thus as an extension of the current Grid, in which information and services are given well-defined meaning for better enabling computers and people to work in cooperation. The requirements and research challenges of the Semantic Grid are identified in an unimpeachable manner in [2] and updated in [3], of which the most related to the knowledge-based markup are a) process descriptions that allow the (semi)automatic composition of services; b) annotation of all the contents in the system (resources, services, provenance data, etc.), which allows automatic discovery and must be done by means of an agreed interpretation (i.e. ontologies); c) context-aware decision support, or the context of the

B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, pp. 341 – 352, 2005.

Grid environment that must be annotated ; and d) the communities that users should be able to form, maintain and disband (this community term correspond with the Grid idea of Virtual Organization (VO) to be analyzed later) .

In addition to these requirements, the Semantic Grid should also be service-oriented, as the Grid is since the emergence of OGSA (Open Grid Service Architecture) [4]. Grid resources are wrapped with services and exposed via a WSDL file (i.e. a set of operations written in a standard XML language). OGSA redefines the concept of VO, a key element for Grid computing. They were considered a group of organizations and/or individuals that share resources in a controlled fashion [5]. Now VOs are considered to be the set of services that these organizations and/or individuals operate on and share [4] (plus some security policies). This idea of service-oriented VO, mixed with agent-oriented and dynamic view, is also described in [3]; in that paper, VOs are considered dynamic agents marketplaces. All these ideas of service orientation have became even more relevant since the appearance of GT4[1] and WSRF [6] which make Grid environments compliant with the most widely accepted Web services standards and technologies (WSDL, SOAP, etc.).

The Semantic Grid may reuse all the emerging technologies related to Semantic Web Services (i.e. IRS [7], OWL-S [8], ODESWS [9], WSMO [10], WSDL-S [11], etc.). These technologies and initiatives should not be considered as off-the-shelf technologies for the Semantic Grid because of the different nature of a Web service and a Grid service, and therefore between a Semantic Web Service (SWS) and a Semantic Grid Service (SGS).

In this paper we present the ODESGS Framework, an ongoing work carried out in the Ontogrid Project[2] (FP6-511513), which is the adaptation of the ODESWS Framework developed in the context of the EU project Esperonto[3] (IST-2001-34372); which was developed for annotating and creating complex SWSs, working at the Knowledge Level [12] thus enabling their discovery and (semi)automatic composition. As we have mentioned, we will start this paper enumerating the differences between SWS and SGS. Then, we will present the ODESGS Framework, which contains all the extensions that we have identified as necessary. This description of the ODESGS Framework comprises an enumeration of its design elements and a detailed description of a stack of ontologies used to describe SGSs. This stack will be called the ODESGS Ontology.

2 From SWSs to SGSs: Minding the Gap

As we have stated, one of the main points for the convergence of the Semantic Web and the Semantic Grid may lie in their service-oriented view. Before reusing the SWSs technology in the Grid, we should analyze the different nature of a Grid service (GS) and a Web service (WS). This analysis will help to clarify the terminology used.

A Web Service is an interface that describes a collection of operations that are network-accessible through standardized Web protocols whose features are described using a standard XML-based language [13][14]. Although there are other ways of

[1] http://www.globus.org/toolkit/
[2] http://www.ontogrid.net
[3] http://www.esperonto.net/

defining a WS, in this paper we adopt the aforementioned definition because it is the one that best captures the interface nature of what a WS is (and where its benefits come from). Other definitions consider WSs as modules or components, but these definitions break the low coupling principle that motivated the creation of WSs. In short, "It's not the components, it's the interfaces" [15].

SWSs, in the context of the Semantic Web, are the markup of WSs that will make them computer-interpretable, use-apparent and agent-ready [16]. This definition raises a simple but important question. Should SWSs be constrained with all the characteristics and limitations that the WS definition imposes (i.e. stateless interfaces, XML compliant, etc.)? Depending on our answer to this question we will distinguish between Semantic (Web Services) or (Semantic Web) Services. More precisely:

- A ***Semantic (Web Service)*** (S(WS)) retains all the characteristics of a WS, adding just semantic annotations to its domain, its inputs and outputs, and describing its functional properties (precondition, postconditions, etc.). However, It says nothing about the internal structure of the service, its state, etc. (as it remains being an interface). It is just a WSDL file plus some semantics (a clear example of this is WSDL-S [11]). S(WS)s have the great advantage of being upgraded easily from current technology to a semantically enhanced one.
- A ***(Semantic Web) Service*** ((SW)S) it is not constrained by the nature of a WS, as it can be a WS, an agent or anything that provides a service-oriented functionality for the Semantic Web. The description of a (SW)S goes far beyond the idea of an interface since we may find internal reasoning process descriptions, explicit lifecycle, state handling, and many other elements. Therefore, they can be considered a superset of S(WS)s.

Current SWSs initiatives are closer to the idea of (SW)Ss, because most of them describe, at least, the internal structure of complex SWSs (thus, they fall outside the semantic description of a simple net-work accessible interface).

Once we have briefly defined WSs and SWSs, let us see what GSs are. As we stated in the introduction, the service-oriented view of the Grid appeared in OGSA [4], where a service is defined as a network-enabled entity that provides some capability. GSs serve to achieve the virtualization (i.e. encapsulation independent of the implementation of physical resources such computational resources, storage resources, networks, programs, data-bases, etc.) of the shared resources.

Thus, by analogy with the aforementioned definition of SWS [16], a SGS is the markup of a GS that makes it computer-interpretable, user-apparent and agent-ready. Note that due to the more generic definition of what a GS is, we are less constrained in the markup of a GS than in the markup of a WS (remember the S(WS) and (SW)S differentiation that we stated above). A GS is not defined as an interface at all, which makes a big difference. However, and for the sake of completeness, we will also introduce a differentiation between Semantic (Grid Services) and (Semantic Grid) Services. Note that this differentiation is made by considering other terms than the S(SW)/(SW)S one.

Thus, we propose the following definitions:

- A ***Semantic (Grid Service)*** is just a "conventional" GS annotated to achieve its design, discovery, invocation and composition in a (semi)automatic way. In other words, a knowledge-aware GS.

- A ***(Semantic Grid) Service*** is a grid compliant knowledge service, a GS situated in the Knowledge Layer [3] that provides any kind of information, which is understood as knowledge that can be applied to achieve a goal, to solve a problem or to enact a decision. Possible examples could be a service that provides ontologies, a SGSs discovery service, a reasoner, etc.

From these definitions and after analyzing the nature of WSs SWSs, GSs and SGSs we have identified the following key features of a SGS not described by a SWS:

- ***VO.*** This is the first and, perhaps, the most important concept to remember. Despite trendy words like services and virtualization, Grid is about sharing resources under a certain set of rules. We should provide a formal and explicit description of a) the institution that is created by the sum of these services; b) the rules that govern the interaction between the entities involved; and c) the entities themselves (i.e. providers, consumers, and all the other roles that may coexist in a VO). The concept of VO does not exist in the SWSs field; SWSs are considered as isolated elements.
- ***Non-functional Properties.*** Non-functional properties are especially important in Grid environments. This is because a) discovery and composition is usually performed manually and depend on them; and b) many issues such as trust, quality of service and workload distribution are dependant on non-functional properties, and have much more importance in the Grid environment than in the Web environment. SWSs focus mainly on functional properties currently. Both types of information (functional and non-functional) should be handled (and therefore annotated).
- ***Provenance.*** Provenance information gives the origin and metadata information of a concrete enactment of a GS. With this information we are able to interpret the enactment results. Provenance seems to be very important in Grid environments, since Grid applications often deal with experiments where knowing which data and services are used to generate the results is very important.
- ***Complex Interactions.*** Interactions for SWSs tend to composed by the pair invoker/invoked-service (a refurbished XML version of the classic client/server interaction). In a Grid environment we should permit more complex interactions, i.e., defining more complex message exchanges and defining the different roles of the participants. These interactions plus the context of the enactment of the services can also be seen as service contracts [3].
- ***Resources.*** Although GSs hide resources, they should be also annotated and considered first-class citizens in a SGS description. Additionally, the handling of resources plus the nature of Grid environments clarify the definition of the real world, which is defined as the set of Grid resources that a SGS handles. In SWSs descriptions, the concept of the real world and the concept of the domain model (the abstract and formal representation of the world) are often confusing and confused.
- ***Transient GS Instances.*** This is one of the trickiest differences between SWSs and SGSs. The discovery, creation, and invocation of transient SGSs instances are a must. The possibility of specifying a concrete instance of a SWS, or even an scheduled invocation of one of its operations is not contemplated by current SWSs orchestrations and descriptions. SWSs work at a “class of services” level while SGSs should allow working at an “instance of service” level instead.

3 The ODESGS Framework

ODESGS Framework is a theoretical framework for annotating, discovering and composing SGSs in a (semi)automatic way. Its main assumptions are: the use of Problem-Solving Methods (PSMs) and ontologies for describing GSs in a formal and explicit way; thus the design and implementation phases of a SGS are clearly separated; VOs will be defined as the sum of SGSs plus some additional information about the hierarchy of roles of each SGS inside the VO; and some security and provenance related issues.

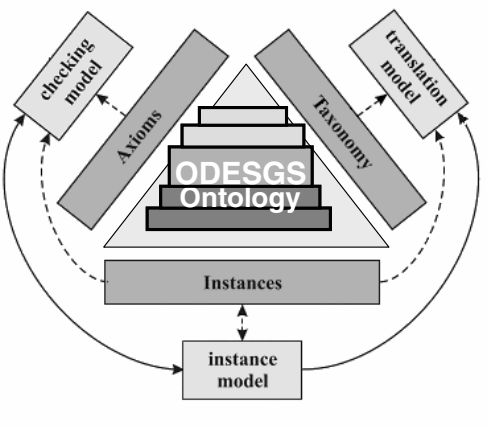

Fig. 1. ODESGS Framework design elements

This framework (see Figure 1) should provide a) service and stateful resource ontologies, rich enough to express the semantics required for service discovery and composition in a Grid environment; b) a set of rules to check whether the proposed design (for complex SGSs and VOs) is correct; and c) a way to translate from this design into a concrete implementation once the SGS has been designed. According to all these requirements, the following elements have been identified:

- ***ODESGS Ontology*.** To describe the features of VOs, SGSs, Grid resources, etc. a set of ontologies will be used. Ontologies are useful to represent their features in a formal and explicit way, which we will use in order to reason about them. This set of ontologies will be described in detail later.
- ***Instance Model.*** To design SGSs or VOs means to instantiate each of the ontologies of the stack and its relations. Each instance constitutes a model that specifies a SGS and VO.
- ***Checking Model.*** Once the instance model has been created, it is necessary to guarantee that such model does not present inconsistencies. Design rules will be needed to check this, particularly when ontology instances have been created automatically (as in the case of (semi) automatic composition). A set of design rules will be used to check both the SGSs annotated and designed by the user, and the different VOs created by aggregating these SGSs.
- ***Translation Model.*** Although SGSs and VOs are modelled in a high level of abstraction, they must be specified in different representational languages to enable

programs and external agents to access their capabilities. Therefore, once the instance that describes the SGS is created and checked, it should be automatically translated into any of the existing SGS or Grid service representational language.

4 ODESGS Ontology

Our aim is to come up with a service and data ontology, rich enough to express the semantics required for VOs formalization and SGSs discovery and composition. This means that VOs and service features should be explicitly and formally described. For this task, the use of ontologies seems to be the most appropriate solution. We propose a stack of ontologies that will complement each other in annotating all the features of a SGS. The stack is composed of the following ontologies a) one that describes VOs; b) another that describes the upper-level concepts that define the features of a SGS; c) a third ontology that describes the PSM to be used for representing both the internal structure and functional features of a SGS and the domain in which the service will be used (and, consequently, the domain of the VO); d) an ontology that defines the knowledge representation entities used to model a SGS and the domain ontology; and, finally, e) an ontology that describes the data types to be used in the domain ontology. Each of these ontologies is explained in the following sections.

4.1 SGS Ontology

The SGS ontology presumes that a SGS is decomposed in a set of operations. Each of these operations will be related to its corresponding Choreography, Model and Profile. Let us see each of these elements in detail and how they are related to elements of the PSM Description Ontology (which appears in Figure 2 and is fully explained below):

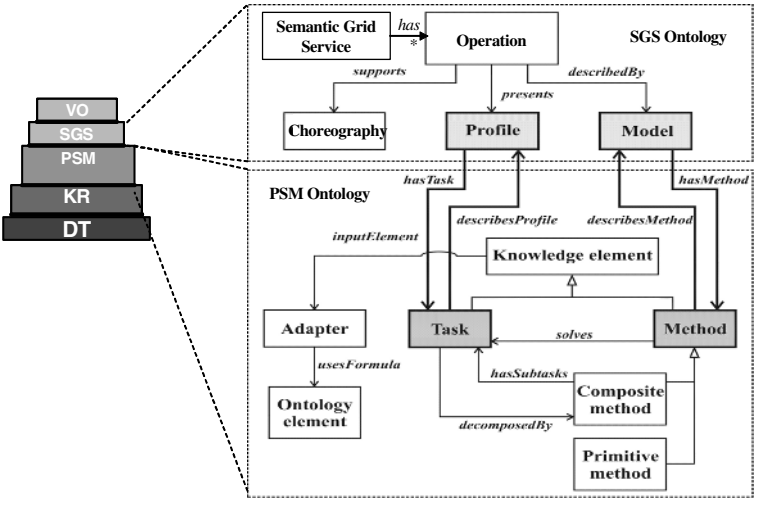

Fig. 2. Stack of ontologies main concepts and relations

- ***Profile.*** The profile stores both functional and non-functional properties of the SGS operation. We have identified a set of useful non-functional properties such as authors, description, accuracy, quality of service, performance, robustness, trust, etc. For describing the functional properties, the profile concept establishes relationships (*hasTask*) with the Task concept of the PSM ontology.
- ***Model.*** The Model concept defines a relationship (*hasMethod*) with the concept Method of the PSM Ontology. This means that a service operation will be described by a method, which solves or decomposes the task associated with the profile of the service operation. Moreover, the consistency in the relationships among the concepts of Task, Method, and SGS Operation are guaranteed; if a SGS operation is functionally described by a task, and executed by a method, there must be a relationship between this task and this method, being this method one of the set of methods that can solve this task.
- ***Choreography.*** The choreography of the operation describes the interaction that should be made to invoke its addressed operation in a formal way. Choreography describes both the messages inter-changed and the roles of those sending and receiving those messages. We will use those formalisms to those presented in [17] to formalize Web services choreographies and their concept of module replaceability, but we will extend it in some ways: a) we will use π-calculus [18] instead of CSS [19] (due to the changing and dynamic nature of the Grid); b) we will add semantic annotation to the messages exchanged, using the domain ontologies; and c) we will map the different actors appearing in the choreography with the roles that we have defined in the VO Roles Model (we will define them later).

4.2 PSM Ontology

Our approach for describing SGSs is based on the Problem-Solving Method paradigm. To decouple the functional features of a service from its internal specification, we propose to apply PSMs [20][21] when modelling SGSs (following the same approach that we did for describing SWSs in ODESWS [9]. A PSM is defined as a domain-independent and knowledge-level specification of the problem solving behaviour that can be used to solve a class of problems [21]. Our ontology for the description of PSM is based on the Unified Problem-solving Method Language (UPML) [23]. The UPML language was developed in the context of the IBROW project [23] with the aim of enabling the (semi) automatic reuse and composition of PSMs distributed throughout the Web. This objective seems to be similar to that of composing services; thus, it can be considered that the IBROW project highlights the close relation between PSMs and SWSs [24] (and SGSs by analogy since OGSA apparition).

- ***Task.*** It describes an abstract operation of independent domain to be solved, specifying the input/output parameters and the task competence; This task competence is composed of 1) preconditions and postconditions, which are logical expressions about the abstract representation of the domain (how this domain should be before and after the execution of the operation, respectively); and 2) assumptions and effects, which are logical expressions about the state of the real world (how the world should be for this operation to be applicable and how will be after the execution of the operation, respectively), being the real world the set of available Grid resources. Note that this task description is independent of the

method used for solving the task and that the PSM paradigm distinguishes between what we want to solve and how we are going to solve it.

- ***Method.*** It details the abstract reasoning process which is domain independent to achieve a task, describing both the decomposition of the general tasks into sub-tasks and the coordination of those sub-tasks to reach the required result. Note that we based our PSMs descriptions on UPML, and it does not define nor impose a language for describing the reasoning processes carried out by the methods. We propose to add a minimal set of programming primitives to describe the operational description of a composite method, a combination of which allows us to derive several basic workflow-like patterns [25]. The formalism that we will use beneath this workflow representation will be Kripke Structures and their translation to temporal logic (see [26] for a complete reference).
- ***Adapter.*** It specifies mappings among the knowledge components of a PSM, adapting a task to a method and refining tasks and methods to generate more specific components [27]. Therefore, adapters are used to achieve the reusability, since they bridge the gap between all the elements of a PSM.
- ***Domain Model.*** Domain Model introduces domain knowledge, and by means of adapters it is attached to the methods and tasks in order to represent a concrete description of an operation in a concrete domain (task and methods are domain independent, as defined before).

4.3 VO Ontology

VOs were originally defined in [5] as a set individuals/institutions defined by a set of resource sharing rules (these sharing rules specify what is shared, who is allowed to share, and the conditions under which sharing occurs). When OGSA appeared, VOs became defined by the services that they operate on and share and this was due to the wrapping of resources by means of Grid services. So, our VOs descriptions will initially be a set of SGSs descriptions. But there are still open issues that an additional formalism should solve. One of these challenges appearing in [28] is to make automatic decisions about which services could be in a VO and what should be their roles these services should have in the VO. We will try to solve this challenge by formalizing what VOs are. One advantage of formalizing VOs is the possibility of discovering VOs; we may think of several VOs and a user wanting to know which VO fits his/her expectations better. We will decompose a VO in:

- ***Metadata Properties.*** Additional non-functional information about the VO (security and trust information, geographical issues, date of creation, involved "real world" institutions, etc.).
- ***Roles Model.*** We will define the roles of SGSs in the VO by means of role taxonomies and a set of restrictions for each role.
  - Each VO will have a set of role taxonomies linked by subsumption relationships. This tree-shaped structure (or structures) contains the possible roles of the services (or external agents) that may interact or belong to the VO.
  - A set of different restrictions for belonging to a role will be defined for each of them. These restrictions will cover different aspects of what SGSs are in our definition. We will distinguish between: *non-functional restrictions*, which

constraint SGSs non-functional properties; *competence restrictions*, i.e., functional properties that a role membership imposes; *choreography restrictions*, (a role may impose certain message interchange compliance to the SGS just by defining an abstract choreography and a type of relationship (bisimulation, strong bisimulation, weak simulation, etc.) that the choreography of the service should have (see [18] for a definition of them); and *method restrictions* (we may impose certain restrictions to the orchestration/dataflow of a complex SGS).

With all these roles and restrictions, we may be able to a) know if a SGS can be added to a certain VO; b) know, in case that a service may belong to a certain VO, which of the different roles the SGS may play inside the VO; and c) use these roles to annotate the actors that appear in each SGS choreography, relating thus the interaction of a concrete service with the other SGSs that compose the VO.

- ***Provenance Model.*** We will initially follow the ideas formulated in ^{my}Grid Project[4] (for a detailed explanation we remit the reader to [29]). Provenance information provides the origin as well as and metadata information of a concrete enactment of a Grid service so as to be able to interpret the results.

4.4 KR Ontology and DT Ontology

The Knowledge Representation (KR) Ontology describes the primitives of the KR model, which contains descriptions about the knowledge and data used by the SGS. We have selected the WebODE knowledge model [30] as KR ontology. The KR Ontology is constructed on top of an ontology that describes the types of the concepts and attributes. This ontology will be based on the XML Schema Datatypes (XSD).

5 A Simple Example

Due to the lack of space we have chosen a very simple example to illustrate how a SGS description is defined, and how it is seamlessly included in the context of a semantically enhanced VO.

Let us suppose that we have a Grid portal that offers some functionality in some given domain. Before accessing any of the services that belong to the portal, the client (user, service, agent, whatever) should provide its identification and some kind of key that guarantees its identity. Note that we suppose that this must be done always before the invocation of any operation of the offered services in the portal.

Needless to say, the first step is the creation of all the ontologies that will be used for the definition of all the elements and models of the VO and all the SGSs that may fit in it. We suppose that in these ontologies, at least, concepts such us *Key, Credential, Identification* are defined, as top level concepts. We also assume that they are also refined, in order to achieve finer grained descriptions of these concepts.

Our very simple VO will comprise the set of GSs that are invoked from outside the portal to obtain some functionality, and some services that are used for authenticating and finding the privileges of the invokers. Therefore, we will define a very simple Roles Model, in which we define two roles, *Authentication Services* and

[4] http://www.mygrid.org.uk/

Offered Services. How are we going to characterize each of them? We may define them by setting restrictions on their operations and in their choreographies. An *Authentication Service* will be defined as a service that should have at least an *authenticate* operation, and this operation should receive, in its invocation process, first an instance of the *Identification* concept and then an instance of the *Key* concept, giving as a result a *Credential* instance. An *Offered Service* in our VO could be any service that works in its domain. We just impose that the first action performed by each of its operations Choreography is to invoke the *authenticate* operation of an *Authentication Service* belonging to the VO.

Once we have defined this simple VO (we left out the Provenance Model and all the Metadata information for the sake of simplicity), we are going to show how to describe two SGSs that belong to the VO. We already know, thanks to the aforementioned Role Model constraints, that an *Authentication Service* should have at least one *authenticate* operation. In this example we will show how to describe this *authenticate* operation of two services. One of them is a simple service called *SimpleSignIn,* which *authenticate* operation receives a user name and its password (both as a string of characters). The other is *SequreSignIn (*a complex SGS that invokes other SGSs), whose *authenticate* operation receives more complete identification information, and an X.509 digital certificate as its key.

In both services operations, the inputs are instances of concepts subsumed by the concepts *Identification* and *Key* and the output in both cases is a credential, so both operations can be described using the same abstract high level task, that we will call *Authenticate,* even thought their methods may be completely different.

Once we have defined the *Authenticate* task, we have described what the *authenticate* operation does. But how does the *Authenticate* task achieve its results? We define that by means of methods. We build two methods, an atomic method called *SimpleAuthenticationProcess*, and a complex method called *ComplexAuthentication Process.* Atomic method means that it does not decompose the task into subtasks, as the complex method does. Because of that, the complex method should define a) what are the subtasks in which *ComplexAuthenticationProcess* decompose the *Authenticate* task; b) how they interchange data between them, i.e. the dataflow; and c) how they are orchestrated, i.e. the controlflow. Figure 3 shows how the *ComplexAuthenticationProcess* is defined, by means of a dataflow diagram and a workflow.

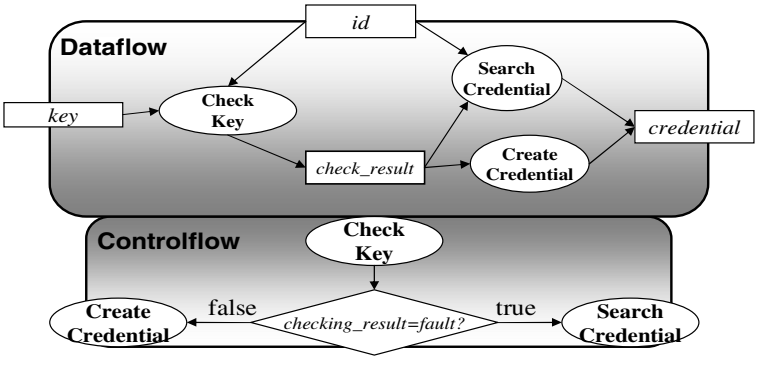

Fig. 3. Dataflow and Controlflow of the *ComplexAuthenticationProcess* method

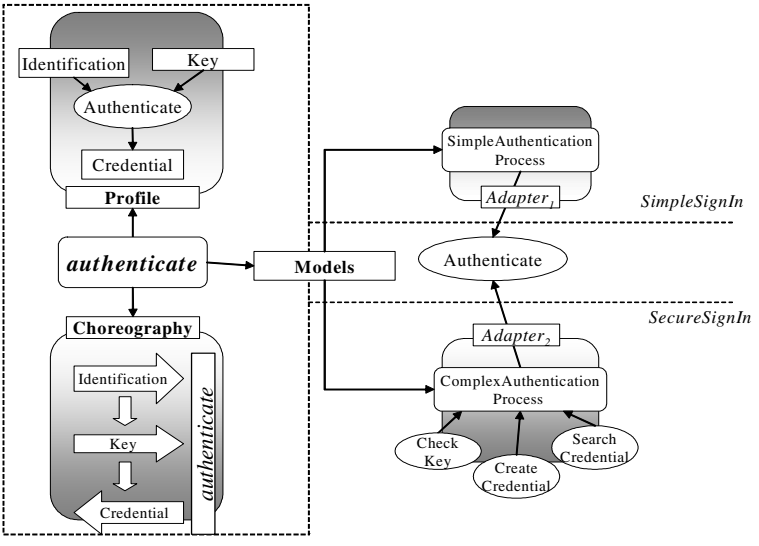

Fig. 4. The different elements that describe the *authenticate* operation

These methods will be glued to the *Authenticate* task by means of adapters. Adapters will also be used to glue the tasks and methods to the domain knowledge.

To sum up, the semantic description of the model of the *authenticate* operation of the *SimpleSignIn* GS will be the *Authenticate* task, the *SimpleAuthenticationProcess* method and all the assumptions and mappings that $Adapter_1$ may contain. The semantic description of the model of the *SequreSignIn authenticate* operation will be also the *Authenticate* task, the *ComplexAuthenticationProcess* method and all the assumptions and mappings that $Adapter_2$ may state.

Figure 4 shows a simplified summary of how both *autenthicate* operations are described, defining their Choreography, Profile (that we have supposed to be equal) and respective Models.

Acknowledgements

This work has been partially financed by the Ontogrid Project (FP6-511513) and by a grant provided by the Comunidad Autónoma de Madrid (Autonomous Community of Madrid).

References

1. Hendler, J. 2001. Agents and the Semantic Web. IEEE Intelligent Systems, 16(2):30–37.
2. De Roure D., Jennings N. R., and Shadbolt N. R.,(2001) Research Agenda for the Semantic Grid: A Future e-Science Infrastructure, NeSC, Edinburgh, UK UKeS-2002-02.
3. De Roure, D., Jennings, N. R. and Shadbolt, N. R. (2005) The Semantic Grid: Past, Present and Future. Procedings of the IEEE.
4. Foster I., C. Kesselman, J. N., and Tuecke S., (2002) Grid Services for Distributed System Integration Computer, vol. 35.

5. Foster I., Kesselman C., and Tuecke S. (2001) The anatomy of the Grid: Enabling scalable virtual organi-zations. Lecture Notes in Computer Science 2150
6. Czajkowski K., Ferguson D.F., Foster I., Frey J., Graham S., Sedukhin I., Snelling D., Tuecke S., Vam-benepe W., (2003) The WS-Resource Framework
7. Motta, E., Domingue, J., Cabral, L., Gaspari, M.:(2003) IRS-II: A Framework and Infrastructure for Semantic Web Services. ISWC 2003. LNCS Vol. 2870. Springer-Verlag
8. OWL Services Coalition (2004), OWL-S 1.1 Release: Semantic Markup for Web Services", Available: http://www.daml.org/services/owl-s/1.0/owl-s.pdf
9. Gómez-Pérez, A., González-Cabero, R., and Lama, M. (2004), A Framework for Design and Composing Semantic Web Services", IEEE Intelligent Systems, vol. 16, pp. 24–32
10. WSMO Working Group, (2004) http://www.wsmo.org/2004/d2/v1.0/
11. Akkiraju, R., Farrell, J. Miller J. Nagarajan M.(2005) WSDL-S Technical NoteVersion 1.0 Web Service Semantics
12. Newell, A.(1982) The knowledge level Artificial Intelligence., vol. 18, pp. 87--127.
13. Kreger, H. (2001) Web Services Conceptual Architec-ture. http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf
14. Curbera, F.; Nagy, W.A.; and Weerawana, S. (2001). Web Service: Why and How?. In Proceedings of the OOPSLA-2001 Workshop on Object-Oriented Ser-vices. Tampa, Florida.
15. Kayne D. (2003) Loosely Coupled, The Missing Pieces of Web Services Rds Associates Inc
16. McIlraith, S.; Son, T.C. and Zeng, H. (2001) Semantic Web Services. IEEE Intelligent Systems, 16(2):46–53.
17. Brogi A.,Canal C.,Pimentel E.,and Vallecillo A..(2004) Formalizing WS choreographies. In Proc. of First International Workshop on Web Services and Formal Methods
18. Milner R., (1999) Communicating and Mobile Systems: the Pi-Calculus Cambridge University Press ISBN: 0521658691
19. Milner, R., Communication and Concurrency (1989) Prentice Hall. ISBN: 0131149849
20. Benjamins, V.R., and Fensel, D. eds. (1998). Special Issue on Problem-Solving Methods. International Journal of Human-Computer Studies, 49(4): 305–313.
21. Motta, E. (1999), Reusable Components for Knowledge Modelling, IOS Press
22. Fensel D., Motta E., van Harmelen F., Benjamins V.R., Crubezy M., Decker S., Gaspari M., Groenboom R., Grosso W., Musen M., Plaza E., Schreiber G., Studer R., and Wielinga B. (2003), The Unified Problem-Solving Method Development Language UPML. Knowledge and Information Systems (KAIS): An International Journal
23. Benjamins, V.R.; Wielinga, B.; Wielemaker, J.; and Fensel, D. (1999). Brokering Problem-Solving Knowledge at the Internet. In Proc. (EKAW-99): Springer-Verlag.
24. Benjamins, V.R. (2003), Web Services Solve Problems, and Problem-Solving Methods Provide Services. IEEE Intelligent Systems, 18(1):76–77.
25. van der Aalst, W.P.; ter Hofstede, A.H.; Kiepuszewski, B.; and Barros, A.P.. Workflow patterns. Distrib-uted and Parallel Databases, 14(2):5–51.
26. Clarke E. M., Grumberg O., Peled D.A., (2000), Model Checking The MIT Press ISBN: 0262032708
27. Fensel, D. (1997), The Tower-of-Adapter Method for Developing and Reusing Problem-Solving Methods. In Proc. of the 7thKnowledge, Modeling and Management Workshop, 97–112: Springer-Verlag.
28. Foster I., Jennings N. R., and Kesselman C (2004) Brain meets brawn: Why grid and agents need each other. In Proc. 3rd Int. Conf. on Autonomous Agents and Multi-Agent Systems, New York, USA
29. Zhao J., Stevens R., Wroe C., Green-wood M. and Goble C. (2004) The Origin and History of in silico Experiments In Proc. of the UK e-Science All Hands Meeting.
30. Arpírez J.C., Corcho O., Fernández-López M., and Gómez-Pérez A (2003).: WebODE in a nutshell. AI Magazine.