GRISINO – A Semantic Web Services, Grid Computing and Intelligent Objects Integrated Infrastructure

Tobias Bürger, Ioan Toma, Omair Shafiq, Daniel Dögl, and Andreas Gruber

Abstract. Existing information, knowledge and content infrastructures are currently facing challenging problems in terms of scalability, management and integration of various content and services. The latest technology trends, including Semantic Web Services, Grid computing and Intelligent Content Objects provide the technological means to address parts of the previously mentioned problems. A combination of the three technologies could provide a sound technological foundation to build scalable infrastructures that provide highly automated support in fulfilling user's goals.

This paper introduces GRISINO, an integrated infrastructure for Semantic Web Services, Intelligent Content Objects and Grid computing, which may serve as a foundation for next generation distributed applications.

1 Introduction

The GRISINO[1] project investigates the use of semantic content models in service oriented architectures based on Semantic Web Services- and Grid-Technology [13] by combining three technology strands: Semantic Web Services [6], Knowledge Content Objects [2] and Grid Computing [7]. By that, GRISINO aims at defining and realizing intelligent and dynamic business processes based on dynamic service discovery and the internal state of complex objects. Advantages of this approach

Tobias Bürger, Ioan Toma, Omair Shafiq
Semantic Technology Institute - STI Innsbruck, University of Innsbruck, Austria
e-mail: {tobias.buerger,ioan.toma,omair.shafiq}@sti2.at

Daniel Dögl
Uma Information Technology GmbH, Vienna, Austria
e-mail: daniel.doegl@uma.at

Andreas Gruber
Salzburg Research Forschungsgesellschaft mbH, Salzburg, Austria
e-mail: andreas.gruber@salzburgresearch.at

[1] http://www.grisino.at

S. Schaffert et al. (Eds.): Networked Knowledge - Networked Media, SCI 221, pp. 113–128.
springerlink.com

include the possibility to establish service based processes ad-hoc based on the user requirements or their alteration during run-time based on the state of the intelligent content objects. The main output of the project is a test bed for experimentation with complex processes and complex objects that takes user requirements into account and fulfils them by dynamically integrating the three underlying technologies. For this testbed, advanced prototypes of each of the technology strands are combined:

- The **Web Service Modelling Ontology (WSMO)** [11], the Web Service Modelling Language (WSML)[2] and the Web Service Modelling Execution Environment (WSMX)[3] as a framework for the description and execution of Semantic Web Services,
- **Knowledge Content Objects (KCOs)** as a model for the unit of value for content to be exchanged between services, together with its management framework, the Knowledge Content Carrier Architecture (KCCA) [2].
- The **Globus toolkit**[4] as an existing Grid infrastructure.

In this chapter we will detail the main results of the GRISINO project: its architecture (section 2) and the core parts of the architecture which realize the integration of the three technologies, i.e. a set of transformers between the protocol and description standards used (section 3 and 4). Furthermore, we provide details about the proof of concept implementation which serves to demonstrate the functionality and interoperability within the GRISINO testbed in section 5.

2 GRISINO Architecture

One of the major driving forces for the Web and its future derivatives is content which can range from multimedia data (with some metadata) to "intelligent objects", i.e. content that itself, can either exhibit behavior or at least, carry semantic information that can (and must) be interpreted by the services on the Semantic Grid. The GRISINO system architecture as shown in Figure 1 provides a set of APIs and an implementation of these APIs to ease the handling and development of applications which intend to use the three technologies together:

- the GRISINO API which gives application developers easy access to the combined functionality of the three technologies.
- the Transformer API including protocol transformations between the technologies,
- the Selector API issuing calls to transformers or the foundational API, and
- the foundational API, which is an abstracted view onto the APIs of the core technologies.

Most notably the GRISINO system architecture includes extensions to the core components that enable communication between the technologies. This includes:

[2] `http://www.wsmo.org/wsm`
[3] `http://www.wsmx.org`
[4] `http://www.globus.org/toolkit/`

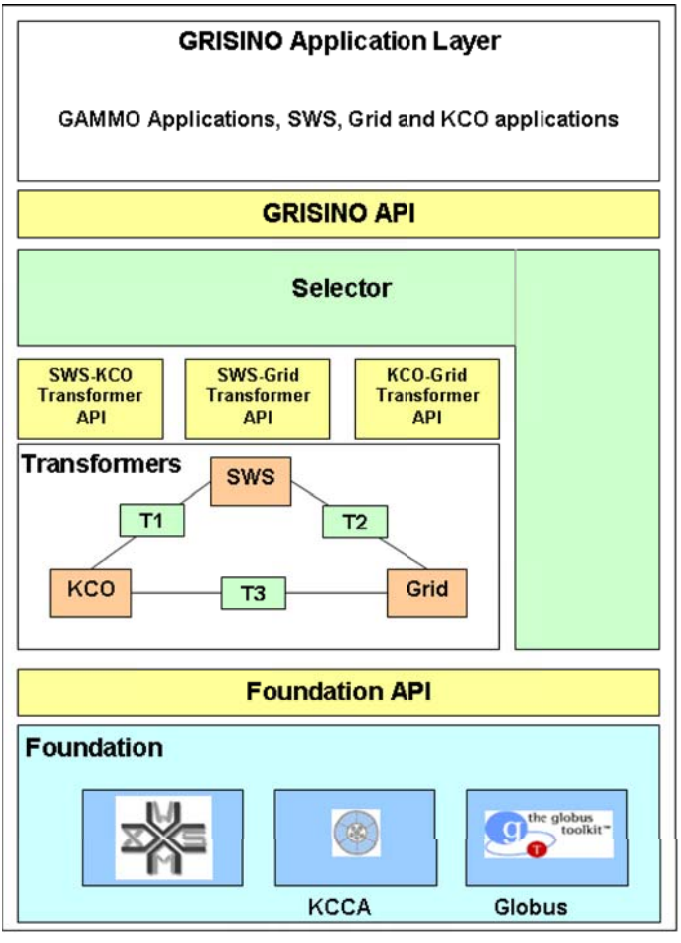

Fig. 1 GRISINO System Architecture

- an extension of WSMX for the interpretation of KCOs,
- a semantic layer for services offered by KCCA to enable their discovery and
- an extension of the Globus toolkit which extends Globus with a semantic layer in order to handle Grid services like other SWS.

The GRISINO system architecture integrates specific SWS and Grid solutions because of the existence of a wide variety of different and diverse approaches: We based our efforts on WSMO and WSMX as execution platforms because they are being well supported by an active research community to handle SWS. Furthermore we are using the Globus Toolkit as being the most widely used Grid computing toolkit which is fully compatible with the OGSA[5] - and Web Service Resource

[5] http://www.globus.org/ogsa/

Framework (WSRF) specifications[6]. The integration of Semantic Web Services and Grid computing includes the extension of the Semantic Web Services infrastructure to model Grid Services and resources on the Grid in order to realize the vision of the Semantic Grid. Benefits of this integration include:

- Resources on the Grid may profit from machine reasoning services in order to increase the degree of accuracy of finding the right resources.
- The background knowledge and vocabulary of a Grid middleware component may be captured using ontologies. Metadata can be used to label Grid resources and entities with concepts, e.g. for describing a data file in terms of the application domain in which it is used.
- Rules and classification-based reasoning mechanisms could be used to generate new metadata from existing metadata, for example describing the rules for membership of a virtual organization and reasoning that a potential member's credentials are satisfactory for using the VO resources.
- Activities like Grid Service discovery or negotiation of service level agreements can be potentially enhanced using the functionalities provided by Semantic Web Service technologies.
- Searches / discovery of SWS can be seamlessly extended to Grid Services.

The integration of SWS and KCO technologies will benefit from each other in several different aspects. In a KCO various kinds of information are modeled in so called semantic facets that allow to deal with KCOs in different situations. A more standardized exposition of KCO facet information would allow to base the actions that take place in a (goal-based) Web Service execution on the facet information of KCOs: for example to search for a KCO that contains certain content or to match a certain licensing schema. Also choreography could be based on facet information, e.g. to fulfill a special licensing schema where you first have to pay before you consume the content. Another benefit would be that these services could also be automatically discovered, which represents a key requirement for ad-hoc instantiation. Further benefits of the integration of SWS and KCO/KCCA include:

- Goal-based Web service execution can be based on the various kinds of information which is modeled in so called semantic facets inside KCOs; e.g. to search for a KCO that contains certain content or to match a certain licensing scheme.
- Choreography of Web services can be based on facet information, e.g. to fulfil a special licensing scheme in which you first have to pay before you consume the content.
- Plans that describe how to handle content and which are modeled inside a KCO can be automatically executed by using SWS or Grid services.

The following section will provide further details about two of the three major aspects of the integration, i.e. the integration of SWS and Grid, as well as the integration of SWS and KCOs.

[6] `http://www.globus.org/wsrf/`

3 SWS-Grid Transformer

The main task of this transformer (mentioned as T2 in the Figure 1) is the realization of the link between SWS based systems and Grid Computing systems. Our approach was to extend and refactor an existing SWS solution, namely the Web Service Modeling Ontology, Language and Execution Environment with Grid concepts in order to address Grid related requirements. The resulting modeling framework for Semantic Grid enriches the OGSA with semantics by providing a Grid Service Modeling Ontology (GSMO)[7] as an extended version of WSMO.

Based on the proposed conceptual model for Semantic Grid services, a new language called GSML (Grid Service Modelling Language) was developed that inherits the syntax and semantics of the WSML language and adds a set of additional constructs reflecting the GSMO model. Last but not least an extension of the Web Service Modeling Execution Environment (WSMX), called Grid Service Modeling Execution Environment has been proposed. More details about the conceptual model, the language and the new execution environment are available in [12].

3.1 Extensions to the WSMO Conceptual Model

The conceptual model of Semantic Web Services provides a set of guidelines or recommendations on how Semantic Web Service descriptions should look like. The Web Service Modeling Ontology (WSMO) [11] refers to the concepts it defines as its top level elements. WSMO has four top-level elements, i.e. Ontologies, Web Services, Mediators and Goals. We have extended the WSMO conceptual model to model Semantic Grid Services based on an analysis of the GLUE schema [1] for which we provided semantic annotations. The proposed extended version, called GSMO has 6 major top level entities which were either newly added to the WSMO conceptual model, are refinements of original entities or are entities which are inherited from the WSMO model. The GSMO elements are graphically represented in Figure 2: The elements *GSMO*, *Job*, *VO*, *Resources*, *Computational Resource*, *Data Resource* were newly added, the element *Grid Service* is the redefined element and finally the elements *Ontology* and *Mediator* have been inherited or adopted:

- *Job* represents the functionality requested, specified in terms of what has to be done, what are the resources needed, etc. A Job is fulfilled by executing one or more Grid Services. Ontologies can be used as domain terminology to describe the relevant aspects. Job as one of the top level entities of GSMO is adapted from WSMO Goals and is taken in GSMO as its extended version.
- *Ontologies* provide the terminology used by other GSMO elements to describe the relevant aspects of a domain. This element has been inherited from the WSMO top level entity as Ontologies.

[7] `http://www.gsmo.org/`

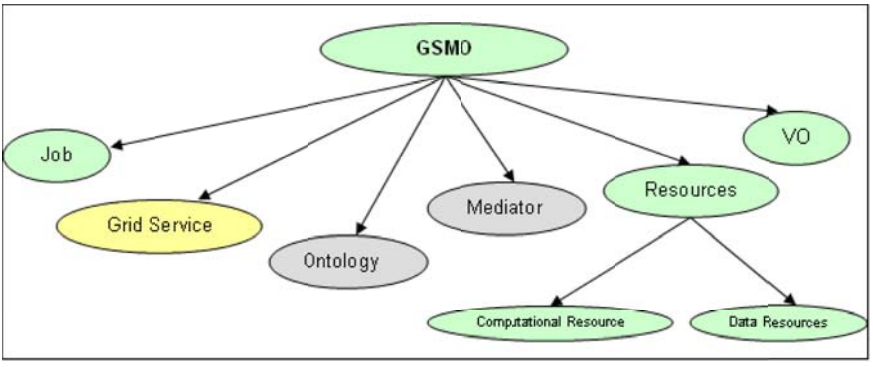

Fig. 2 Grid Service Modeling Ontology (GSMO)

- *Grid Service* describes the computational entity providing access to physical resources that actually perform the core Grid tasks. These descriptions comprise the capabilities, interfaces and internal working of the Grid Service. All these aspects of a Web Service are described using the terminology defined by the ontologies. The Grid Service top level entity has been adopted from WSMO's Web Services as its top level entity.
- *Mediators* describe elements that overcome interoperability problems between different WSMO elements. Due to the fact that GSMO is based on WSMO, it will be used to overcome any heterogeneity issues between different GSMO elements. Mediators resolve mismatches between different used terminologies (data level); communicate mismatches between Grid services (protocol level) and on the level of combining Grid Services and Jobs (process level).
- *Resources* describe the physical resources on the Grid which can be further classified into computing resources and storage resources. These computation- and storage-resources are key elements of the underlying Grid.
- The *Virtual Organization* element describes any combination of different physical resources and Grid Services formed as virtual organizations on the Grid. This element will help in automated virtual organization formation and management.

3.2 Extensions to the WSML Formal Language

Based on the conceptual model for Semantic Grid services presented in the previous section, this section introduces the basic constructs towards a semantic language for describing entities in the realm of the Semantic Grid. We propose a new language which is based on an existing language for Semantic Web services, namely the WSML language. The new language called GSML (Grid Service Modelling Language) inherits the syntax and semantic of the WSML language. Additional constructs not defined in WSML such as *VO* and *resource* can be used to semantically

describe Virtual Organizations and resources on the Semantic Grid. The constructs *webService* and *goal* from WSML are replaced by *gridService* and *job*.

As mentioned above, GSML follows the conceptual model of GSMO defining a clear syntax for each of the elements described in the previous section. The top level constructs introduced by GSML will be further described below:

A Grid service (gridService) in GSML has the following structure:

```
gridService = 'gridService' id? header* capability? interface*
   usesResources* belongsToVOs*
```

The **id**, **header**, **capability** and **interface** constructs from a Grid service definition are defined in the same way as described in WSML. Additionally the **usesResources** and **belongsToVOs** constructs with n-ary cardinality could be used to specify the resources used by the service in order to provide its functionality, respectively the VOs the service belongs to. A simplified example of a Grid service which provides movie rendering functionality is given below:

```
namespace {_"http://www.gsmo.org/movieRenderGS#",
dc_"http://purl.org/dc/elements/1.1#",
rO_"http://www.gsmo.org/renderOntology#"}
gridService_"http://www.gsmo.org/movieRenderGS.wsml"
    nonFunctionalProperties
        dc#title hasValue "Movie Render Grid service"
        dc#publisher hasValue "GSMO"
    endNonFunctionalProperties
    capability
        sharedVariables {?model}
        precondition
            definedBy ?model memberOf rO#Model.
        postcondition
            definedBy ?movie memberOf rO#lMovie and rO#
               hasModel(?movie,?model).
    interface MovieRenderServiceInterface
    choreography MovieRenderServiceChoreography
    orchestration MovieRenderServiceOrchestration
    usesResources { _"http://www.gsmo.org/resources#proc1,
        _"http://www.gsmo.org/resources#mem1}
    belongsToVOs { _"http://www.gsmo.org/resources#VO1,
        _"http://www.gsmo.org/resources#VO2}
```

The **usesResources** and **belongsToVOs** are defined as follows:

```
usesResources = 'usesResources' idlist

belongsToVOs = 'belongsToVOs' idlist
```

The **idlist** construct from the above definitions are the IDs of resources, respectively VOs. According to the principle inherited from WSML, the elements in GSML are identified mainly by IRIs, and thus idlist is a list of IRIs.

A user job (**job**) in GSML has similar structure than a **gridService** construct. Additionally an application element can be defined to explicitly specify the application to be run:

```
job = 'job' id?
    header* capability? interface* usesResources*
    belongsToVOs* application?
application = 'application' id? header* name? version?
    executable? argument* environment* input? output?
    error? working_directory?
```

Equally as in the **gridService** construct definition, the **id**, **header**, **capability** and **interface** constructs are inherited from WSML. The **usesResources** and **belongsToVOs** constructs are to be used in the same way as described above for the **gridService** construct. A simplified example of a job specification, a request for a movie rendering is given below:

```
namespace {_"http://www.gsmo.org/movieRenderGS#", dc
_"http://purl.org/dc/elements/1.1#", rO
_"http://www.gsmo.org/renderOntology#"}

job _"http://www.gsmo.org/MovieRender.wsml"
    nonFunctionalProperties
        dc#title hasValue "MovieRender Grid service"
        dc#publisher hasValue "GSMO"
    endNonFunctionalProperties
    capability
        sharedVariables {?model}
        precondition
            definedBy ?model memberOf rO#Model.
        postcondition
            definedBy ?movie memberOf rO#1Movie and rO#hasModel
            (?movie,?model).
    interface MovieRenderServiceInterface
        choreography MovieRenderServiceChoreography
        orchestration MovieRenderServiceOrchestration
    usesResources { _"http://www.gsmo.org/resources#proc1}
    belongsToVOs { _"http://www.gsmo.org/resources#VO1}
```

Equally, a job can be specified using the application element instead of the capability element. In this case the functionality is explicitly specified by naming the application that needs to be executed to fulfill the job. The elements of an application description include: the name of the application (**name**), the version of the application (**version**), the main executable file of the application (**executable**), the arguments (**argument***), any additional libraries needed to run the application (**environment**), the input data for the application specified in the input file (**input**), the output file (**output**), the error file (**error**) and finally the **working_directory**.

```
namespace {_"http://www.gsmo.org/movieRenderGS#", dc
_"http://purl.org/dc/elements/1.1#", rO
_"http://www.gsmo.org/renderOntology#"}

job _"http://www.gsmo.org/MovieRender.wsml"
    nonFunctionalProperties
        dc#title hasValue "MovieRender Grid service"
        dc#publisher hasValue "GSMO"
    endNonFunctionalProperties
            usesResources { _"http://www.gsmo.org/resources#proc1}
            belongsToVOs { _"http://www.gsmo.org/resources#VO1}
            application
    nonFunctionalProperties
        dc#title hasValue "MovieRender application "
        dc#publisher hasValue "GSMO"
    endNonFunctionalProperties
    name movieRender
    version 0.1
    executable /bin/usr/movieRender
    input /home/grisino/model.mod
    output /home/grisino/movie.avi
    error /home/grisino/error
    working_directory /home/grisino
```

By describing the Semantic Grid services and jobs in a symmetric manner, using terminology provided by ontologies, a semantic matchmaker will be able to determine if jobs and services hosted on the Semantic Grid match in terms of functionality, behavior, resources and VOs requested, respectively provided.

A Grid resource (**resource**) in GSML has the following structure:

```
resource = 'resource' id?
    header* hasDefinition? hasPolicy* belongsToVOs*
```

The **id**, and **header** constructs from a resource specification are defined in the same way as described in WSML. The **hasDefinition** contains a logical definition in terms of concepts and relations from ontologies describing the resource. The **hasPolicy** construct specifies the policy and access rules associated with the resource. The **belongsToVOs** construct is used to specify the VOs the resource belongs to. A **resource** could be further refined as described in the previous section in **computationalResource** and **dataResource**.

A **VO** construct in GSML has the following structure:

```
vo = 'vo' id?
     header* hasMembers* hasDescription?
```

The **id** and **header** constructs in **VO** specification are defined in the same way as described in WSML [5].

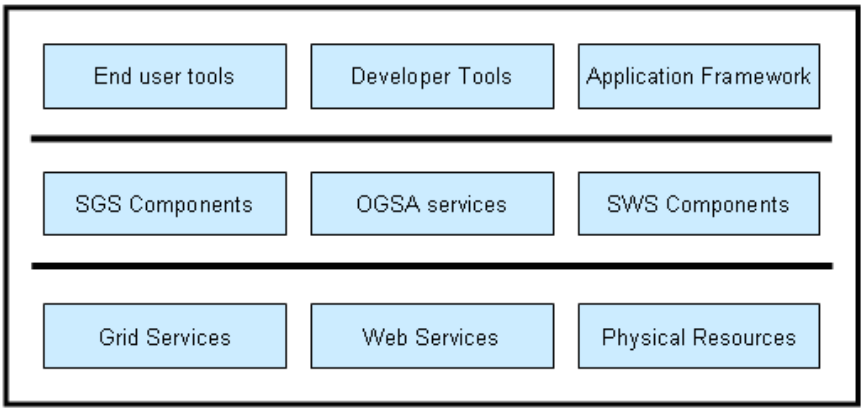

Fig. 3 Towards a Grid Services Execution Environment

3.3 Extensions to the WSMX Execution Environment

This section presents the initial architecture of the Grid Service Execution Environment which will be layered on top of OGSA based Grid toolkits (e.g. Globus Toolkit[8]). The objective of the Grid Service Execution Environment is to process the semantically enabled descriptions of Grid Services and to process the semantic descriptions of Jobs submitted on the Grid. The Grid Service Execution Environment will take care of the service execution management of user-defined applications defined at the semantics layer, which may require specific resource requirements, and imply complex interactions between services. The execution management will extend the conventional execution management of jobs on the Grid, including the execution of Web and Grid Services, with semantically enhanced descriptions of required resources.

The architecture of the proposed framework is based on the Web Services Execution Environment (WSMX) which itself is compliant to the Service Oriented Architecture (SOA) paradigm and consists of a set of loosely coupled collaborating software components. The architecture of Semantic Grid Services Execution Environment is shown in Figure 3. It will act as reference architecture for all the components (existing WSMX and Globus components and the new ones proposed based on GSMO) and integrate them inside one infrastructure. The following newly added components in the Semantic Grid Services Execution Environment are based on GSMO:

- Resource Management which deals with semantic-based resource discovery, advanced reservation, negotiation, deployment and provisioning of computational and storage resources on the Grid

[8] `http://www.globus.org/toolkit/`

- Virtual Organization Manager which deals with creation and management issues of dynamic business oriented Virtual Organizations of services, resources and users in the Grid
- Extended WSML Reasoner for GSML which addresses the knowledge representation and reasoning aspects for discovery, composition and mediation of Grid resources described in GSML.
- Extended Execution Management which covers the implementation of execution semantics of internal Grid Service Execution Environment, and also for external user-defined services and jobs including scheduling, fault-management, and support of the monitoring of execution.

Figure 3 shows the initial architecture of the Grid Service Execution Environment. It shows the initial set of required components which are grouped in three different layers: The upper layer is the problem solving layer in which end user tools, development tools and application frameworks are situated. In this layer already available development tools for WSMX will be extended. Moreover, it includes the Application Framework to support developers in building applications for the Semantic Grid based on the Grid Service Execution Environment. The middle layer (the application layer) includes the newly introduced components based on the extensions of WSMO as GSMO and WSML as GSML, i.e. the VO Manager, Resource Manager, GSML reasoner, the extended execution manager, as well as existing components in the application layer of WSMX such as discovery, selection, composition, negotiation, mediation etc., and the core OGSA services. The bottom (base) layer includes the foundation of the environment such as Grid service descriptions based on Web Services infrastructure and physical resources including computing and storage resources on the Grid.

4 The KCO-SWS Transformer

The main objective of the KCO-SWS transformer (mentioned as T1 in Figure 1) is the realization of the link between knowledge content based systems (resp. the KCCA system) and its Knowledge Content Objects with Semantic Web Service based systems (resp. WSMX). Our intention was to use information stored inside Knowledge Content Objects (KCO) for service discovery and plan execution, e.g. to automatically negotiate or to automatically enrich content and knowledge about that content during the execution of web based workflows like e.g. a document-based business process or workflow to index and enrich documents with additional knowledge. In order to do so, WSMX needs to be able to interpret KCOs and the services offered by KCCA need to be able to communicate with the other services offered by the GRISINO system.

The approach to integrate existing KCO / KCCA technology with the SWS/Grid technologies in the GRISINO system was twofold:

- Metadata descriptions that are contained inside KCOs are translated into WSMO descriptions in order to be useable for service discovery and ranking.

- The KCCA system is wrapped with Web service descriptions that describe its invoke-able functionality. These descriptions are further semantically described.

We started the integration with the investigation of the ontology of plans [8] as well as the "Description and Situation" modules embedded in the OWL DL 397[9] version of foundational ontology DOLCE. Similar work has been reported in [3] or [10]. However, no fully functional translation between DOLCE (resp. its plans extension DDPO[10]) and WSMO has been developed so far for obvious reasons: WSMO in general is a richer knowledge representation language than OWL-DL. The same holds - in principle - for DOLCE, but in order to comply with the restrictions of current semantic web machinery, DDPO has been designed for the restrictions of OWL-DL. Therefore, OWL DL has to be the lowest common denominator for WSMO and DDPO with respect to defining a mapping between the two knowledge representation schemes. This task been done partially already by the WSMO Community [9].

The remaining task was to map the concepts of the DOLCE Design and Plan Ontology (DDPO) and the regarding constructs for the KCO community facet (which are 'static' descriptions of conceptualizations over situations or states) onto WSMO descriptions. In GRISINO we used a subset of concepts defined in DDPO. The services developed within the project are focussed on fairly small parts of document processing, in which goals and plans usually described in KCOs are more generic and most likely closer to a business goal description. Furthermore, they likely include (human) agents in their description, while GRISINO is focussed on automatic manipulation of processes. The concepts 'description', 'situation', 'task', 'role', 'parameter', 'perdurant', 'endurant' and 'region' describe the community facet of the KCO and have been mapped onto WSMO elements as shown in Table 1[11]:

1. DDPO:Goal is not the same as WSMO:Goal because DDPO:Goal is a description of a very general (and semantically open) desire, whereas WSMO:Goal is a specification of a resulting situation for which several plans and executions may exist and where there is a rigid structure containing a domain ontology, a mediator, a capability and an interface. In addition, DDPO:Goal does not play a central role in the actual execution of a DDPO:plan.
   Conclusion: DDPO is epistemologically more open than WSMO. The universe of discourse for WSMO is the world of web services which is linked to the world of "goals" (i.e. desired states of the world) via mediators and whose actual queries (i.e. the goals) are formulated according to the vocabulary of arbitrary ontologies. All we want to ever express in WSMO is a desired state for which it is assumed that it can be reached by the execution of a sequence of web services. For a mapping between DDPO and WSMO it is therefore sufficient to constrain DDPO to the generative power of WSMO. Furthermore, since KCOs only require a very specific set of WSMO descriptions, we can constrain DDPO to those WSMO descriptions which cover KCCA functions for KCOs.

[9] `http://www.loa-cnr.it/ontologies/DLP_397.owl`
[10] `DDPOisaacronymusedforDOLCEDesignandPlanOntology`
[11] The numbers in the listing below refer to the rows in the table.

Table 1 Conceptual mapping of DDPO and WSMO elements

	DDPO Concept	DDPO Features	WSMO	WSMO Features
1	DDPO	GOAL, PLAN (subclass of 'description') SITUATION, TASK, ROLE, PARAMETER, ENDURANT, PERDURANT, REGION	WSMO	GOALS, ONTOLOGIES, WEB SERVICES, MEDIATORS
2	DESCRIPTION and SITUATION		GOAL	constrained-by: nonFunctionalProperty (=labelling) Ontology Capability Mediator Interface
3	PLAN	ROLE, TASK, PARAMETER	Ontology	terminology for specifying the goal
4	SITUATION	(ENDURANT, PERDURANT, REGION)	Capability	nonFunctionalProperty (=labelling) Ontology ooMediator sharedVariables axioms (Precondition) axioms(Assumption) axioms (Post condition) axioms (Effects)

2. DDPO:Plan is the conceptual and descriptive equivalent of WSMO:Goal in its ability to define and reuse concepts that can classify situations, i.e. states in the world. WSMO:Goal, via its capability section describes the specific situation that needs to be fulfilled by a web services capability. The axioms here are used to determine the pre- and/or postconditions for achieving the goal.
3. The concepts role, task and parameter are the descriptive counterparts to classify "objects", "events" and "values" of a given setting. These concepts of DDPO provide the terminology for specifying goals and to describe the domain knowledge.
4. A DDPO:Situation holds the relevant information to describe the pre- and postcondition. A particular situation can be mapped to an axiom used within a capability.

5 Use Case Example

Today information retrieval and text analytics for special interest searches are usually realized as "one of a kind" expert systems. The user input usually is a set of information sources and some definition of the "typical" users point of view, which are fed into a sequence of steps like information acquisition, processing, extraction, annotation, analysis, ... with the goal to deliver rich search and filtering capabilities based on authoritative, domain specific background knowledge. The systems tend to be mostly monolithic, with predefined processes for the specific domains in question.

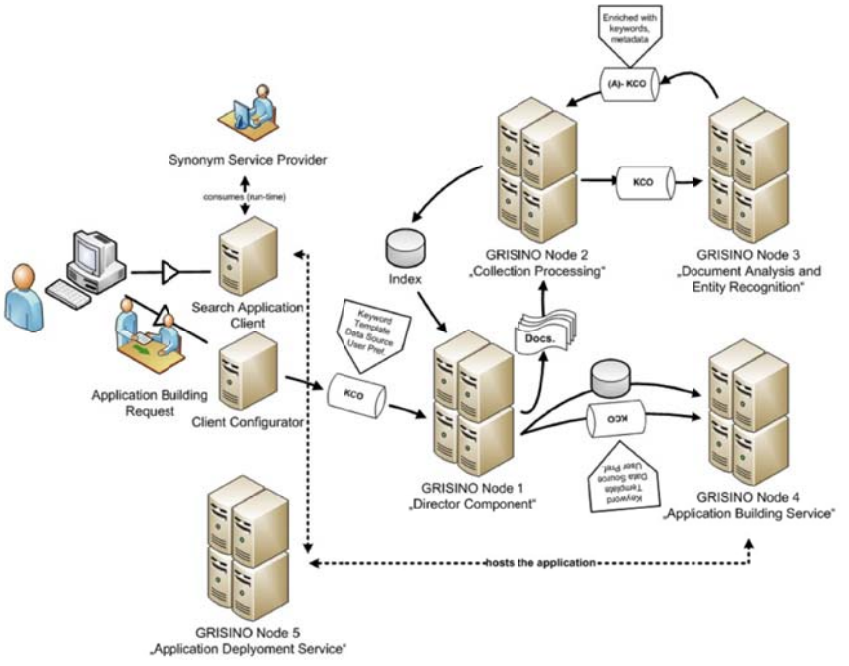

Fig. 4 GRISINO Demonstrator

While such systems deliver a good user experience, they are hard to build, because the one who builds the system has to provide in depth domain knowledge, technological knowledge, many different skills and different resources packaged as a single high quality service.

To meet the demands of the users it is considered very favorable, often even mandatory to be able to personalize the whole processing pipeline according to the needs of an user. Additionally the number of specialized providers for knowledge, content and services is growing, so that as a solution builder you have to consider integration of these providers, as it is hard if not impossible to be expert in all relevant areas, and have all needed information and specialized knowledge readily available, to answer the demands of the user. Thus we will need to build uniform but personalized solutions in the future, which transform a currently static, hardwired process into a dynamic, service oriented process. This dynamic process should be driven by the goals of the user, utilize and bundle services provided by different organizations and gather and combine information and data from different sources. The vision for such a process is to

> *"Transform the goal of the user into the appropriate process, execute it and deliver the solution automatized."*

In order to demonstrate the functionality of the integration and the interoperability between the technologies in the GRISINO test bed, a semantic search application

has been designed that realizes a scalable, flexible and customizable search application generator that enables knowledge-based search in unstructured text. The search applications generated are customized and tailored to specific needs expressed by end users. The search applications include very specialized knowledge about a particular domain (e.g. football in the 19^{th} century), collected from different knowledge bases and consolidated into one index to provide a single point of access.

To achieve this, a number of processing services deployed on the grid, are tied together to selectively collect, index and annotate content according to different knowledge bases and to generate custom search applications according to a users' input. The foundation of the users' input is his/her knowledge background or special interests. In particular the search application generator decomposes the user input (e.g. data sources of interest, specific keywords or entities considered important, etc.), into different sub goals which are used to consider different service providers for enriching the initial input. It queries these services to ask for related terms and entities, as well as authoritative information sources, such as popular websites according to the topic of interest. Using additional services, such as clustering services, the collected documents are then indexed and deployed for the use by the end user.

The goal of the scenario is amongst others to exploit as much of the GRISINO functionality as possible, e.g. to select services based on plans modeled inside KCOs or based on document types, and to parallelise indexing on the Grid. The underlying GRISINO infrastructure enables automation of the whole process of putting together the custom search application by using a number of different services from different service providers and bundling its output into a coherent application that combines knowledge and functionality from different sources. This reflects the particular and very common situation in which both knowledge found in all kinds of knowledge bases and specific skills encapsulated in special technical functionality is not found within one organization or provided by a specific technology provider, but is spread over a greater number of specialized organizations. While the benefit for the user obviously is a richer output informed by knowledge of a number of authoritative service providers, this model allows the commercial aspect of contributing specialized services as input to an open service mix by selling functionality and/or encapsulated knowledge bundled into one coherent service.

6 Conclusions

The GRISINO project brought forward the integration of three distinct technologies as detailed in this chapter. Two major sub-results of GRISINO are a new approach to realize the Semantic Grid which has been the goal of the SWS - Grid transformer and the possibility to use self-descriptions of documents for dynamic SWS discovery in order to automate and execute specific tasks. Regarding the first result, we have followed a new, and previously unexplored approach. More precisely we started from a SWS system (i.e. WSMO/L/X) and added Grid specific features and by that transformed an SWS system into a SWS-Grid system. Furthermore we support the integration of legacy systems such as Globus. The second result, might

be applied in document processing, multimedia content adaptation or other similar scenarios. The semantic search application generator implemented as a proof-of-concept, shows the added value of the GRISINO system both for service providers as well as for end users.

Acknowledgements. The reported work is funded by the Austrian FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) programme under the project GRISINO - Grid semantics and intelligent objects.

References

1. Andreozzi, S., Burke, S., Field, L., Fisher, S., Konya, B., Mambelli, M., Schopf, J., Viljoen, M., Wilson, A.: GLUE schema specification (December 2005)
2. Behrendt, W., Arora, N., Bürger, T., Westenhaler, R.: A Management System for Distributed Knowledge and Content Objects. In: Proc. of AXMEDIS 2006 (2006)
3. Belecheanu, R., et al.: Business Process Ontology Framework; SUPER Deliverable 1.1 (May 2007)
4. Bürger, T.: Putting Intelligence into Documents. In: Proc. of the 1st European Workshop on Semantic Business Process Management (SBPM) held in conjunction with ESWC 2007 (2007)
5. de Bruijn, J., Lausen, H., Krummenacher, R., Polleres, A., Predoiu, L., Kifer, M., Fensel, D.: The Web Service Modeling Language WSML. Technical report, WSML. WSML Final Draft D16.1v0.21 (2005), `http://www.wsmo.org/TR/d16/d16.1/v0.21/`
6. Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications 1(2), 127–160 (1991)
7. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
8. Gangemi, A., Borgo, S., Catenacci, C., Lehmann, J.: Task Taxonomies for Knowledge Content. METOKIS Deliverable D07 (2004), `http://metokis.salzburgresearch.at/files/deliverables/metokis_d07_task_taxonomies_final.pdf`
9. Keller, U., Feier, C., Steinmetz, N., Lausen, H.: Report on reasoning techniques and prototype implementation for the WSML-Core and WSMO-DL languages. RW2 Deliverable (July 2006)
10. Mika, P., Oberle, D., Gangemi, A., Sabou, M.: Foundations for Service Ontologies: Aligning OWL-S to DOLCE. In: Proc. of the 13th Int. World Wide Web Conf (WWW 2004). ACM Press, New York (2004)
11. Roman, D., Lausen, H. (eds.): Web service modeling ontology (WSMO). Working Draft D2v1.2, WSMO (2005), `http://www.wsmo.org/TR/d2/v1.2/`
12. Shafiq, O., Toma, I.: Towards semantically enabled Grid infrastructure. In: Proc. of the 2nd Austria Grid Symposium, Innsbruck, Austria, September 21-23 (2006)
13. Toma, I., Bürger, T., Shafiq, O., Dögl, D., Behrendt, W., Fensel, D.: GRISINO: Combining Semantic Web Services, Intelligent Content Objects and Grid computing. In: Proc. of EScience 2006 (2006)
14. Stärk, R., Schmid, J., Börger, E.: Java and the Java Virtual Machine. Definition, Verification, Validation.: Definition, Verification, Validation. Springer, Berlin (2001)