Talea: An Ontology-based Framework for e-Business Applications Development

Guido Levi[1], Andrea Vagliengo[1], Anna Goy[2]

1 CSP Innovazione nelle ICT s.c.a r.l., Torino, Italy
{levi, vagliengo}@csp.it

[2] Dipartimento di Informatica, Università di Torino, Torino, Italy
{goy}@di.unito.it

Abstract. In this paper we present Talea, a platform aimed at supporting the development of web-based e-business applications. Talea supports a flexible matching between service provision and request. The platform can be easily customized thanks to XML-based communication, Semantic Web technologies, and the exploitation of a generator/performer design pattern which greatly simplifies the task of adding new functionality. Moreover, Talea provides multi-device access to both service providers and final users. An ontological description of the application domain, expressed in RDF/RDFS format, is exploited in order to facilitate the customization and to provide personalized navigation as well as semantic-based search. The ontology-driven personalized navigation is particularly useful for limited display devices (like smartphones or PDAs), since it reduces the amount of information displayed. A first evaluation of the current prototype is planned with a restricted number of users and will be carried on by the Local Tourist Organization.

1 Introduction

In this paper we present Talea [13], a platform aimed at supporting the development of web-based e-business applications. Talea is a software designed and developed within Diadi 2000 [6], a Piedmont Region project co-founded within the European Structural Funds framework. The main goal of Diadi 2000 is to support the technology transfer to small/medium enterprises in the Piedmont Region, in order to increase their competitiveness and technological development.

In order to understand the motivations underlying Talea architecture and functionality, it is necessary to understand the context in which the Diadi 2000 project has been conceived and carried on. Italian small/medium enterprises, especially in the north-west area (Piedmont), are traditionally reluctant to the introduction of technological innovation. Their business strategies are usually based on well-established practices, in which face-to-face interaction plays a major role and new technologies, as well as new business models, are often considered with suspicion and distrust.

Even the Internet has encountered difficulties in being accepted: the majority of small/medium enterprises now have a web site, but a very small number of such sites

are actually integrated with the e-business activities (transaction management, customer care, marketing, etc.)[1].

In this perspective, the Enterprise Application Integration process [1] must start from the very beginning, i.e., from a change in mental attitudes towards innovation. For this reason, the main goal of the Diadi 2000 project is to convince small/medium enterprises that the exploitation of ICT technologies could represent a real added value to their business. In particular, with the Talea software platform, the innovation consists of exploiting Semantic Web technologies, and especially shared and customizable ontologies, to support the customer-supplier interaction, in an Enterprise Application Integration perspective.

The main obstacle to be faced in order to achieve this goal is, again, a change in the mental attitude of small/medium enterprises leaders: the Diadi 2000 project, by means of Talea, aims at showing that a different approach, based on Semantic Web standards coupled with an Object-Orientd architecture, can actually result in an improvement of the customer-supplier interaction and in a more tight integration of enterprise systems. These results, in turn, can increase the effectiveness of business management, and thus the competitive advantage.

The described scenario highlights two important issues concerning Talea:

- It has been designed with the goal of exploiting Sematic Web technologies in an Enterprise Application Integration context, and not in an open environment such as the web.
- The design of Talea had to represent a trade-off between innovation and acceptance: the first prototype, that will be evaluated by small/medium enterprises in Piedmont, had to show effective but simple features, in order to be have a chance to be accepted.

Talea has been designed taking into account this scenario, which imposed two furher important requirements to the framework: easy customization and multi-device access. Easy customization is essential for the instantiation of Talea in different domains, i.e., for its exploitation by a significant number of small/medium enterprises, with a relatively small effort. The multi-device access is a requirement imposed by the current way of doing business, in which both customers and suppliers rely on wireless connections and mobile devices (PDA, smartphones, etc.) to buy/offer services. The platform can be easily customized thanks to XML-based and ontology-based knowledge representation. Moreover, it provides multi-device access both to service providers, that can publish their services, and to final users that can find the required service.

Talea can be viewed as a generic *matchmaker* for e-business (see [14], [15]), since it supports a flexible matching between service provision and request: it enables service providers to advertise theirs products and services, and customers to search for

[1] For instance, in 2003 almost all Italian PMI (small/medium enterprises) had an Internet connection, two out of three of them had a static web site, but only 5,5% exploited the Internet to do business on-line – see MATE-Commercio Elettronico Italia, “Osservatorio PMI I semestre 2003”, www.commercenet.it.

resources they are looking for[2]. In this perspective, Talea differs from other approaches, such as semantic Information Retrieval systems (e.g., [2], [7], [9]) or semantic query languages (e.g. [3], [5]) that have different goals, since they aim at performing search in open environments, while Talea, as we already mentioned, is conceived to be exploited within a group of known enterprises.

The Talea first prototype represents a first step in the direction of bringing ICT, and in particular the Semantic Web, to small/medium enterprises, and of course it can be enhanced in different directions. In order to provide a first trial, the Talea platform was tested as a prototype service supporting the collaboration process among minor actors, participating in the local tourist sector, and the Local Tourist Organization Agency. More specifically, the service was tested in the mountain area that will host in Piedmont the next Winter Olympic Games in February 2006, with the support of the Local Tourist Agency MontagneDoc.

After the testing phase, that took place in the period June-September 2005, Talea has been released in compliance with the general principles of Open Source and its licences, with the aim of encouraging its use to implement and provide high quality personalized services within the supply chain. Talea has been made available for small and medium enterprises with the aim of increasing their local competitiveness, introducing ICT within cooperation processes and transferring towards local companies, belonging to the computer science and ICT sectors, a new model of re-use and customization of basic software components. In line with this aim, a first group of local companies has been selected in order to concretely apply Talea technology in their real business, building up new services to be launched in the market. Those "early adopters" will give us the opportunity of evaluating Talea on real business domains and will provide us with an important feedback, that will be useful to improve the system.

2 The Talea Framework

The Talea framework is used in order to implement applications for different business domains; this implementation requires the customization of the framework to the specific domain. In the following we will refer to the person (or the team) who performs such a customization as the *customizer*. The result of the customization is an application that can be exploited by the final user (from now on, simply, *user*) to provide and consume *resources* (products or services). Talea users can be both providers and consumers[3]. Provider users are small/medium enterprises, while customer user can be individuals or agencies. For instance, in the first prototype, the domain is mountain tourism; customer users are local travel agencies and provider users are mountain guides, innkeepers and so on.

---

[2] At the moment there is no automatic negotiation: costumers can reserve products/services and Talea notifies the corresponding providers about such a reservation (by e-mail, sms, or fax).

[3] Sometimes it is the case that a single entity plays a double role, acting both as provider and consumer.

The design of the Talea framework has been focused on the *customizer*, in order to provide him with:

- an infrastructure that supports the matching between provider and consumer (Talea Backend);
- a customization mechanism that supports the description of the particular application domain (Talea Ontology extension)
- a programming pattern that support the easy implementation of new features, to extend functionality (Generator/Performer Pattern)

By exploiting these tools, the framework aims at supporting customization at three different levels:

- the presentation layer;
- the business logic;
- the application domain.

The customization of the presentation layer requires the design of the final user interface supporting the interaction with the Talea Backend: such a user interface is handled by the Frontend and provides multi-device access (see Sect. 4).

The customization of the business logic involves the implementation of new features to extend the functionality of the Talea Backend (see Sect. 4).

The customization of the knowledge about the domain is achieved through the extension of the the Talea Ontology (see Sect. 3).

3 The Talea Ontology

Within the Talea framework, the ontology has a major role in the customization task: the *customizer* has to describe the application domain by providing an ontological description that includes user and service categories as well as the relations between them: in this way she provides the knowledge about the particular business domain, exploited by Talea to perform the semantic search of available resources and the personalized navigation, tailored to the user role.

The ontology included in the Talea framework (henceforth Talea Ontology) is expressed in RDF/RDFS format [11] and represents the top-level classes. The customization process consists in the extension of this top-level; the exploitation of a standard format such ad RDF/RDFS guarantees the possibility of exploiting standard tools, such as Protégé [10], to extend the ontology.

Talea Ontology describes the semantics of the matchmaking process since it represents the relations between users and services, i.e. the fact that small/medium enterprises provide and/or consume services. Talea Ontology defines two classes, together with theirs properties, and the relations between them. In particular:

- The *User* class describes a generic Talea final user;
- The *Service* class describes a generic Talea service;
- The *provide* and *consume* relations represent the link between users and services.

The *User* and *Service* classes in the Talea Ontology are generic concepts, because theirs features depends on the particular application domain.

Using the Protégé editor [10], the *customizer* defines *User* and *Service* subclasses; she also defines the relations among such subclasses by specifying which *Service* subclasses are provided and consumed by each user category (subclass of *User*). For example, in the first prototype, instantiated on the tourist domain, examples of subclasses of *Service* are *Course*, *Excursion*, *Lodging*; *MountainCourse* is a subclass of *Course* and *MountainExcursion* is a subclass of *Excursion*; examples of subclasses of *User* are *MountainGuide* and *TravelAgency*; instances of *TravelAgency* can *consume* all kind of services, while *MountainGuide* can only *provide* reasources that are instances of *MountainCourse* and *MountainExcursion* (and not, for instance, of *Lodging*).

The application resulting from the customization of the Talea framework supports the users to provide, reserve and consume instances of the *Service* classes (i.e., resources). Resources are stored in a database, in order to manage the booking process, while instances of the *User* class (registered users) are stored in an Authentication Server in order to manage the authentication process. The structure of the resources database can be generated automatically starting from the ontology. In particular, at customization time, the *customizer* can select the level of the ontology classes that are converted into database tables. For instance, she can select the classes directly under *Service* (e.g., *Course*, *Excursion*, *Lodging*), or she can decide to have a larger number of smaller tables by selecting a lower level (e.g., *HikingCourse*, *RaftingCourse*, which can be subclasses of *MountainCourse*): an ontology class corresponding to a database table is called a *table class*: only *table classes* (and their subclasses) can have instances stored in the database.

At run-time, the link between each resource and its semantic description in the ontology is represented by a field (*instance_of*) that specifies the class the resource belongs to. This mechanisms enables the Talea semantic engine to support the semantic search of services: in order to find a particular resource, the name of the database table where it is stored is not required, since the name of the class it belongs to is enough for the semantic engine to find the corresponding table and perform the SQL query.

In the Authentication Server the information about each user includes the ontology classes she belongs to; if a user belongs to several classes at the same time (because she is, for intance, both a mountain guide and an innkeeper), she will be able to provide/consume the union of the resources that instances of those user classes are allowed to provide/consume.

This mechanism supports a personalized interaction with the application and the ontology defines the underlying navigation structure for both consumers and providers. Let's illustrate this mechanism with an example. Let's imagine that Gio, a mountain guide, connects to the system through his mobile phone in order to publish his availability for a rafting course on August 13th. After login, the system recognizes him by retrieving information from the Authentication Server: being a mountain guide, he does not consume services but provides them and, in particular, knowing the class he belongs to, the system can retrieve thye type of services he can offer (e.g. *MountainCourses, MountainExcursions*). As a result, Gio will be presented a list of

such classes, among which he can select the one corresponding to the resource he wants to insert; the selection action corresponds to ontology browsing (e.g., he selects *MountainCourse*, and then *RaftingCourse*). When Gio reaches a *table class* (or a subclass of a *table class*), he is enabled to insert a new instance of such a class or to select a more specific class going on in browsing the ontology. In this example we suppose that the *MountainCourse* is the *table class*: Gio can insert a generic mountain course or decide to insert a more specific course, e.g. a *RaftingCourse* (subclass of *MountainCourse*).

The travel agency employee can search for a *RaftingCourse* (or a more generic *MountainCourse*) available on August 13th; she will be presented a list of rafting courses, coupled with their providers; from this list she can choose the course provided by Gio and reserve it.

The domain ontology is the key factor for customization and thus its careful design is crucial for the successful development of new application based on the Talea framework. This can be viewed as a limitation, since it assigns to the *customizer* a great responsibility. However, it is worth mentioning, again, the particular context in which Talea has been conceived: the local small/medium enterprises business domains can usually be modeled by means of a small number of classes, resulting in ontologies with a few levels and a small branching factor.

The knowledge represented by the Talea Ontology extended for specific domains is exploited by the Semantic Engine in order to support personalized navigation and semantic search (see Sect. 5).

4 System Architecture

Talea architecture (see Fig.1) was designed in order to facilitate domain-specific customization. This principle suggested us a number of criteria for the design of the system architecture.

Backend and Fronted. First of all, the platform architecture is composed by two main separated parts, i.e. a Backend, containing the business logic, and a Frontend, managing the presentation layer.

XML messaging. The second important criterion that inspired the design of Talea is the principle of a typical service-oriented application: XML messages are used to communicate data among system modules.

The separation in two independent parts (Backend and Frontend), communicating through XML, was introduced to enable a flexible implementation of multi-channel and multi-device access to the system. We considered two main categories of client: browser enabled clients (Pc, PDA, XHTML-capable mobile phones) and browser-less clients (stand-alone applications, MHP clients, ecc.). While the first can exploit XHTML markup to populate interfaces, the latter have no browser capabilites and need to extract content directly from XML objects. This is the reason why the Backend core is isolated and the Frontend can act as a "meta-client", manipulating XML content through XSL transformations to generate the final XHTML markup.

The customization of the presentation layer (see Sect. 2) requires the *customizer* to implement a new Frontend or to modify a provided one, i.e. the XHTML Frontend by

rewriting XSL transformations. The customization of the Frontend is important for the final application, because it defines the layout in which the user interaction will take place.

Modules and design pattern. Finally, the system was designed with particular attention to make the implementation of the high-level operation as modular as possible. There is a class for each high-level operation, in order to isolate as much as possible its management. In particular, the platform modules are based on the *generator/performer* design pattern, which greatly simplifies the task of adding new features (to extend functionality). Each Talea core functionality corresponds to an XML message format, devoted to a specific request/response type; each request type is handled by a specific *Performer* module. The Performer that is in charge of managing the user request is dynamically selected at run-time, on the basis of the request type.

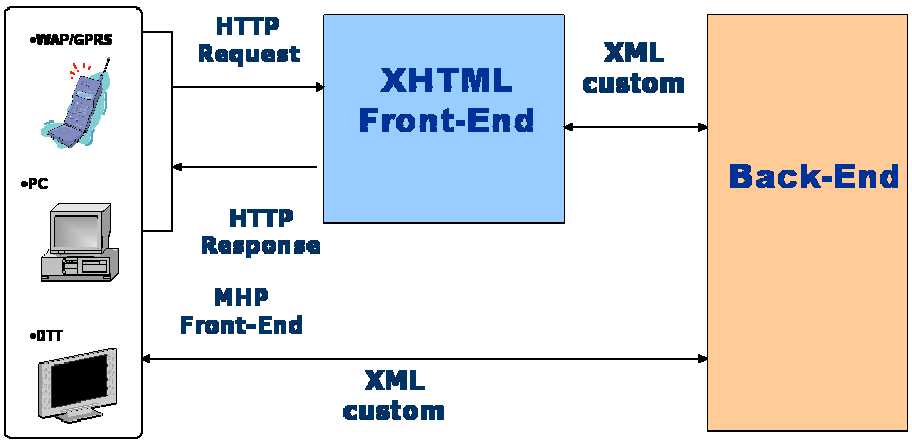

Fig. 1. The architecture of the Talea framework

The Frontend modules, that manage HTTP requests and generate the corresponding XML message, are based on the same pattern. In particular, for each request type (encoded in a parameter within the HTTP request), a *Generator* module creates the XML message (based on the HTTP request attributes), to be sent to the Backend.

The interaction between Frontend and Backend follows the request/response protocol:

- The Frontend sends an XML message to the Backend;
- The Backend forwards the message to the specific Performer;
- The Performer handles the request and sends the Frontend a response message.

The creation of a new Performer module is the most challenging customization task, since it enables the *customizer* to add a new feature to the system. For instance, in order to add new advanced search capabilities, the creation of a new Performer module that encapsulates the new search strategy should be required. However, the addition of a new Performer is not the only way to customize the business logic in Talea.

In fact, the framework supports the modelling of domain-spcific business logics, by customizing the basic Performer modules supplied with the framework extending their functionality when needed. To this purpose, the framework includes two public interfaces, *PreAction* and *PostAction*, that can be used to customize the Performer's pre-conditional (and post-conditional) logic.

The most notable aspect of Talea framework, from the architectural point of view, is that core modules do not make any assumption about domain-specific semantics: the business logic embedded within system modules is totally independent with respect to the specific type of resources or users as well as to their semantics. This guarantees the generality of the approach.

5 The Semantic Engine

Talea Semantic Engine is the Backend module that performs semantic search and personalized navigation exploiting the ontology. The Semantic Engine is composed by different layers (see Fig.2). In order to access the knowledge described by the ontology (in RDFS format), the Semantic Engine use SeRQL, the RDF query language supported by Sesame (see [12]). In particular, Sesame provides a set of API for RDF Schema querying and inferencing and the Semantic Engine exploits them for the implementation of the Talea Semantic API.

The goal of the Talea Semantic API implementation is to facilitate the business logic customization by providing a set of parametric ontological queries that can be used by each Perfomer, thus avoiding the direct use of SeRQL. In this way a Perfomer module can access the semantics of resource categories, i.e. its properties, its sublcasses, its corresponding *table class*.

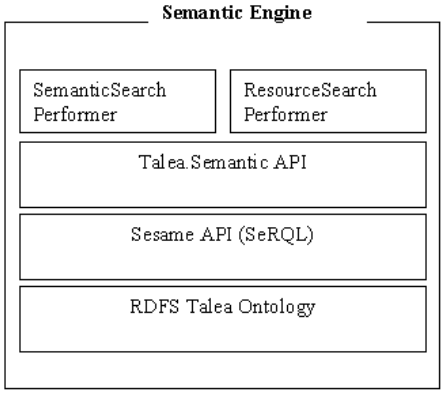

Fig. 2. Talea Semantic Engine

The two performers provided by the Talea framework use the Talea Semantic API: in particular the Semantic Search Performer (see Sect. 5.1) supports personalized

navigation and the Resource Search Performer (see Sect. 5.2) supports the search for specific resources.

5.1 Semantic Search Performer and Personalized Navigation

As we have seen in the example described in Sect. 3, Talea provides an ontology-driven personalized navigation, which is particularly useful for devices with limited display capabilities (like smartphones or PDAs, extensively used by the users of the current prototype). The underlying ontology makes the navigation more user friendly[4]. The ontology-driven personalized navigation, in fact, can reduce the amount of information that the user has to provide to perform her search, because it enables her to operate only on those resources specified by the categories she belongs to. Moreover, by ontology browsing, the user can operate at different detail levels (i.e she can insert/search a mountain course or a more specific course like a rafting course) and the number of clicks required during the browsing only depends on the user task.

Let's now illustrate how ontology browsing supports resource insertion and search. When the user browse the ontology, the Semantic Search Performer comes into play: each user browsing action triggers a semantic search. The request corresponding to a user browsing action specifies the categories the user belongs to, and the clicked item; its format is:

```
<what> clicked_item </what>
<who> user_class1 </who>
<who> user_class2 </who>
```

The response depends on the *clicked_item* position in the ontology:

- At the very beginning the user is asked to choose if she wants to insert (provide) a resource, or if she wants to look for (consume) a resource. In this interaction step, the user action (*clicked_item*) corresponds to the selection of a relation (*provide* or *consume*), thus the response is the list of service categories provided or consumed by each *user_class*.
- While browsing the ontology, usually the user clicks on service categories; in these cases, *clicked_item* is a class, thus the response is the list of its direct sublcasses.
- Finally, *clicked_item* can be a leaf (a class with no subclasses); in this case the response is the list of its properties and the user can express requirements about the values of such properties.

In this way the Semantic Search Performer supports the navigation by looking for the "meaning" of the clicked item and providing the user with the next navigation step, on the basis of such a meaning.

In the case the user is a provider and she wants to insert a new resource, when the response list contains a *table class* (or a subclass of a *table class*), the next step depends on the user: she can insert a new resource (an instance of such a *table class*, for

[4] This is true especially if the ontology is not too complex. But as we already mentioned, the particular context in which Talea is exploited should guarantee such a requirement.

example a mountain course) or she can go on browsing, in order to get to a more specific category (for example, if she wants to insert a rafting course).

In the case the user is a consumer and looks for resources, at each step she can stop browsing the ontology in order to search for the resources available for the current class. Available resources correspond to instances stored into the database. For example, she can stop at *MountainCourse* and search for all available mountain courses or she can browse one step more and stop at *RaftingCourse*, if she is looking for a more specific kind of course). Actually, the current user interface also enables her to search for generic mountain courses and then specify the subclass (e.g. *RaftingCourse*) by filling in a form. This mechanism allows the user to avoid browsing the whole ontology and represents the best solution when the number of direct sublclasses is to high for visualization as a list.

Finally, it is worth mentioning that the search, differently from the insertion, is not constrained by the definition of *table classes*: as we will see in Sect. 5.2, the user can search for instances of a class that is more general than a *table class* (this is not possible for the insertion, where only instances of *table classes* can be inserted into the database).

5.2 Resource Search and Insertion Performers

The Resource Search Performer is the module that performs the instance search into the database. The request corresponding to a resource search is translated by the Performer in a SQL query to the database; in order to make this translation easier, the format of the request is SQL-like (and it also allows joins between categories of services and users):

```
<select> Property1, Property2 </select>
<FROM> class </FROM>
<where> list_of_conditions </where>
```

The resource insertion request is similar to the resource search in being close to an SQL insertion query.

When a user performs a resource search, i.e. she looks for instances of a particular class, the salient properties of such instances are returned by the Semantic Search Performer; in this way the user can set a list of conditions on the values of those properties (e.g., type of resource, date and time, and so on), by filling in a form. The resource search response is a list of instances that match the conditions list.

The resource search/insertion functionality can be used independently from the personalized navigation and also in its place, when the ontology structure is too complex to support a user friendly navigation.

6 Implementation Details

Talea is implemented in Java and exploits the Java Servlet tecnology, running on an Apache Tomcat Web Server. The resources databases is implemented in MySQL and

exploits the JDBC Connector, while the users data are stored in an LDAP Server (OpenLDAP). The genaration of user interfaces for browser-enabled devices are based on XML/XSLT tecnologies, while a MHP client has been developed for the DTT user interface. Apache Cocoon has been used as server-side web development tool. Moreover, the system relies on the already mentioned Protégé [10] and Sesame [12], the former as ontology editor and the latter to support the semantic engine. Eclipse has been used as development environment.

7 Conclusions and Future Work

In this paper we have introduced the first release of the Talea system that represents a starting point for the implementation of an ontology-based framework supporting e-business applications development.

Future works will follow two directions. First, we are going to support the selected small/medium enterprises during their customization process. We expect that this process will suggest us a set of improvements in the direction of a more general and powerful framework, usable in real application domains. In particular, an interesting feedback is represented by the problems that small/medium enterprises will face in the the integration with legacy applications (like enterprise databases and ERP).

The second direction is the implementation of a new release involving a Web Service architecture and semantic orchestration: the idea is to wrap the Performer modules in Web Services interfaces and to exploit semantic orchestration in order to provide macro-functionality [4], [8].

We are also planning to improve the semantic aspect in resource search by introducing a semantic repository for instances storage. This solution would support resource semantic search (no more SQL-like), but requires the *customizer* to create a gateway module to manage interaction with existing DBMS.

Finally, we would like to study an intelligent brokering solution, that could be based on intelligent agent negotiation.

References

1. Alonso, G., Casati F., Kuno, H., and Machiraju V.: Web Services - Concepts, architectures and applications. Springer, Berlin Heidelberg New York (2004)
2. Bonino, D., Bosca A., Corno F., Farinetti L., and Pescarmona, F.: H-DOSE: an Holistic Distributed Open Semantic Elaboration Platform. First Italian Semantic Web Workshop (SWAP2004), Ancona, Italy (2004)
3. Bouquet, P., Kuper G., Scoz M., and Zanobini, S.: Querying the Semantic Web: A New Approach. First Italian Semantic Web Workshop (SWAP2004), Ancona, Italy (2004)
4. Bruijn, J., Fensel, D., Keller, U., and Lara R.: Using the Web Services Modelling Ontology to Enable Semantic eBusiness. Communications of the ACM (CACM) - Special Issue on Semantic eBusiness (2005)

5. Catarci, T., Di Mascio, T., Franconi, E., Santucci, G., and Tessaris, S.: An Ontology Based Visual Tool for Query Formulation Support. Proc. of the 16th European Conference on Artificial Intelligence (ECAI2004), Valencia, Spain (2004)
6. The Diadi 2000 Project: http://www.diadi.it/
7. Maynard, D., Yankova, M., Aswani, N., and Cunningham, H.: Automatic Creation and Monitoring of Semantic Metadata in a Dynamic Knowledge Portal. Proc. of the 12th International Conference on Artificial Intelligence: Methodology, Systems, Applications – Semantic Web Challenges (AIMSA2004), Varna, Bulgaria (2004)
8. McIlraith, S. and Martin, D.: Bringing Semantics to Web Services. IEEE Intelligent Systems, 18(1) (2003)
9. The Mondeca Semantic Web Portal: http://www.mondeca.com/
10. Protégé: An Ontology Editor and Knowledge-Base Framework: http://protege.stanford.edu/
11. Resource Description Framework (RDF): http://www.w3.org/RDF/
12. Sesame: an Open Source RDF Database with Support for RDF Schema Inferencing and Querying: http://www.openrdf.org
13. The Talea System: http://talea.csp.it/
14. Trastour, D., Bartolini, C., and Gonzales-Castillo, J.: A Semantic Web Approach to Service Description of Matchmaking of Services. Proc. of the International Semantic Web Working Symposium (SWWS2001), Stanford, California (2001)
15. Trastour, D., Bartolini, C., and Preist, C.: Semantic Web Support for the Business-to-Business E-Commerce Pre-Contractual Lifecycle. Computer Networks: The International Journal of Computer and Telecommunications Networking, 42(5), Special Issue on The Semantic Web: an Evolution for a Revolution, North Holland/Elsevier (2003)