4th WSEAS Int. Conf. on COMPUTATIONAL INTELLIGENCE, MAN-MACHINE SYSTEMS and CYBERNETICS Miami, Florida, USA, November 17-19, 2005 (pp115-122)

Auto-generation of Multi-fielded Domain-Specific Search Forms

ROBERT STEELE
Faculty of Information Technology
University of Technology, Sydney
PO Box 123, Broadway, NSW 2007
AUSTRALIA

Abstract: - Custom made domain-specific query interfaces can provide substantially greater search power than single-field keyword search interfaces. However while there are tools available to automatically build single-field search systems for a given repository of documents there is currently no way to automatically generate multi-fielded domain-specific query interfaces and systems. In this paper we propose a technique that given an object-oriented conceptual model of data as input can automatically generate a complex domain-specific navigable query interface and search system for that data. Such techniques can have significant impact in making it easier to build e-Commerce Web sites or in rapidly developing query interfaces for integrating data systems as is typically required by the dynamically changing business partners in Extended Enterprise scenarios.

Key-words: - e-Commerce, Object-oriented conceptual models, UML, RDF, Extended Enterprise

1. Introduction

It is very important to be able to search across Web data and enterprise data. Search engines provide for searching of the Web and enterprise search systems provide for searching across enterprise data. Enterprise search systems [20] are able to automatically provide single-field or keyword-based search systems but cannot automatically provide complex domain-specific query interfaces.

Enterprise data (and Web data) tends to be highly heterogeneous, involving multiple differing storage formats, and is distributed in the sense that it is typically not stored in a single monolithic repository, particularly in extended enterprise scenarios. With the wide acceptance of object oriented modeling more and more systems are being built using OO model syntax [8]. The Unified Modeling Language (UML) [14], a representation for such models, has become widely used and supported by software tools, as such object oriented conceptual models represent an already widely deployed model that can provide a unifying model across data that is both heterogeneous and distributed. While such models can be utilized as a unifying model in this way they also have a particularly natural fit with some data storage systems. In *particular the data in OO databases* can be most naturally modeled in this way.

In this paper we present a technique for automatically creating a powerful multi-fielded query interface and data search system solely from an OO conceptual model of the data. This technique has the benefits that it does not require domain-specific manual expertise and effort to build the search system. In extended enterprise scenarios where enterprises relatively dynamically adjust collaboration partners, such easy to establish powerful query systems are very important. Via RDF it supports leaving heterogeneously stored data in its native formats while adding semantic markup, it allows for distributed storage of the data, it provides a novel and powerful search interface for data and allows dynamic generation of query interfaces. The technique has a level of broad applicability given that it can be applied to any collection of heterogeneous data that has been modeled using a OO conceptual model.

2. Background

Object-oriented conceptual models provide a way to model the structural aspects of objects [6]. The kinds of relationships supported are generalization, association and aggregation. As OO modeling is widely adopted for building software systems, OO modeling of data lends itself to natural integration of data sources with software systems. OO conceptual

models are adopted here for these reasons and due to its well developed tool support.

EXtensible Markup Language (XML) [1] and XML Schema [4] allow flexibility in defining the structure of documents and services [16]. Resource Description Framework (RDF) [2] and Resource Description Framework Schema (RDF(S)) [3] are XML-based technologies that provide a way to attach semantic information to resources. Resources can be words, parts of documents, Web pages, whole Web sites or other data units. The attaching of semantic information requires the writing of RDF statements. These take a subject-property-object form. For example <http://www-staff.it.uts.edu.au/~rsteele, created by, Robert>. RDF(S) introduces the concept of class, or resource "type", and defines some inbuilt semantic relationship primitives (inbuilt properties) such as subClassOf, subPropertyOf, domain and range. RDF(S) hence enables the defining of classes, relationships between classes and the definition of new relationships (properties).

This paper brings together OO conceptual models, RDF and auto-generation of GUIs (in particular WWW hypermedia query interface). This topic exploits the relationship between meta-data (or semantic information about data) and search.

Substantial work has been done to explore the implications of RDF and RDF-based systems for knowledge management and the integration of heterogeneous data sources. Projects in this area include On2broker [9] and Hera [18]. Hera makes use of RDF with an aim to provide a semantic interface for integration of heterogeneous data sources. While these projects explore many ways of utilizing semantic markup, they do not explore the relationship between object-oriented conceptual models, RDF and user interfaces and how these can be linked to provide an automated system development technique.

Recent research, in some cases as part of these above projects, has also explored how to provide a user interface to RDF-based repositories [11, 12, 13, 19]. These systems typically propose graphically complex interfaces that are also typically not browser-based. The emphasis is on navigation and browsing interfaces to repositories not on the auto-generation of a data query interface. In addition querying in these projects makes use of complex query languages. The technique proposed in this paper aims to produce a simple but powerful query interface for the data that is hypertext-based and does not require the use of a query language.

It has been observed that the adoption of semantic markup technologies such as RDF has faced a "chicken and egg" problem [5]. That is, that individuals and enterprises have generally not provided such semantic mark-up of their data as there have not been compelling applications to make use of it, and there have not been compelling applications that make use of such mark-up as semantic mark-up is not yet widespread. The technique proposed in this paper would provide one possible immediate and tangible benefit from adding RDF markup – an auto-generated multi-field query interface.

3. Auto-generation of Query Interfaces

The technique for automatically generating query interfaces and search systems from an object-oriented conceptual model has the following steps:

1. Develop an OO conceptual model of the data
2. Transform to RDF Schema representation of the OO conceptual model
3. Attach the RDF markup to the data informed by the conceptual model
4. Auto-generate the query interface (HTML, WML) from RDF representation of model

3.1 Develop Object-Oriented Conceptual Model

The object oriented conceptual model or the underlying data may have been produced first. Ideally a model for the data will have been first developed and then the data stored in terms of this model. In this case the data will already be labeled in terms of the classes and relationships in the conceptual model. An object-oriented database would clearly conform directly to this model.

In other cases the data will exist prior to object – oriented modeling. This will be the case with large amounts of legacy data. In this case the creation of the object-oriented conceptual model will proceed from examination of the existing data. Some domains will afford a more natural fit between data and this modeling approach. Examples of well suited domains might be electronic health records, business documents/ records and other application domains that have a fielded format or obvious specialization/ generalization relationships.

A common and convenient representation format for the object-oriented conceptual model is UML. A UML class diagram for instance can be used to represent the data model. As our extended example in this paper, consider the scenario of a roaming mobile device user, accessing a location-based service system, who wishes to search for products and services of businesses in the local vicinity. In this example the entities involved can be represented using UML. A small fragment of the UML-encoded conceptual model for this scenario is shown in Figure 1.

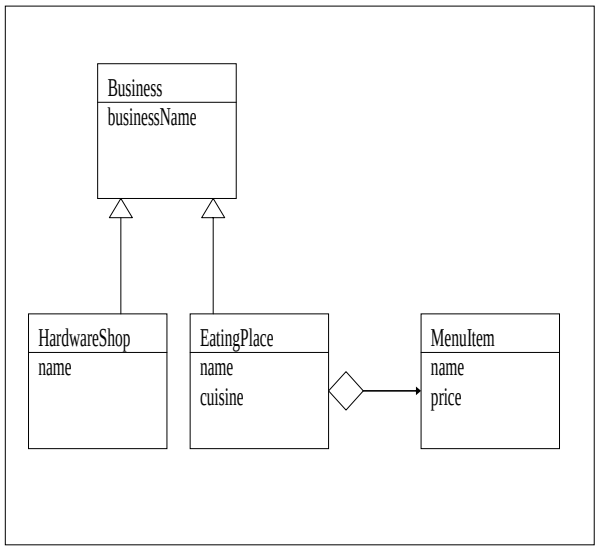

Fig. 1: UML class diagram of business and restaurant related classes

For example the model to represent this scenario might include classes such as Business, HardwareShop and EatingPlace. The Business class is a generalization of the HardwareShop and EatingPlace classes (amongst others). Each Business has a businessName and HardwareShops and EatingPlaces also have names. EatingPlaces also have an attribute that is "type of cuisine". The EatingPlace class also has an aggregation relationship with MenuItem. That is each EatingPlace can have one or more MenuItems. These MenuItems also have some complex structure: a name and a price.

3.2 Transformation to an RDF Schema Representation of the Conceptual Model

We transform the object-oriented conceptual model into an RDF Schema representation for two reasons:

- We now have an XML-based representation of the model that can be used as an input for further automated transformation and generation
- RDF is an XML-based format intended and well-suited to labeling resources/data in terms of semantic relationships and it can be used to markup the data that is to be searched. Once it is used to markup the data it will enable the applying of XML query techniques such as XPath queries that can search in terms of named concepts of the conceptual model

UML class diagrams can be automatically transformed into RDF Schema. Existing related work by Feng, Chang and Dillon [7, 8] provides transformations from object oriented conceptual models to XML Schema. Of particular relevance to this paper is that an RDF Schema can then be used to automatically generate a query interface to the data that has been modeled.

An example of the RDF Schema that would be generated for our scenario of a system to access local mobile services is shown in Figure 2. The generated RDF Schema can then be used to markup documents. In our example this would involve the marking-up of data about the product and service offerings of local businesses.

In this transformation each class in the UML diagram is mapped to a corresponding RDF class. The RDF subClassOf property is used to represent the generalization/specialization relationships from the OO conceptual model. In our example this means that the relationship between the Business class and the HardwareShop and EatingPlace classes is represented in the RDF representation via the subClassOf RDF(S) property.

Where ever a class has an attribute or a class in an aggregation relationship a corresponding RDF property is defined. *However where ever a class* ***attribute****(e.g. name, cuisine) is present in the OO conceptual model it is mapped to a* ***property with a range of type RDF Literal*** *in the RDF representation of the model.* Where there is an aggregation relationship in the conceptual model to another class, it is mapped to a property in the RDF representation that has as a *range* the class that is to be aggregated.

```
<rdf:RDF
   xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-
syntax-ns#"
   xmlns:rdfs= "http://www.w3.org/2000/01/rdf-
schema#"

<rdfs:Class rdf:ID="Business"/>

<rdfs:Class rdf:ID=”EatingPlace”>
   <rdfs:subClassOf rdf:resource="#Business” />
 </rdfs:Class>

<rdfs:Class rdf:ID=”HardwareShop”>
   <rdfs:subClassOf rdf:resource="#Business” />
 </rdfs:Class>

<rdfs:Class rdf:ID=”MenuItem”/>

<rdf:Property rdf:ID="hasMenuItem">
  <rdfs:range rdf:resource="#MenuItem " />
  <rdfs:domain rdf:resource="#EatingPlace" />
</rdf:Property>

 <rdf:Property rdf:ID="name">
  <rdfs:range rdf:resource =
"http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="#MenuItem" />
  <rdfs:domain rdf:resource="#EatingPlace" />
  <rdfs:domain rdf:resource="#HardwareShop" />
  </rdf:Property>

<rdf:Property rdf:ID="businessName">
  <rdfs:range rdf:resource =
"http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="#Business" />
</rdf:Property>

 <rdf:Property rdf:ID="cuisine">
  <rdfs:range rdf:resource =
"http://www.w3.org/2000/01/rdf-schema#Literal" />
  <rdfs:domain rdf:resource="#EatingPlace" />
  </rdf:Property>
</rdf:RDF>
```

Fig. 2: Object-oriented conceptual model represented using RDF Schema

3.3 Attaching RDF markup to the Data

Providing “semantic markup” such as RDF markup is typically seen as something that needs to be manually implemented. However in the technique proposed in this paper this markup activity may in a number of cases be wholly or partially automatable. This would be particularly true where the search system to be developed is to apply to search of data within a single enterprise.

Where search is to occur within an enterprise there is a greater level of common authority over the data and hence the ability to enforce a model across all of the data. In this scenario in some cases a conceptual model would be developed first and then data stored and represented consistently with this model from the beginning. If this were the case it would be possible to automate the process of marking up the data with RDF. Each piece of data corresponding to a particular class or property within the conceptual model will now be labeled with its RDF equivalent as explained in Section 3.2. This might for instance involve having the XML/RDF that asserts this markup being located in a separate file and this file providing a link property that specifies the URL where this piece of data is actually located.

```
<rdf:RDF
xmlns:rdf = "http://www.w3.org/TR/WD-rdf-
syntax#"
xmlns:mn = http://www.it.uts.edu.au/menu#

<rdf:Description id="menuItem1">
  <rdf:type=mn:MenuItem>
  <mn:name> Chilli Basil Chicken </mn:name>
  <mn:price> 5.80 </mn:price>
</rdf:Description>

<rdf:Description id="menuItem2">
  <rdf:type=mn:MenuItem>
  <mn:name> Green Curry Chicken </mn:name>
  <mn:price> 5.80 </mn:price>
</rdf:Description>
</rdf:RDF>
```

Fig. 3: An example RDF snippet

There would also be no need for manual RDF markup where the data source is an OO database. As such the technique proposed in this paper provides a way to auto-generate a powerful query interface to OO databases without manual intervention.

In other cases where search occurs across multiple enterprises for example it will be more difficult to enforce a common conceptual model preceding data storage. The running example used in this paper is closer to this scenario. In this case it is likely that some amount of manual effort will be

required to provide the RDF markup. For example, in our scenario each business would need to provide an RDF statement that their business is for example of class Business, HardwareShop or EatingPlace or other appropriate category.

See Figure 3 for an example of the RDF markup. Two MenuItems for one particular EatingPlace are marked up. This has involved specifying the properties for each that have MenuItem class as the domain. In reality there would typically be many more MenuItems for each EatingPlace. In addition there would be many EatingPlaces, HardwareShops and other Businesses. If a more example complete conceptual model was given for our example scenario it could also include classes to represent products and services in other businesses.

An advantage of this approach is that existing files providing such information as the menu items of a restaurant or the inventory of a hardware shop can be marked-up just by adding the RDF tags. This of course could be done with graphical-based tools. Or for example information about menus or a business can remain in separate files that involve different formats (different businesses could use different data formats also). In this case the RDF (XML) code showing the properties provided by the conceptual model would still have to be provided in the case of all businesses. This RDF/XML code would include URLs to the existing data files.

3.4 Automatic Generation of Query Interfaces

Given the RDF Schema representation of the conceptual model a multi-fielded query interface can be automatically generated. The basic structure of the interface will be driven by the class hierarchy of the model. This will have a tree-like or multiple tree-like structure. While RDF supports a directed graph-like structure in general, OO models have a more tree-like structure.

See Figure 4 for an example of a generated HTML query interface. The use of HTML here underscores the fact that the technique introduced in this paper can be applied to automatically generating Web application interfaces in some cases.

The class hierarchy (specified by the subClassOf property in the RDF) can be completely or partially shown. Typically just a subset of the classes might be shown. Hops down the class hierarchy are in fact displayed in our interface as offsets to the right from the parent class. Sibling classes in the inheritance/ specialization hierarchy are shown offset vertically from each other but with equal

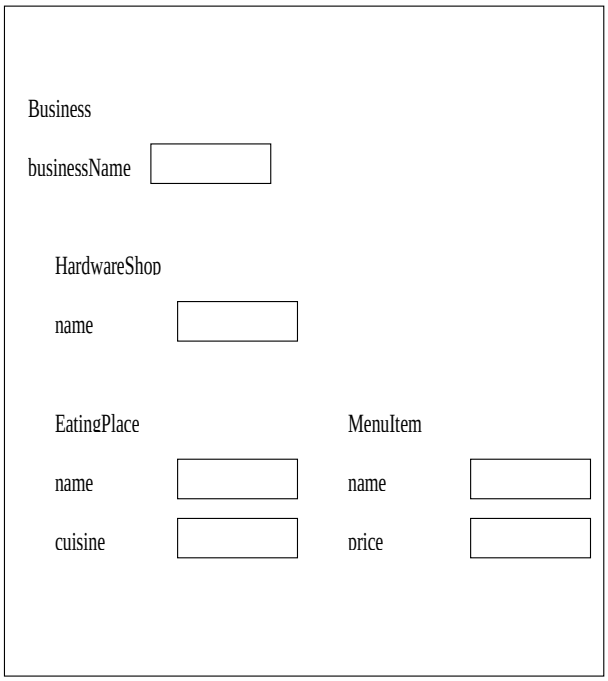

Fig. 4: Generated HTML page for querying

offset from the right. In our example this manifests itself as presenting the Business class at the top and HardwareShop and EatingPlace both below it but offset slightly to the right by the same amount (as they are sibling classes). Any child classes, of for example EatingPlace, would appear further to the right again and below the EatingPlace class.

Each property that has a RDF Literal as its range now has a *textfield* displayed next to its property name and each of these property names and associated textfields appears directly under the class which is the domain of the properties (See Figure 4). That is, attributes from the OO conceptual model become in the RDF representation, properties with RDF Literal as their range and now become the textfields into which the user can enter search words.

Classes in the hierarchy will be the domain of some properties that have as a range another class. This other class (the range class) will be displayed directly to the right of the domain class including its textfields that correspond to its properties with Literal domain. An example of this in Figure 4 is the MenuItem class, along with its *name* and *price* textfields, that is displayed to the right of the EatingPlace class.

4. Using the Query Interfaces

The advantages of the automatically generated query interface in terms of search capability are:

- Multiple meaningfully labeled textfields appear on the interface. Search words entered into these textfields are implicitly understood to be queries of the classes of the conceptual model/ RDF model, corresponding to the textfield used.
- The semantic mark-up of the resources/data and the semantic relationships this encodes allows the propagation of queries at query time to appropriately semantically related data that was not targeted directly by the user search

Each RDF statement is a triple of the form:
{pred, sub, obj}

Where pred is a property (member of Properties), sub is a resource (member of Resources), and obj is either a resource or a literal (member of Literals).

The notation [I] denotes the resource identified by the URI I and quotation marks denote a literal. RDF Schema [3] defines classes as "types" of resources.

A GUI textfield labeled p, corresponds to RDF property p.
A user is assumed to enter query "q" into the textfield p

Let C be a RDF class s.t. {domain, p, C}

Let R1
= the set of resources of class C that have "q" as the object of property p
= {r | {type, r, C} $\wedge$ {p, r, "q"}}

Let SubC
= the set of all descendent classes of C

Let SubCProp
= the set of all properties of descendent classes of C with Literal range
= {x | $\exists$ SC $\in$ SubC {domain, x, SC}$\wedge$ {range, x, rdf:Literal}}

Let R2
= {r2 | $\exists$ SC $\in$ SubC $\exists$ sp $\in$ SubCProp {type, r2, SC}$\wedge$ {sp, r2, "q"}}

The query results for query "q" in textfield p
= R1 $\cup$ R2

Fig. 5: Formal specification of search semantics

The first advantage mentioned means that the user implicitly provides more information to the search system than via a single-fielded interface. By choosing a particular textfield the user is signaling their interest in a particular class of information and these textfields in fact map to RDF classes that have defined semantic relationships with other data in the repository. This provides extra helpful information to be factored into the query processing. The automatic generation of the interface provides the dual advantage that it obviates the need for human effort in this task and also provides close semantic associations between the textfields and entered values and the semantics of the backend system and data.

The second advantage is that queries to a particular textfield can be propagated to semantically associated classes of data. In particular the system will propagate a query entered into a textfield to all descendant classes of the class to which this textfield is a property. The system will try to match the entered query value to all values of properties with a range having RDF Literal type of all descendant classes. This corresponds to the user having carried out the same search in all of the textfields of the descendant classes. A formal specification of these search semantics is provided in Figure 5.

For example if a user enters the word "Thai" into the *cuisine* textfield this will lead to a search of all *cuisine* elements e.g. <mn:cuisine> Thai food </mn:cuisine>, <mn:cuisine> Italian </mn:cuisine> found in the repository. This search will be implemented as an XPath query. There will be an attempt to match "Thai" against the text value found in each *cuisine* element. A list of associated EatingPlace elements will then be returned. The display of results would be the matching EatingPlace elements with just their properties with Literal class as their range having their values displayed. These could then be selected and be further searched.

If a user enters Thai in the businessName textfield of the Business class it will first be used to search against the text value of all businessName elements in the repository. However, the Thai string entered by the user will then also be used to search against all properties with RDF Literal as their range further down the class hierarchy (that is all properties that have textfields). In this case if Thai

were mentioned in the *name* attribute of any subclass of Business (HardwareShop or EatingPlace) or as the *cuisine* attribute of an EatingPlace these businesses would also be listed to the user. The search would not extend to MenuItem attributes – it only applies to attributes of the subClassOf descendants of Business on the basis that as they are specializations of the higher level class and the search is semantically relevant to them in this way.

In our example, a user can also type a food name directly into the MenuItem, *name* textfield. In this case all MenuItems across all EatingPlaces would be searched. The set of matching MenuItems would be returned to the user. The user will then be provided with "backlinks": in this case to the corresponding EatingPlaces.

5. Discussion

The query system implicitly adduces from users, semantic information about their query by requiring them to choose a textfield that has a defined semantic position in the data model. While a single-field keyword interface will match an entered keyword to any document or part of a data repository, this is not the case with this auto-generated query system. The system will start with a semantically constrained subset (set of elements) of the repository based on the textfield into which a query is made.

The auto-generation of query interfaces gives the ability to dynamically vary the interface in a number of ways. For example an interface representing a different subset of the conceptual model could be presented to different users based on their access role. This could be used to give larger or smaller search capabilities to different users. More trivially it gives the ability to generate a device-specific interface – in one case for a mobile device in another case for a PC client.

6. Conclusion

This paper has presented a technique for automatically generating a query interface to data from an object-oriented conceptual model of that data. The object-oriented conceptual model provides a unified semantic over heterogeneous data and is also the model used by many existing data stores. By generating a multi-fielded, domain-specific interface a superior from of search system can be provided. This ability to quickly build a powerful query system can help in the building of e-Commerce Web sites and is well suited to quick system integration as is required by the changing business partners in an extended enterprise. In addition via providing a ready benefit from RDF markup this may contribute to relieving the current "chicken and egg" problem that the widespread adoption of semantic markup is facing.

References:

[1] Consortium, W. W. W. 2000. Extensible markup language (XML) 1.0. Available at http://www.w3.org/ TR/REC-xml

[2] Consortium, W. W. W. 1999. Resource description framework (RDF) Model and Syntax Specification. Available at http://www.w3.org/TR/REC-rdf-syntax/.

[3] Consortium, W. W. W. 2000. Resource description framework (RDF) schema specification 1.0. Available at http://www.w3.org/TR/rdf-schema/.

[4] Consortium, W. W. W. 2001. XML Schema Part 0: Primer. Available at http://www.w3.org/TR/ xmlschema-0/.

[5] Dill, S., et. al. SemTag and Seeker: Bootstrapping the Semantic Web via Automated Semantic Annotation. WWW03, Budapest, Hungary, 2003.

[6] Dillon, T., Tan, P. Object Oriented Conceptual Models. Prentice Hall. Inc. 1993.

[7] Feng, L., Chang, E., Dillon, T. A semantic network-based design methodology for XML documents. ACM Transactions on Information Systems (TOIS), Volume 20, No. 4, 2002, pp 390 – 421

[8] Feng, L., Chang, E., Dillon, T. Schemata Transformation of Object-Oriented Conceptual Models to XML. Intl. Journal of Computer Systems Science & Engineering, 1, 45-60, 2003.

[9] Fensel, D., Angele, J., Decker, S., Erdmann, M., Schnurr, H.-P., Staab, S., Studer, R., and Witt, A., On2broker: Semantic-based access to information sources at the WWW, in World Conference on the WWW and Internet (WebNet99). 1999: Honolulu, Hawaii.

[10] Google. http://www.google.com.

[11] Hendler, J. Agents and the Semantic Web. IEEE Intelligent Systems. March/April 2001 (Vol. 16, 2).

[12] Maedche, A., Motik, B., Stojanovic, L., Studer, R., Volz, R. Ontologies for Enterprise Knowledge Management. In IEEE Intelligent Systems, January/ February, 2003.

[13] Noy, N.F., Sintek, M., Decker, S., Crubzy, M., Fergerson, R.W., Musen, M.: Creating Semantic

Web Contents with Protege-2000. IEEE Intelligent Systems 48(2): 60-71, 2001

[14] Rumbaugh J., I.Jacobson and G.Booch, The UML Reference Manual, Addison-Wesley, 1999.

[15] SOAP 1.1 Technical Report, http://www.w3.org/TR/SOAP/, W3C, 2000.

[16] Steele, R., A Web Services-based System for Ad-hoc Mobile Application Integration, IEEE Intl. Conf. on Information Technology: Coding and Computing '03, Las Vegas, 2003.

[17] UDDI.org: Universal Description, Discovery and Integration for the Web. http://www.uddi.org.

[18] Vdovjak, R., Houben, G.J.: RDF-based architecture for semantic integration of heterogeneous information sources. In: Workshop on Information Integration on the Web. (2001) 51-57

[19] Vdovjak, R., Barna, P., Houben, GJ. EROS: A User Interface for the Semantic Web. 7th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida, July 27-30, 2003.

[20] Verity Incorporated. http://www.verity.com.

[21] Wi-Fi Specification. Available at http://grouper.ieee.org/groups/802/11/.

[22] WAP Forum. Wireless Markup Language Version 2.0. Available at http://www1.wapforum.org/tech/documents/WAP-238-WML-20010911-a.pdf.

[23] Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl