INFORMATION MANAGEMENT AND KNOWLEDGE SHARING IN WIDE

T. SMITHERS[6], J. POSADA[6], A. STORK[1,†], M. PIANCIAMORE[2], N. FERREIRA[3], S. GRIMM[7], I. JIMENEZ[6], S. DI MARCA[5], G. MARCOS[6], M. MAURI[2], P. SELVINI[2], N. SEVILMIS[1], B. THELEN[4], AND V. ZECCHINO[5]

[1]*Fraunhofer Institut für Graphische Datenverarbeitung, Darmstadt*
[2]*CEFRIEL, Milan*
[3]*Centro de Computação Gráfica, Guimarães*
[4]*Schenck Pegasus GmbH, Darmstadt*
[5]*Italdesign - Giugiaro SpA, Moncalieri (Torino)*
[6]*VICOMTech, Donostia / San Sebastián*
[7]*FH Karlsruhe University of applied Sciences, Karlsruhe*

[†]*Author to whom all correspondence should be sent to at* `ist-wide@igd.fhg.de`

WIDE is an IST project whose aim is to investigate the use of emerging Semantic Web technologies, tools, and standards in the support of effective knowledge sharing and information management in real multi-disciplinary activities, such as innovative product design. This paper presents the approach to knowledge sharing and information support that has been developed and adopted by the WIDE product, the information system architecture that is being developed to test both this approach, and the Semantic Web techniques that are used in its implementation, some early results from the project, and a discussion of related work in information systems and Semantic Web techniques and tools.

1. Introduction

The principal aim of the WIDE[1] project, is to investigate the use of emerging Semantic Web technologies and standards (see [1, 2]) to support effective information management and knowledge sharing in innovative product design. To do this, a prototype information support system is being developed to support the different kinds of designers and engineers involved in modern product design teams.

Sustaining innovative product design is fundamental to the continued success and viability of companies in the engineering sector, particularly for the small to medium size enterprises (SMEs) that dominate this sector in Europe. Effective innovation depends upon a shared knowledge of what has been done before, and a shared understanding of new options and opportunities. The key to sustaining innovation is thus effective support of the information management and knowledge sharing activities of the multi-disciplinary design teams involved.

[1] Semantic Web-based Information Management and Knowledge Sharing for Innovative Product Design and Engineering (IST-2001-34417): <`http://www.ist-wide.info/`>.

The WIDE project seeks to understand how effective information management and knowledge sharing support can be delivered using a combination of conventional (existing) information system and Semantic Web techniques. Semantic Web techniques are seen as a way of providing effective support for the different kinds of designers and engineers involved in product design, in a flexible and manageable way. In seeking to apply Semantic Web techniques to a real industrial need, the WIDE project also constitutes an important test for current Semantic Web techniques and standards.

2. The WIDE Approach to Knowledge Sharing

The main challenge in seeking to provide effective support for product design teams, is to both accept and to actively support the different ways of working of the different kinds of designers and engineers involved, and to effectively support their knowledge sharing and inter-working. Concept designers and product engineers, for example, need active support to understand each other, and thus to share their knowledge in resolving design issues and problems.

This need for sharing knowledge, is often characterised as a need for a *common understanding* in Knowledge Management, and a *common working language* is usually prescribed to deal with it. Davenport and Prusak [3], for example, argue that "people can't share knowledge if they don't speak a common language." When it is possible for two or more communities to agree upon such a common language, this approach has been shown to work, but it is not always possible, or desirable. This is the case for innovative product design teams. The education, training, and practices of concept designers, industrial designers, graphic designers, product engineers, systems engineers, etc. all tend to encourage quite different approaches to, and points of view of the products they are involved in designing that result in quite different ways of thinking and talking about what they do: they each have their own working culture.

Effective knowledge sharing does not necessarily require sharers to speak a common language or terminology. It is sufficient for them to speak their own 'language,' (terms) *and* be able to understand the 'language' (terms) of the other (or others). Nor, in practice, does it require a complete understanding of what the others say; just a sufficient understanding in particular contexts. The WIDE approach to supporting knowledge sharing, is thus to support different kinds of users, who are trying to find information using their own terminology. And to support other kinds of users, who are involved in the same task, by (re)presenting queries and returned information in the terms of these other users, and not just in the terms of the user who requested it.

An important consequence of adopting this approach is that the WIDE information system *must* also be able to deal with the common and natural situation

that designers and engineers, of all kinds, often find themselves: they know well what kind of information they need or want, but they do not know how to effectively ask for it. The way any particular kind of designer might know about or talk about some subject or concept may not be (and often is not) the same terms that are used to organise and store the information he or she needs. To deal with this issue, forming an effective information request is treated as a design activity, not as a query specification task. In other words, the WIDE information system supports users in designing effective information requests, rather than supporting them in specifying correct queries in some query language. Designing effective queries is modelled as an incremental exploration of different possibilities: a model which is based upon a knowledge level theory of designing [4, 5].

The WIDE information system architecture and functionality, which we will now present, reflect this WIDE approach to effective knowledge sharing in multidisciplinary design teams, and the support of information query design.

3. The WIDE Information System

The WIDE Information System is divided in to three basic levels: the User Interface level; the Meta Level; and the Content level. Each of these levels are implemented as independent subsystems, and a fourth subsystem, the Agency—a multi agent system—is used to "glue" them together.

The User Interface (UI) provides a graphical front end to the user. This supports both the incremental development of a user query (UQ), and the presentation of the returned results. The Meta Level (ML) supports the user query design, and subsequent semantic processing of a user query into System Queries (SQs). To do this, the ML uses a domain ontology (for car design, in the WIDE project), together with a Task Type ontology, User Type ontology, and dictionaries of description terms and user type terms. All these different kinds of knowledge are used to produce the SQs that are then passed to the Agency. This subsystem identifies and locates information sources in the Content Level (CL) to which the SQs can be sent to produce effective returns. This may require some term mapping or reformatting of the SQs, and the mapping back of any returns. The Agency also provides the WIDE system's gateway to the Web, which is also considered part of the CL. Essentially, Web sites and Web search engines are treated as weakly structured information sources. The CL also contains "in-house" information sources, which consist of legacy database systems, which are well structured, but may have only weak metadata descriptions, or use particular terminologies. Figure 1 presents the main components of the WIDE Semantic based Information System, and how they are related.

A WIDE Information System user first logs on to the system as a known user, and type of user, and selects a task from the systems' Task Type ontology. The

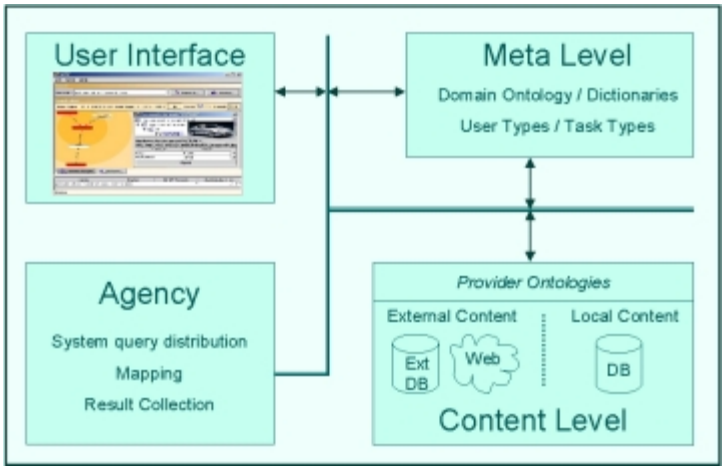

Figure 1: The WIDE Information System Architecture

users' personal preferences profile, together with the User Type and Task Type information is then used to configure the way the system responds to and supports the user in developing useful queries, and in the way it presents returned query results. A personal profile specifies a user's preferred terms, user interface settings, and user role (used by the ML to select an appropriate User Dictionary). A User Type defines the kinds of tasks a user can perform, and which User Dictionary is used by the system to map user terms into the ML internal terminology. Task Type specifications are used to configure the kind of query a user needs to develop for the kind of task that he or she has selected.

The UI provides text-based and a graphical-based support for users to specify information queries. The graphical version uses domain knowledge (from the ML, suitably selected and presented using the User Type and Task Type specifications) to offer the user a point-an-click way of building correct queries. Users can use a combination of both text input and graphical selection to form a query. This is then checked against a BNF grammar [6], for correctness, and passed to the ML. The ML then processes the User Query UQ, using its domain ontology, User Type Dictionary, and personal user dictionary, to discover what other concepts it has that are related to the concepts in the UQ. These further domain ontology concepts, and the ways they are related to the UQ concepts, are then returned to the UI as an ontology fragment that represents the UQ and its immediate conceptual context, where it is graphically presented to the user. This ontology fragment, or Query Structure, as it is called, supports further navigation of the concepts and properties present, allowing the user to further extend the fragment by including further concepts along selected relations. In this way, a user is able to see how

the system understands his or her query, and is supported in further exploring around it, to see how it might be changed, adapted, or extended, to be form a more effective query: more precise and/or more complete.

The ML initially supports this user query design stage, it then processes the UQ into System Queries (SQs), which are then passed to the Agency. The ML first uses its domain, Task Type, and User Type ontologies, together with the User Type Dictionary to generate a standard internal form of the UQ, by translating all recognised terms in to ML internal terms, converting all plural terms to singular form, and making everything lower case. The BNF parse tree of this internal form UQ is then expanded to include other related terms and relations, that can be inferred from the domain ontology, and by terms that can be identified as synonymous. This expanded form of the UQ structure is then used to generate a set of SQs (expressed as an RQL query [7]), each one being derived from some part of the expanded UQ structure.

So, for example, a concept designer might start by asking for

UQ1: Photographs of Maserati cars

In response, the UI (with support from the ML) would show that it understands *photograph* to be a kind of picture, where *drawing*, *image*, and *sketch* are other kinds of *picture* concepts. As a result, the user might then change the query to

UQ2: Pictures of Maserati cars

to be more inclusive of other possible kinds of pictures. This is then transformed by the ML in to the following internal form

UQif: picture about maserati car

where 'picture' is a know document type, 'about' is the term used to connect the document type to the concept, and 'maserati' and 'car' are understood as two terms forming an attribute value qualifying phrase. The ML then expands this UQ, based upon its knowledge that maserati is the name of an individual of the concept *brand*, and that *brand* is defined as the range of a *has property*, whose domain is *car*. The resulting expanded SQ thus looks like

SQx: picture $\xrightarrow{shows}$ car $\xrightarrow{has\_a}$ brand $\xrightarrow{is\_a}$ maserati

Expressed as an RQL query, this then looks like

```
SELECT pt, mc
  FROM {pt:$pt} @p {mc:$mc},{rc1} @w_a1{c1:$c1},
    {rc2} @w_v1 {v1:Literal}
  WHERE @p = ''has_info_about''
    AND ($p1 = ''PICTURE'') AND $mc = ''CAR''
    AND mc = rc1 AND @w_a1 = ''with_attr''
    AND $c1 = ''BRAND'' AND c1 = rc2
    AND @w_v1 = ''with_value'' AND v1 = ''MASERATI''
```

Which, says: select all the pictures and all the cars where picture is a presentation type that has_info_about the cars and the cars have an attribute named brand, whose value is maserati.

The Agency takes this SQ (in RQL) and sends all or parts of it to the various different information sources in the CL, adapting the format in each one. These information sources may be existing well-structured databases, partially structured document stores, image stores, or Web sites, for example. In general, there is *not* one single information source that can return query matches, and, in general, the different information sources will not store the matching data in the same way, nor in the same terms. Furthermore, different information sources may only contain data to match parts of a complete SQ. To deal with this (typical) situation, the Agency adopts a two-stage process. It first broadcasts an SQ to all the CL information sources. Each information source then responds saying which part of the full SQ it can try to match. With this information, the Agency, decides on which (sub)set of information sources to ask returns from. It then collects all the returns, from the various information sources, and prepares them for passing back to the ML.

In order for this process to work with reasonable efficiency, each information sources needs to have an explicit characterisation of the data it contains, its organisation (data model) and the data terms used. This is called a *provider ontology*. Since, again, in general, we cannot expect there to be a simple, nor exact mapping between the provider ontologies of the information sources in the CL, and the ML domain ontology, the SQs, generated by the ML, need to be adapted for each information source. This gives rise to a kind of negotiation process. First a CL information source tries, using its provider ontology, to find synonyms matches for any unrecognised SQ terms, if this fails to resolve an unrecognised SQ term, it can ask the ML (via the Agency) for other synonym suggestions.

To make both existing and new data more accessible to SQs generated by the ML, and so more accessible to users, the information source provider ontologies are also used to *annotate* the data in the information source. We call these information sources *semi-enriched* information sources.

As a result of all this SQ processing by the Agency and the different information sources in the CL, a set of data matching at least parts of the SQs is formed. This set of System Query Returns contains both the matched data item, an identification of which information source provided it, and any local context information from the information source provider ontology. The Agency is responsible for compiling this return set, and for passing it to the ML. The ML then tries to order the returns using its own domain ontology, together with the User Type and Task Type ontologies, and to translates terms into the known user terms. It also structures these returns according to the original UQ relational structure, and passes everything back to the UI, where these results are graphically presented. In this way, a user

is presented with a response to his or her query, formed by returns from the different information sources, and presented using the structure of his or her original query; a structure which he or she is familiar with, having actively developed it while designing the query.

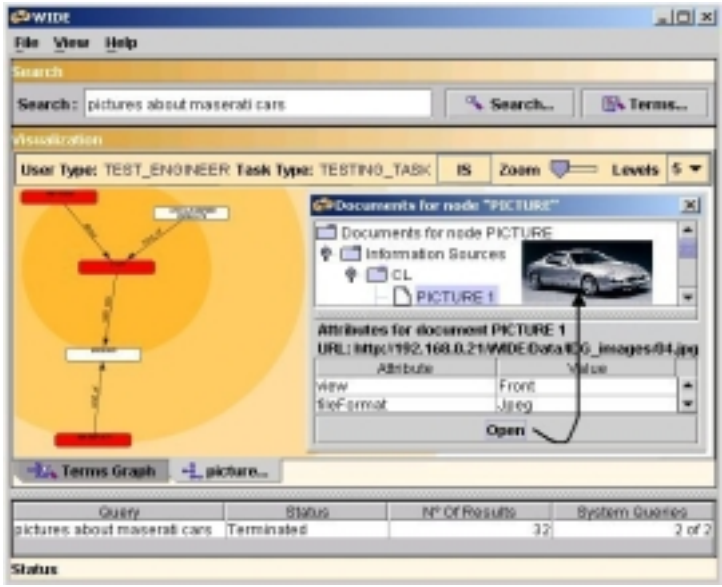

Figure 2: The WIDE Information System User Interface

4. Current Status, Early Results, and On Going Work

The complete WIDE Information System has been implemented and developed as an incremental series of prototypes. The current prototype includes an image store, developed by Italdesign Giugiaro SpA (one of the industrial partners), and an existing ASAM-ODS [8] database at Schenck Pegasus GmbH (the other industrial partner). Each prototype has also been tested and positively evaluated by the two industrial partners. The results of these user tests have played an important role in identifying needed improvements and new functionalities, and have confirmed the basic WIDE approach and system architecture. The need to explicitly support processes (sequences of partially ordered tasks) is one of the new functionalities currently being developed, and for this a new Process ontology is being developed for the Meta Level, together with an extension of the User Interface to support collaborative working in processes.

The implementation and development of the prototype systems have also made significant use of Semantic Web techniques, methods, and tools. The most important ones being RDF/RDFS [9, 10], OWL [11], RQL [7], Protégé [12], Sesame

[13], and RACER [14]. Based upon the work done so far, the main conclusions with respect to the use of these can be summarised as: (i) OWL better supports the knowledge representation work involved in building the domain ontology, and other ML ontologies; (ii) RQL offers an effective low level query language; (iii) Protégé, with the OWL plugin, provides a good ontology editor and development environment; (iv) Sesame, like other general purpose ontology stores, is currently too slow to support the kind of ontology-based inferencing needed by the ML; (v) RACER can provide useful support to ontology development, but becomes too slow for ontologies like the ML domain ontology (with approximately 790 concepts and individuals, and 150 relations); and (vi) none of the published ontology development methods [15], either do not have a validation step, or are strong enough to support effective validation of realistic sizes of ontologies. The rather toy examples typically used to present these methods also don't help much in understanding how to apply them to real ontology developments.

References

1. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, pp 34–43, May 2001.
2. S Taab and R. Struder, (Eds.), "Handbook on Ontologies," Springer Verlag, 2004.
3. T. H. Davenport and L. Prusak, "Working Knowledge: How Organizations Manage what they Know," Cambridge, MA: Harvard Business School Press, 1998.
4. T. Smithers and W. O. Troxell, "Design is intelligent behaviour, but what is the formalism?", AI EIDAM, Vol. 4, No. 2, pp. 89-98, 1990.
5. T. Smithers, "Synthesis in Designing", in J. S. Gero (Ed.), Artificial Intelligence in Design 02, Dordrecht: Kluwer Academic Publisher, pp 3–26, 2002.
6. Backus-Naur form (BNF), WIKIPEDIA, <http://en.wikipedia.org/wiki/Backus-Naur_Form>.
7. The RDF Query Langauge (RQL), FORTH Institute of Computer Science, <http://athena.ics.forth.gr:9090/RDF/RQL/>.
8. Association for Standarisation of Automation and Measuring Syetems (ASAM)Open Data Service (ODS), <http://www.asam.net/01_asam-ev_01.php>.
9. Resource Description Framework (RDF), W3C Semantic Web Activity, Technology and Society Domain, <http://www.w3.org/RDF/>.
10. RDF Vocabulary Description Langauge 1.0: RDF Scheme, W3C Technical Reports and Publications, <http://www.w3.org/TR/rdf-schema/>.
11. OWL Web Ontology Language Overview, W3C Technical Reports and Publications, <http://www.w3.org/TR/owl-features/>.
12. The Protégé Ontology Editor, Stanford Medical Informatics, Stanford University School of Medicine, <http://protege.stanford.edu/>.
13. J. Broekstra and A. Kampman and F. van Harmelen, "Sesame: A generic architecture for storing and querying RDF and RDF Schema, International Semantic Web Conference (ISWC), pp 54-68, 2002.
14. RACER: Semantic Middleware for Industrial Projects based on RDF/OWL, <http://www.cs.concordia.ca/ñaarslev/racer/>.
15. A. Gómez-Pérez, M. Fernández-López and O. Corcho, "Ontological Engineering," London: Springer-Verlag, 2004.