Bridging the Gap between the Semantic Web and Existing Network Services

Nickolas J. G. Falkner Paul D. Coddington Andrew L. Wendelborn
School of Computer Science, University of Adelaide, Adelaide, South Australia 5005,
{jnick,paulc,andrew}@cs.adelaide.edu.au

Abstract

This paper presents an overview of a mechanism for bridging the gaps between the Semantic Web data and services, and existing network-based services that are not semantically-annotated or do not meet the requirements of Semantic Web-based applications. The Semantic Web is a relatively new set of technologies that mutually interoperate well but often require mediation, translation or wrapping to interoperate with existing network-based services. Seen as an extension of network-based services and the WWW, the Semantic Web constitutes an expanding system that can require significant effort to integrate and develop services while still providing seamless service to users. New components in a system must interoperate with the existing components and their use of protocols and shared data must be structurally and semantically equivalent. The new system must continue to meet the original system requirements as well as providing the new features or facilities. We propose a new model of network services using a knowledge-based approach that defines services and their data in terms of an ontology that can be shared with other components.

Keywords: distributed system, ontology, semantic web, domain name system

1 Introduction

The introduction of wide-area networking has led to the requirement to produce large-scale widely distributed network information services that provided a range of data to hosts on the network. These diverse services include the complex mappings of domain name information, from the Domain Name Service (DNS), and the distribution of synchronised time information available from the Network Time Protocol (NTP) servers. As the network infrastructure has become more widespread, so too has the range of services and the types of information that are used by systems.

In this paper, we will build from our previous work on mapping the DNS into a knowledge-domain representation [3, 2], and the lessons learned from that, to discuss using a generalised approach to provide components that can integrate with multiple existing distributed systems at the same time. This provides interoperation, translation and full provision of services. There are many different benefits of this work and these are explained in detail elsewhere [5]. In this paper, we concentrate on bridging and translation services. We discuss the development of two complex distributed systems, the DNS and the Semantic Web, in order to illustrate the potential problems that can arise during interoperation and describe the method and implementation of our approach.

The development of the DNS (in the late 80s) was motivated by a need to provide name to IP address mappings for an increasingly large number of computers. The final system design is hierarchical, widely distributed and robust[11, 12] but, most importantly, it meets existing requirements and allows systems to expand their capabilities.

The Semantic Web is an extended application of the underlying technology of the World Wide Web (WWW), enhanced with metadata annotation to allow improved reuse and data sharing [6]. The semantic web adds layers of context to data which can associate meaning with the stored data. The semantic web employs eXtensible Markup Language (XML)[15], Resource Description Framework (RDF)[9] and Web Ontology Language (OWL)[14] to structure documents, model the data, provide a vocabulary for the data and allow the expression of relationships between data. The data contained in the semantic web is metadata rich and, potentially, heavily annotated. The metadata can become a mandatory requirement for true interoperation between services and prevent interoperation if absent.

The burden of annotation becomes significant when the information systems provide streams of data in specialised formats or have no in-built concept of metadata. The WWW already has some concept of metadata, albeit limited, because the Hyper Text Markup Language (HTML)[13] has information that describes the content. There is no semantic overlay but there is a conceptual step that takes us above the data to describe it. We will always encounter an annotation burden if we wish to insert data into an annotation-using, or meta-data rich, system from an un-annotated source. How-

Proceedings of the 17th International Conference on Database and Expert Systems Applications (DEXA'06)
0-7695-2641-1/06 $20.00

ever, some of these sources are valuable and the addition of their data, as well as relevant metadata and provenance information, is extremely useful.

The components of a distributed system must be capable of successful intercommunication as, without the ability to transfer information between elements, the system cannot function. When a new system, such as the semantic web, seeks to interact with a pre-existing network service, the two systems must also be able to intercommunicate. We propose the use of ontologies to provide such interoperability, and to specify the relationships, classes and properties of metadata. These techniques are integrated with existing systems, to provide enhanced clients or servers, or used in mediating servers that analyse service data and automatically annotate the datastream.

2 Motivation

The semantic web allows the use of metadata to place a structure on the data stored and used within a system and to show the relationships between this data. Ontologies can then be used to interpret the metadata and classify it to provide a truly machine-interpretable form of the data. Data within the system is annotated and context-rich. By comparison, data outside the system is context-free and, hence, meaningless to an automated system. Any system that is a source of data without annotation will either need to be modified to add the annotation or the data user will have to find some way to annotate the data source. We use the term *wrapper* to describe a system that provides a semantic web-friendly annotated data stream. Logically, this wrapping function can be located within a system or on the notional boundary of the system. The term alignment is used to describe the activity of matching up one system's semantic or structural definitions with those of another system. This alignment does not always proceed smoothly, especially when one system has a strongly defined set of metadata and the other systems lacks even the notion of metadata. We define *boundary translation issues* as those problems that cause misinterpretation of data due to errors in the semantic classification, mis-alignment between the two semantic representations or errors in structural equivalence. If boundary translation issues are minimised then data can move in and out of the semantic web with ease and, significantly, can have the same usefulness and semantic-richness in both environments. We show how these issues can be minimised by defining an ontology that spans more than one system or service.

Ontologies are already in use within the semantic web and are a vital tool in the organisation and interpretation of data. Ontologies can represent a distributed system as if it were composed of abstract knowledge domains. A concrete instantiation of the conceptual representation gives a statement of the system, as a knowledge domain, but in an actionable form. We already have semantic web data systems defined using ontologies. It is a logical extension to define other network-based systems ontologically, such as the existing network-based information systems, and it is a small step to take this to its logical conclusion by stating these systems initially as knowledge domains. It is intuitively simpler to derive a working program from a sound knowledge model than it is to derive a correct knowledge model from an existing program, providing that we have a sound and mature environment to support a system described in this manner. A knowledge-based model also provides us with a sound basis to improve automated alignment and, hence, reduce boundary translation issues and we discuss this in Sections 4 and 5.

3 Benefits

Alignment is often an arduous and predominantly manual process that is supported by automatic tools, rather than a process that can proceed autonomously. Rapid alignment reduces the time taken to integrate new systems. Other work [1, 7] uses ontologies for system alignment but focusses on components in workflows, rather than large-scale systems, and also starts by aligning structural information and using a registration scheme to provide hints for semantic alignment. This alignment is provided by the production of an XQuery or XPath translation between the output of a component and the input to the next component. While a similar functionality is captured in our model, we also look at the addition of transformations to the components themselves - if we can gain access to them. Our rewriting rules are not constrained by any representation or overall structure.

We also propose the use of ontologies for operational, as well as structural, information to provide the ability to change the operation of a system by changing its knowledge model, rather than requiring a programming change. We have discussed this in our earlier work with the DNS [2, 3] as a means of adding additional functionality at a local level without causing problems at a global level. The key benefit is that we can restrict our operational changes to only those areas that deal only with semantic and structural alignment. We can isolate those components of clients and servers that may interact with semantic web, or otherwise metadata-responsive systems, and allow them to engage in more meaningful exchanges. The separation of those aspects that must change and the aspects that can stay the same is a key benefit of our solution and reduces the dependency on what has been done before.

By adding in our own clients and servers which can communicate with the unenhanced system and also communicate to the semantic web with annotation, we can take advantage of both systems without significant penalty. These

new nodes can operate as full members of any and all of the systems described in their ontologies. They are also capable of translating from one system's information to another as they have rewriting rules, defined in the ontology, to translate to and from their internal storage format for all described systems.

Although we expect to achieve increased ease of interoperation, we realise that this will come at a performance cost in the first instance. This is due to the overhead in translation and the movement of some aspects of services from compiled code to interpreted code. Our implementation shows that the overheads need not be so large as to overshadow the benefits of automated annotation streams operating from the originating server, rather than from a secondary repackaging node. Importantly, a change to the data or data structures at the server automatically updates the outgoing accompanying data and does not require the re-coding of a separate node acting as a data annotator or enhancer.

4 Method

We use a divide-and-conquer strategy to separate the independent components of the system under analysis into those aspects that deal with the logical relationships between data, those that deal with the representation of data structures, and those that describe the operational semantics of the system. Definitions of classes vary depending on whether we are looking at logical, representational or semantic branches of the ontology but, in the majority of cases, class membership is based on key properties that identify relationships between elements. We must regard the system in terms of data and data transformations.

Classes are defined to match the major data components of the underlying system in a way that also reflects the transformation of one form of data to another. Key components of the data should have their own classes, as should the defining data that is presented to the user. Where data has a common purpose, but possible minor variations, it can be grouped into one class providing that the same structure and operational semantics can be used for all variations. Once the data sets stored by the system have been encoded into classes, we can then encode the data. In our implementation, classes are assigned to key data sets,t he families of data transformations, data sets that can be grouped because of commonality of handling and purpose and characteristics which may be derived from OWL constructs rather than data entry.

For example, in the Domain Name System, domains (groups of hosts under common administration) can contain smaller sub-domains with hosts forming one class and domains forming another. Domains must have properties associating them with hosts. In the same way, hosts must

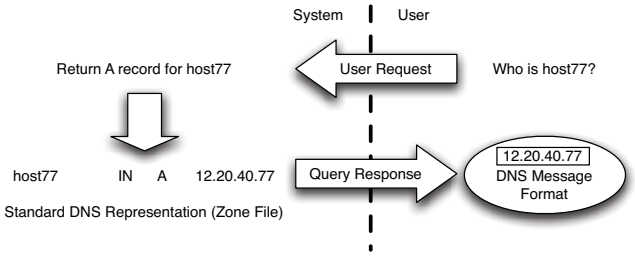

Figure 1. A standard DNS query/response

have properties associating them with IP addresses. The provision of name or IP address information is one of the key requirements of the DNS and it is a logical choice for a class. The output that the DNS presents to a user is delivered in DNS message format[11]. Figure 1 shows a standard DNS query/response pair. Figure 2 shows how, given the query of Figure 1, our approach will represent the data stored in the DNS, the result delivered to the user and the possible alternative form of response. We now have a possible response built from the available information and using a separate data transformation that produces output in XHTML, rather than DNS message format.

In our approach, we can define the additional OWL constructs required to infer additional data from that which is present in the system. For example, the existence of a final point in a hierarchical representation implies that, if the representation is known, we can infer the existence of other aspects of the hierarchy. In the DNS, the existence of the domain "cs.adelaide.edu.au" implies that there are at least domains called au and edu.au. We could also infer the existence of a domain adelaide.edu.au and produce associated place holder records and an overall structure that extends our understanding of the context of our data.

We now have a basic model of the data in the system and have added inference rules. We refine transformations to complete the definition of the initial model of the original system, and define transformations that convert to and from a storage format, convert to and from an internal for-

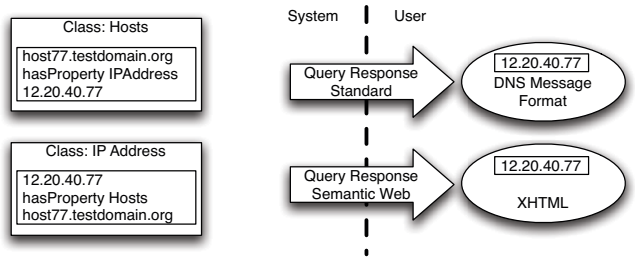

Figure 2. A DNS query supported by knowledge domains

mat and convert to and from the format that is seen by users. Many of these transformations involve multiple sources of input to produce the correct output and are complex methods, rather than simple rewriting rules. To be more precise, they are heavily context-dependent and often use metadata associated with the input to produce the correct output. Where possible, the transformations are simple, small code elements, as this allows reuse and facilitates model verification. Large transformations are made up from a series of transformational components, rather than writing many slightly different large transformations. We see this component model, that extends from the beginning all the way through to the final information system, as a key aspect of our approach. If the entire system is described through the definition of components and their composition, or interactions, then we have a flexible basis for developing new system models and new operational modes.

To maintain the standard system operation, the standards-based component of the information system is identified and encoded as an ontology. We import this core ontology into a parent ontology that will define the whole system. The parent consists of two ontologies, the *core* and the *extension*. This is an administrative mechanism to prevent the accidental removal or alteration of standards-based aspects of the system. The extension ontology is a separately written ontology that extends existing function, or provides alternatives, under strictly observed criteria. These are that the extension will not prevent a standard system component from obtaining correct results, the extension does not duplicate a function in the standard ontology, and that the extension is appropriate for the system.

The first criterion is self-explanatory in the context of maintaining global operation. The second prevents us from having to carry out parallel maintenance if the standard ontology changes. The third prevents the overloading of an existing, and specialised, network information system with unnecessary function.

5 Implementation

Our system is Java-based and uses the HPL Jena[10] libraries to represent RDF graphs as 3-tuples in memory. OWL/RDF/XML is read in and is used to produce the internal representation which is an annotated 3-tuple space. Part of this annotation consists of links to functional libraries exposed from underlying operating systems.

On system start-up, the system-definition ontology is read in and transformed to the internal representation. At this point, the network ports are opened in one of two modes. If the system is running as a standard implementation then the network ports are opened up supporting only one network protocol. If multi-protocol format is used, to allow service overloading, then a single port can take protocol requests and will parse the input sequence to determine the appropriate decoding and operational transformations.

The ontology can be considered as a group of smaller ontologies that are logically associated with each other because of their focus, with the smaller ontologies strongly focussed on a specific area of interest associated with either the problem domain or the provision of the service. The subordinate ontologies, or branches, have differing levels of granularity depending on their focus but are all linked together to provide a consistent knowledge-based representation of the service and its provision. For example, the logical relationship of elements in the DNS hierarchy is coarse-grained as it deals with discrete elements in a hierarchical relationship. However, the ontology that describes the functional operation of user-defined function elements in the system is fine-grained as it supports the decomposition of component services and must also co-operate with the defined thread and tasking model defined for the service.

Once the final ontology is produced, the same ontology is used for two separate knowledge representations in the final system. The first is the conceptual representation that specifies the system as it should look, at a given state depending on the operational semantics, and the second is the executing representation that shows how closely the executing system is to the conceptual model. The key difference between the two representations is that the RDF graph of the conceptual model has blank nodes in it while the executing system is a fully-instanced graph with no blank nodes. We do not discuss this in detail here but we use the RDF semantic property of entailment to compare the two graphs and carry out system repair or assessment of alternative operational modes [8].

To demonstrate the production of data for the semantic web and web services, our trial implementation provides a DNS service that can be delivered in DNS message format, SOAP/XML, XML and XHTML. It operates in both single-protocol and multi-protocol format for DNS message format, SOAP/XML and XML. It provides an additional port for XHTML to prevent confusion with XML messages. We describe this in more detail in other work [4].

Figure 3 shows an output stream composed from the traditional DNS items with metadata annotation. The DNS data required for, say, an experiment needs a certain level of provenance metadata associated with it and, in Figure 3, the Network Time Protocol (NTP) timestamp and Zone file serial number allow the provenance of the DNS data to be established. All of this information is automatically associated with the outgoing data and any change to the underlying structures will also cause changes to this output data. There is no need to recode a separate wrapper to cope with a new annotation or transformation strategy.

We have used this to provide DNS data streams annotated with time of response, server responding, age and se-

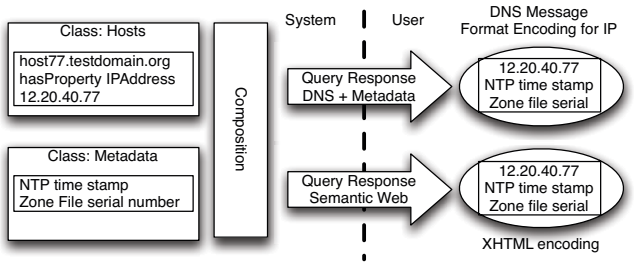

Figure 3. DNS and Metadata Composition

rial number of the network information source and detailed packet information to show where packets were going while the request was being carried out. Clients included standard DNS clients, enhanced DNS clients that use ontologies, and semantic web browsers.

6 Conclusions

The semantic web is a very useful technology but it suffers from boundary translation issues when using unannotated data sources. When these data sources are useful network-based information systems, these translation issues can lead to difficulty in assimilating and using information that can affect system reliability.

We have demonstrated that an ontological representation of a system can be used to capture the behaviour and structures of several systems and allow them to coexist on the same host. This allows the host to function as a fully-operational node in more than one system and, under control, translate one system's information to another format. In this way, we can annotate the data from an unannotated network service and inject it into a system that makes use of, or requires, metadata - such as the Semantic Web.

By looking at existing systems as a set of data and associated transformations, and capturing this knowledge in ontologies, we can provide enhanced systems that can function as the bridge between the semantic web and existing systems. However, these systems are not just bridges, they are fully functional members of both communities that have inbuilt translational capabilities. Most importantly, these capabilities can be altered for local requirements without causing widespread problems and they can be altered without having to recode the underlying server.

References

[1] S. Bowers and B. Ludäscher. An ontology-driven framework for data transformation in scientific workflows. In *Proc. of the 1st Intl. Workshop on Data Integration in the Life Sciences (DILS)*. Springer, 2004.

[2] N. J. G. Falkner. Towards a Semantically Enhanced Internet: Developing an Ontology for the Domain Name System. Technical Report DHPC-161, The University of Adelaide, October 2005.

[3] N. J. G. Falkner, P. D. Coddington, and A. L. Wendelborn. Developing an Ontology for the Domain Name System. In *Proc. of the 4th Intl. Workshop on Web Semantics*, Copenhagen, 2005. IEEE.

[4] N. J. G. Falkner, P. D. Coddington, and A. L. Wendelborn. Optimising Performance in Network-Based Information Systems. Technical Report DHPC-171, The University of Adelaide, 2006.

[5] N. J. G. Falkner, P. D. Coddington, and A. L. Wendelborn. Using Ontologies to Support Customisation and Maintain Interoperability in Distributed Information Systems and an Application to the Domain Name System. Technical Report DHPC-174, The University of Adelaide, 2006.

[6] D. Fensel, W. Wahlster, H. Lieberman, and J. Hendler. Introduction. In D. et al, editor, *Spinning the Semantic Web*. MIT Press, 2003.

[7] C. Goble, C. Wroe, P. Lord, J. Zhao, and R. Stevens. Using Semantic Concepts in the myGrid Project. GGF9, Chicago, Oct 2003.

[8] P. Hayes. RDF Semantics, 2004. http://www.w3.org/TR/rdf-mt/.

[9] F. Manola and E. Miller. The RDF Primer, 2004. http://www.w3.org/TR/rdf-primer.

[10] B. McBride. Jena A Semantic Web Framework for Java. http://jena.sourceforge.net/.

[11] P. Mockapetris. Domain Names–Concepts and Facilities. RFC 1034, 1987. http://www.dns.net/dnsrd/rfc.

[12] P. Mockapetris. Domain Names–Implementation and Specification. RFC 1035, 1987. http://www.dns.net/dnsrd/rfc.

[13] D. Raggett, A. L. Hors, and I. Jacobs. HTML 4.01 specification. http://www.w3.org/TR/html4/, 1999.

[14] M. K. Smith, C. Welty, and D. L. McGuinness. OWL Web Ontology Language Guide, 2004. http://www.w3.org/TR/2004/REC-owl-guide-20040210/.

[15] F. Yergeau and et al. eXstensible Markup Language (XML) 1.0 (Third Edition), 2004. http://www.w3.org/TR/2004/REC-xml-20040204/.