An Architecture for Standardized Terminology Services by Wrapping and Integration of Existing Applications

Ronald Cornet MSc, Antoon K. Prins MSc
Dept. of Medical Informatics, Academic Medical Center, University of Amsterdam
Amsterdam, The Netherlands

ABSTRACT

Research on terminology services has resulted in development of applications and definition of standards, but has not yet led to widespread use of (standardized) terminology services in practice. Current terminology services offer functionality both for concept representation and lexical knowledge representation, hampering the possibility of combining the strengths of dedicated (concept and lexical) services. We therefore propose an extensible architecture in which concept-related and lexicon-related components are integrated and made available through a uniform interface. This interface can be extended in order to conform to existing standards, making it possible to use dedicated (third-party) components in a standardized way. As a proof of concept and a reference implementation, a SOAP-based Java implementation of the terminology service is being developed, providing wrappers for Protégé and UMLS Knowledge Source Server. Other systems, such as the Description Logic-based reasoner RACER can be easily integrated by implementation of an appropriate wrapper.

INTRODUCTION

During the last decade, standards have been proposed for providing terminology services. Despite the emergence of these standards, implementations thereof have not yet come to fruition. Currently existing applications for browsing terminological systems are dedicated to one system, such as SNOMED or GALEN, often in a manner where the functionality is integrated with a particular user-interface. In order to make terminological systems and the related functionality more exchangeable, we define a client-server based architecture. In order for the server to be generically useable, it has to provide a standardized interface that clients can use to provide functionality to users. We will discuss issues that need to be addressed with respect to such an interface. Finally, we will describe a modular, layered approach for a terminology server, based on reuse of existing special-purpose applications, providing their functionality in a generalized manner.

BACKGROUND

The department of Medical Informatics at the University of Amsterdam performs ongoing research and development on terminological systems and services. This has lead to development of a terminological system on Reasons for Admission in Intensive Care, called DICE [1, 2], representing concepts in both Dutch and English. The main reasons for development of this system were the need for a terminology that supports semantic definitions of concepts, and assignment of multiple synonymous Dutch terms to these concepts.

To facilitate knowledge modeling and terminology-based registration, a terminology server for this system has been developed, together with clients for modeling and browsing. This implementation has various limitations, impeding the generic usability of the applications. The terminology server (written in Java) implements Remote Method Invocation (RMI), a protocol requiring clients to be written in Java as well. The storage format (serialization) of both the concept knowledge and the lexical knowledge is also Java-specific, making it difficult to exchange data with other systems. User evaluations have demonstrated that language-processing features should be enhanced to better fit the users' needs, for example to construct a concept from a detailed natural language expression entered by a user. Finally, as we aim at supporting advanced reasoning by deploying the powers of Description Logic-based reasoners, we want to incorporate such a reasoner into the terminology service.

Client/Server Terminology Implementations

The plea for a client-server-based approach for terminology services is not new. Already in the early nineties of the last century, the GALEN project aimed at development of "a terminology server to manage language-independent shared systems of concepts for clinical applications" [3]. This project has led to development of a terminology server (ROIS), which is currently used at a number of sites. Around the same time, UMLS released the Knowledge Source Server [4], which provides both a web-based interface and an API (Application Programming Interface) to retrieve the Metathesaurus and Semantic Network information from the UMLS server by means of a number of methods. Some other terminology servers have become available since, such as DTS (Distributed Terminology Server), "the Apelon server software for viewing and/or using

published terminologies through an API" [5], and the CIC Look Up Engine (CLUE), "an API that makes it easy to use the power of Version 3 of the NHS Clinical Terms and SNOMED"[6].

Standards for Terminology Services

Besides the development of applications, definition of standardized interfaces for terminology servers has been a research issue. A variety of such standards currently exist, of which some have led to actual implementations. The first effort to come to a standard for terminology services was the Terminology Query Services (TQS) specification [7]. This specification defines an extensive API for querying a terminology server, in which thirteen modules that can be distinguished. Apart from a reference implementation, OpenEMed [8], no implementations of TQS are known to the authors.

A different approach is taken in the definition of the Terminology Query Language (TQL), "a declarative, set-based query language built on a generic entity-relationship schema"[9]. It uses SQL-like queries to retrieve data from a terminology server, which is returned in XML format. There is an Open Source implementation of TQL: jTerm[10].

Currently, another standard is being developed within the HL7 consortium, the Common Terminology Services (CTS) specification, that "identifies the common functional characteristics that an external terminology must be able to provide"[11]. A notable difference between CTS and the other specifications is that CTS does not specify how the service is to be implemented.

This non-exhaustive enumeration of standards shows that there is a great variety in proposed and implemented ways to provide terminology services. It also demonstrates how some implementations provide multiple ways of interfacing, for example the UMLS KSS, which provides both a socket-based implementation and a Java client API on top of that.

COMPONENTS

As much as the various standards and implementations differ in their choices for protocols, platforms, and paradigms, they agree on the core components that constitute a terminological system: they all have notion of a "concept representation" (concepts) and "linguistic designation" (terms). There is a range of other possible elements of a terminological system, such as a coding scheme, or context management.

Although "concept representation" and "linguistic designation" are common in all specifications for terminology services, the formalisms on which these are based vary. As a consequence, there are differences in the provided (or proposed) functionality, as some functionality is dependent on the underlying representation formalism. We will focus on the concept representation formalism and the linguistic designations to provide insight in the implications of different formalisms.

Concept Representation Formalisms

Concept representation formalisms can be categorized as those without support for universal and automic reasoning, such as frames and entity-relationship models, and those with support for automated reasoning, most notably Description Logic-based representation. In fields varying from applied science to mathematics, research is ongoing to increase the understanding and utilization of these formalisms. Similar to the research on terminology services, applications are being developed to exploit the merits that the formalisms provide, and standards are being defined. Probably the best-known example of an application based on frames-representation is Protégé[12], which provides an application for knowledge acquisition as well as an API to directly manipulate a frame-based knowledge base. An example of a standard for addressing frame-based applications is OKBC (Open Knowledge Base Connectivity)[13], which however has not been used in many applications.

Description Logics form the basis for a variety of reasoners, which over time have been capable of dealing with more and more expressive logics. FaCT[14] and RACER[15] are two examples of reasoners that support reasoning with very expressive logics. Realizing the need for increased interchangeability, the DL community has initiated definition of a web-based standard for accessing DL-based knowledge representation systems, named DIG[16], as a successor of KRSS (Knowledge Representation System Specification)[17].

Natural Language Representation

Whereas the concept representation is essential for classification and computerized manipulation of concepts by providing semantics to a concept, a natural language representation is crucial for dealing with language as used in everyday practice. It is a basis for the support of tasks such as recognition and generation of lexical variants, spelling variants, and synonyms. A major example of a system that provides such information for medical terms is the UMLS SPECIALIST Lexicon with accompanying tools[18]. A general system is WordNet, which has words "organized into synonym sets, each representing one underlying lexical concept"[19]. No standards for such systems are known to the authors.

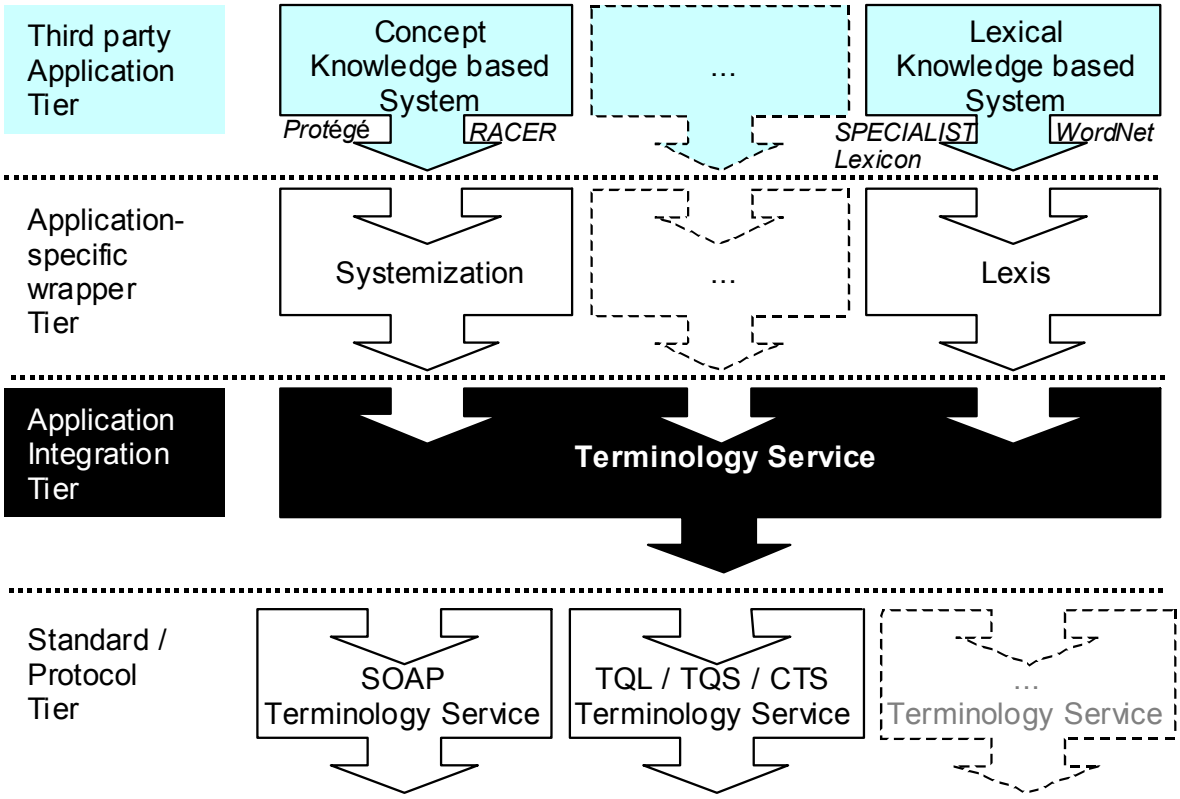

Figure 1: A Multi-tier Architecture for Terminology Services. Dedicated (third-party) applications (examples are given, printed in Italics) are wrapped to provide them with a standardized interface. Various modules are integrated into the Terminology Service, which can be equipped with a standardized, protocol specific interface.

INTEGRATED TERMINOLOGY SERVICES

In spite of the advancing expertise in concept and lexical representation, the generated knowledge and applications are rarely combined in currently existing implementations. Moreover, the introduction of new insights and methods is hampered by the necessity to implement them into existing terminological systems. We therefore define an architecture in which existing components, such as the ones mentioned earlier, can be integrated. In this way terminology services will be able to keep pace with the ongoing developments in various fields. Besides, the use of already existing concept and lexical models will be easier, if the related application can be integrated. The architecture, depicted in Figure 1, defines four layers.

Third-Party Applications. These are both the concept knowledge representation systems, such as Protégé, FaCT or RACER, and the lexical knowledge representation systems, such as the UMLS SPECIALIST Lexicon.

Application Wrappers. In order to be able to integrate a range of applications into a terminology service, interfaces are defined to provide the third-party applications with a standardized interface. Examples of methods involving concept representation are "checkSubsumption" and "getParents". Examples of lexical queries are "getPreferredTerm" and "normalizeString".

Application Integration. The terminology service brings together functionality and data from the various applications, interrelating them. For example, terms are related to concepts, and codes can be provided for concepts. More advanced functionality is described in more detail below.

Standards and Protocols. As the process of defining standards for terminology services is still ongoing, standards are separated from the integration tier, so that a terminology service can be delivered with one or more appropriate standards and protocols.

Interfaces

The divergent functionality of similar systems is a complicating factor in our approach, but inevitably needs to be overcome. For example, a Frame-based knowledge representation system offers different functionality than a Description Logic-based system, such as interrogation of frames to retrieve their properties, versus inferred classification for DL-

based concepts. Two measures are taken to deal with these disparities. First, a mechanism that is also used in TQS is adopted, where methods defined in the wrappers can throw a "not implemented" exception. Hence, a relatively extensive set of methods can be defined in the interfaces of the wrappers, without the necessity that third-party applications actually provide all functionality.
On the other hand it is likely that third-party applications will provide functionality that is useful to some clients, although it is not part of the common functionality of a terminology service. To address these native functions, applications send a "native" request to the terminology service, specifying the requested application, method, and parameters. The terminology service then delegates this request to the appropriate application-specific wrapper, in which the method is defined. In this way, flexibility is provided, but on the cost of not allowing complex data types for parameters and return values.

Extended Terminology Services

This architecture not only aims at integration of terminology services, but also at extending the functionality. An example of such an extension is a method comparable to the TQS method 'get_entity_graph', that returns a graph representation of a part of the hierarchy held in the concept knowledge representation system. Whereas for example Protégé provides a "genHierarchy" method to provide this kind of functionality, for other systems it might be necessary to recursively iterate subclasses to generate a representation of the hierarchy. The terminology service can provide this functionality either by using the first method of the wrapper (comparable to genHierarchy) or, if this method is not implemented in a wrapper, by recursive retrieval of subclasses.

REFERENCE IMPLEMENTATION

Currently, the implementation of the described architecture is ongoing. We focus on implementation of a SOAP-based interface on a Java terminology server, providing services in a web-based fashion. Wrappers are implemented as Java classes that implement the proper interfaces. For the purpose of demonstration, and in order to ease migration of DICE to this new architecture, we are implementing wrappers for the concept and lexical knowledge base of DICE. Besides, we will develop a wrapper for the Description Logic reasoner RACER [15] that will be used to enhance reasoning capabilities of the terminology service. This process not only aims at actual implementation, but also as a proof of concept of this architecture, and as a means for furthering the specification of the interfaces of the wrappers and the terminology service itself.
Figure 2 shows an example of the flow of requests being performed by the terminology service. A client sends a SOAP request to the terminology service, which delegates various parts of the request. It first retrieves a tree representation from the Protégé wrapper, which queries Protégé for a part of the hierarchy. The received concept identifiers are mapped to KSS LUIs (Lexical Unique IDs) by means of an internal mapping table. These LUIs are sent to KSS to retrieve the terms from the UMLS Knowledge Source Server.

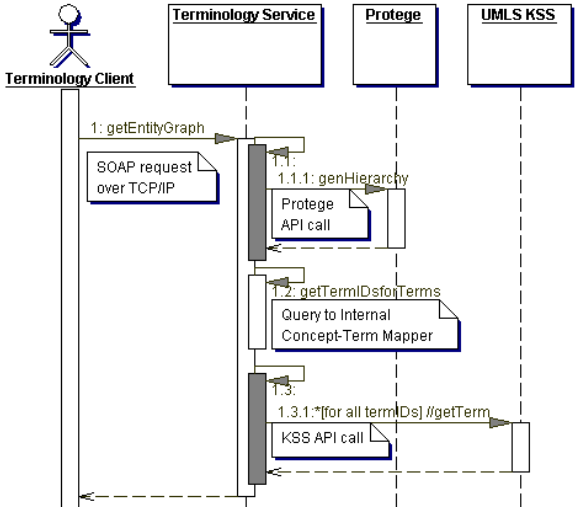

Figure 2: UML Sequence Diagram representing interaction between Client, Server and Third-party Applications (Protégé and UMLS Knowledge Source Server). Wrappers are shown in gray. The SOAP interface to the terminology service is not represented

FURTHER RESEARCH

The focus for the architecture has been on querying the terminology service, as is the focus of the currently existing standards. We foresee that modeling can also be made possible through the architecture by extending the interface. An important issue to be addressed when loosely coupling terminological resources is consistency, as the modification of one knowledge base (e.g. concept knowledge) may require an update of any related knowledge base (e.g. lexical knowledge).
Another issue that will be explored further is the standardization of components for clients. Currently, development of a client requires implementation of functionality that will largely be common, but of which the presentation is depending on the platform and paradigm of the client application. To make development of terminology clients easier, methods that represent data in a way closely related to the

required presentation will be implemented in the terminology service.

CONCLUSIONS

We suggest a new approach in providing terminology services. Current efforts have been focusing either on development of complete terminology solutions, or only on parts of the constituent elements of terminology services. As the latter have come up with solutions that are thoroughly studied in the fields from which they originate, we strive at employing their individual strengths. Three issues have been addressed to realize this. The first issue is the use of wrappers in order to standardize the communication with dedicated (third-party) applications, overcoming the lack of implemented standards for such systems. The second issue is that of integration, both of data and functionality of the concept and the lexical knowledge based system. Thirdly, pending the development of a commonly accepted standardized interface for terminology services, multiple interfaces can be defined, for protocols such as SOAP or CORBA.

Work on a reference implementation is ongoing, demonstrating the feasibility of the architecture. This implementation provides a SOAP interface to a Java implementation of the terminology service. Wrappers for Protégé and RACER will enable use of both a frame-based and Description Logic based concept knowledge representation system. Wrappers for the in-house developed DICE system will increase the usability of DICE for various purposes. Overall, this architecture will increase interchangeability of knowledge-based systems, third-party applications and terminology clients.

Acknowledgements

This research is supported by the NICE Foundation and the Dutch Ministry of Health, Welfare and Sport.

References

1. de Keizer NF, Abu-Hanna A, Cornet R. Evaluation of DICE, a terminological system for intensive care. Stud Health Technol Inform 2000;77:208-12.
2. de Keizer NF, Stoutenbeek CP, de Jonge E, Timmers T, Zwetsloot-Schonk JHM. An intensive care diagnosis classification supporting the care process and quality assurance. In: MedInfo 1998; 1998; Seoul, Korea; 1998.
3. Rector AL, Solomon WD, Nowlan WA, Rush TW, Zanstra PE, Claassen WM. A Terminology Server for medical language and medical information systems. Methods Inf Med 1995;34(1-2):147-57.
4. McCray AT, Razi AM, Bangalore AK, Browne AC, Stavri PZ. The UMLS Knowledge Source Server: a versatile Internet-based research tool. Proc AMIA Annu Fall Symp 1996:164-8.
5. Apelon website, http://www.apelon.com/, Last Accessed: 2003, June 27th
6. CLUE website, http://www.clinical-info.co.uk/clue.htm, Last Accessed: 2003, June 27th
7. OMG. *Lexicon Query Service Specification*: Object Management Group; 2000 july 2000. Report No.: 00-06-31.pdf.
8. OpenEMed website, http://openemed.net/, Last Accessed: 2003, June 27th
9. Hogarth MA, Gertz M, Gorin FA. Terminology Query Language: a server interface for concept-oriented terminology systems. Proc AMIA Symp 2000:349-53.
10. jTerm website, http://termserver.cs.ucdavis.edu/jterm/index.html, Last Accessed: 2003, June 27th
11. CTS Specification Version 0.8, http://www.hl7.org/Library/Committees/vocab/CTSSpecv08.zip, Last Accessed: 2003, February 24th
12. Gennari JH, Musen MA, Fergerson RW, Grosso WE, Crubézy M, Eriksson H, et al. The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. International Journal of Human-Computer Interaction 2002;58(1):89-123.
13. OKBC website, http://www.ai.sri.com/~okbc/, Last Accessed: 2003, June 27th
14. Horrocks I, Sattler U, Tobies S. Practical reasoning for very expressive description logics. Logic Journal of the IGPL 2000;8(3):239-63.
15. Haarslev V, Möller R. High Performance Reasoning with Very Large Knowledge Bases. In: International Workshop in Description Logics 2000 (DL2000); 2000; Aachen, Germany; 2000.
16. Bechhofer S. The DIG Description Logic Interface: DIG/1.0. Manchester: University of Manchester; 2002 October 1st, 2002.
17. Patel-Schneider P, Swartout B. Description-Logic Knowledge Representation System Specification from the KRSS Group of the ARPA Knowledge Sharing Effort; 1993 1 november 1993.
18. McCray AT. The nature of lexical knowledge. Methods Inf Med 1998;37(4-5):353-60.
19. WordNet website, http://www.cogsci.princeton.edu/~wn/, Last Accessed: 2003, June 27th