Concept Maps: Theory, Methodology, Technology
Proc. of the Second Int. Conference on Concept Mapping
A. J. Cañas, J. D. Novak, Eds.
San José, Costa Rica 2006

KEA: A KNOWLEDGE EXCHANGE ARCHITECTURE BASED ON WEB SERVICES, CONCEPT MAPS AND CMAPTOOLS

Alberto J. Cañas, Greg Hill, Larry Bunch, Roger Carff, Thomas Eskridge, Carlos Pérez
Florida Institute for Human and Machine Cognition (IHMC), USA
www.ihmc.us

Abstract. The CmapTools program, and concept mapping in general, have seen a steady growth in usage in the last few years. With new users come new requirements and needs for the software that we can't always satisfy. Until now, the CmapTools architecture has been relatively closed to third party developers. In this paper we report on a new open architecture, KE (Knowledge Exchange Architecture), that provides the data and procedural framework to allow third party developers to design programs that interact with the CmapTools suite of programs. KEA consists of a portable, standard file format for data exchange (CXL), and a set of protocols for retrieving, storing, and manipulating Cmap information. We also describe how KEA was implemented and provide three sample client-server applications illustrating the steps we have taken toward the completion of this implementation.

1 Introduction

The CmapTools program (Cañas et al., 2004) was designed to be a simple to use but powerful environment that facilitates collaboration and sharing during the construction of concept map-based knowledge models (we consider a knowledge model to be a set of concept maps (Cmaps) and associated resources about a particular domain of knowledge (Cañas, Hill, & Lott, 2003). The software was designed to have a low threshold, making it easy to learn for new and naïve users, but at the same time to have a high ceiling, enabling sophisticated and expert users to develop large, complex knowledge models.

The use of CmapTools is experiencing rapid growth worldwide (see http://pictor.ihmc.us/gl for a live world map of CmapTools installations). Users are constantly discovering innovative ways of doing concept mapping and using the CmapTools software. The constant growth in the use and usages of CmapTools leads to needs for new ways to manipulate Cmaps. Some of these needs are very specific to a small group of users, or for a variety of

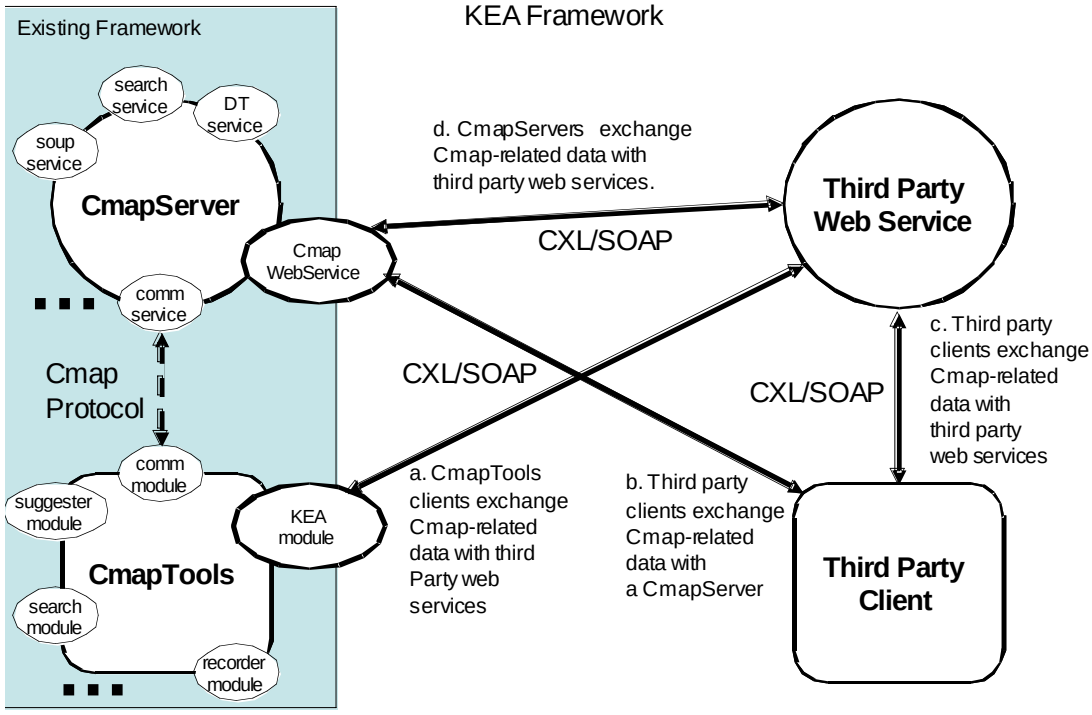

Figure 1: KEA opens the CmapTools architecture and extends the available pathways of Cmap data exchange.

reasons are beyond the scope of what we have chosen to implement in CmapTools.

To meet the needs to extend the functionality of CmapTools without continually integrating new features directly into our CmapTools software, we must go beyond the current, relatively closed architecture that is depicted by the shaded area on the left side of Figure 1. Here the data on the CmapServer is only accessible to CmapTools clients. Further, the only way to extend the application involves creating either a new 'module' in the CmapTools client or a new 'service' in the CmapServer (or both). Such development requires detailed knowledge of the inner workings of the CmapTools software.

In this paper we report on a new open architecture, KEA[1] (Knowledge Exchange Architecture), that provides the data and procedural framework to allow third party developers to design programs that interact with the CmapTools suite of programs. KEA consists of a portable, standard file format for data exchange (CXL), and a set of protocols for retrieving, storing, and manipulating Cmap information. Figure 1 illustrates the new paths of information exchange that have been made available by adapting the CmapTools client and the CmapServer to this new framework which is based on the Web Services Architecture. By adding a web services-aware module to the CmapTools client, we will enable end users to configure CmapTools to exchange Cmap related information (Cmaps, proposition lists, etc.) with third party web services designed specifically for manipulating Cmap data, as illustrated by path 'a'. By extending the functionality of the CmapServer into a web service, third party clients can now access and store CmapServer resources that were previously available only to CmapTools clients, as depicted by path 'b'. By adopting a standard open format (CXL) for Cmaps, we hope to promote the exchange of concept mapping information between third party developers, as depicted by path 'c'. And finally, the CmapServer may itself desire to exchange Cmap information with a third party web service, or a third party web service might want to request information from a CmapServer, as seen in path 'd'.

In the remainder of this paper we describe how KEA was implemented and we provide three sample client-server applications illustrating the steps we have taken toward the completion of this implementation.

2 How CmapTools Supports Open Knowledge Exchange in KEA

The open exchange of Cmaps and knowledge models requires (1) establishing open standards for the format of Cmap data and (2) defining standard protocols for client-server interaction. In both of these tasks, we have adopted the recommendations of the World Wide Web Consortium (W3C) and their emerging standards for a Web Services Architecture (Booth et al., 2004). KEA extends both the CmapTools client and the CmapServer to support such standards, thereby allowing the CmapTools client to access third party web services and allowing third party clients to access the CmapServer.

Establishing an open data format makes it possible for any client or server application to read and write the full textual, graphical, and relational content of Cmaps. We selected the "Extensible Markup Language" (XML) and defined the "Concept Mapping Extensible Language" (CXL) dialect specifically for describing the contents of Cmaps. Section 3 provides a detailed description of the CXL format. XML is a widely adopted standard maintained by the W3C (Bray et al., 2000) with multiple benefits as a representation for Cmaps. The text-based markup is independent from any programming language or operating system. It can be understood and authored by people and also parsed and generated by multiple third party software tools, making this format accessible to developers. The extensibility of the language allows for an explicit representation of concept mapping constructs (e.g. concept, linking phrase, resource link) and further enables other researchers and developers to extend this data structure without compromising compatibility.

Establishing standard interaction protocols makes it possible for any online client or server application to invoke operations on another application and receive the results. We have implemented this interoperability as web services using the Web Services Description Language (WSDL) (Chinnici et al., 2006) and the Simple Object Access Protocol (SOAP) (Gudgin et al., 2003) standards. These web services are based on XML and benefit from platform and language independence, exchanging Cmap data in the CXL format. WSDL and SOAP are emerging standards with several benefits to support the KEA interoperability goals. Web Services are self-describing through

[1] All the information needed to implement these new services is available at the CmapTools website (http://cmap.ihmc.us).

the WSDL descriptions of inputs, outputs, and bindings for each service call. This enables clients to find desired services through third party discovery mechanisms (e.g. UDDI) and to bind to such services, often very easily using one of several code generation tools that support these standards. Section 4 provides a detailed description of the web service applications we created by extending the CmapTools client and the CmapServer in order to integrate them into this more open and standardized web services framework.

3 CXL – Concept Mapping Extensible Language

CXL is a publicly available XML-based language for describing the content of Cmaps. Implementing this open and extendable representation in CmapTools is the first step toward establishing CXL as an international standard for the storage of concept maps as well as the mechanism for exchanging concept maps between programs. All concept mapping users and researchers are strongly encouraged to adopt this format, contribute to its maturation, and extend the language in their own useful and interesting ways. A brief introduction to CXL follows, in order to familiarize readers with its most basic features A full specification of the language is available at http://cmap.ihmc.us.

The CXL language is written in plain text such that users with some knowledge of the format can simply type a description of a Cmap in any text editor and CmapTools will be able to import and display the map. Similarly, any application can be enhanced to read and write files in this format. Figure 2 contains an example of a Cmap in CXL containing the proposition “Plants have Leaves”.

There are four basic sections in a CXL data file. The first section defines the XML document type and any name space mappings. The second section, specified as a ‘res-meta’ element, contains resource level information about the Cmap such as the title, description, and author. The third section is the Cmap Data section (the second non-shaded region in Figure 2) which contains lists of concepts and linking phrases and the list of connections that link them together. The final section, Cmap Appearance, defines graphical information, such as, location, size, colors, and fonts for entities defined in the Cmap Data section. Note that the gray sections of Figure 2 are not required for non-graphical representations of a Cmap.

```
<?xml version="1.0" encoding="UTF-8"?>
<cmap xmlns="http://cmap.ihmc.us/xml/cmap.dtd"
      xmlns:dc="http://purl.org/dc/elements/1.1/">

    <res-meta>
        <dc:title>Plants</dc:title>
        <dc:description>How do plants grow?</dc:description>
    </res-meta>

    <map>
        <concept-list>
            <concept id="1" label="Plants"/>
            <concept id="2" label="Leaves"/>
        </concept-list>
        <linking-phrase-list>
            <linking-phrase id="3" label="have"/>
        </linking-phrase-list>
        <connection-list>
            <connection from-id="1" to-id="3"/>
            <connection from-id="3" to-id="2"/>
        </connection-list>

        <concept-appearance-list>
            <concept-appearance id="1" x="73" y="56"/>
            <concept-appearance id="2" x="136" y="158"/>
        </concept-appearance-list>
        <linking-phrase-appearance-list>
            <linking-phrase-appearance id="3" x="104" y="107"/>
        </linking-phrase-appearance-list>
    </map>
</cmap>
```

Figure 2: A Simple CXL Example of “Plants have Leaves”

As simple as the example in Figure 2 is, CXL can represent all the additional characteristics (links to resources, images, nested nodes, etc.) that can be added to a Cmap in CmapTools. More extensive information about the CXL format including how to specify resource links, style sheets, and propositions can be found at the CmapTools website (http://cmap.ihmc.us).

4 Implementation and Examples

As was mentioned in Section 2, we chose to implement KEA within a web services framework for achieving interoperability between CmapTools software and third party software. Doing so required that we integrate both the CmapTools client and the CmapServer into the web services architecture. To demonstrate the utility of our approach, we developed three examples of client-server applications.

Figure 3 illustrates both the specific components that we designed for our implementation and some standard components that one finds in many applications based on web services. We will first describe the standard components so that we can refer to them when describing our three example applications in detail. The standard components that play a role in all three applications are:

- The WSDL (Web Services Description Language) document. There is one WSDL document per web service. It is written in XML, and contains a description of the operations (and their inputs and outputs) provided by that web service. In order to interact with a web service, a client need only understand the interface described within the WSDL. The programming language in which the client is implemented is independent of the programming language of the web service. Their 'lingua franca' is XML.
- The XML schemas. There will normally be one or more of these that are referred to by the WSDL document for the purpose of defining the data types of each operation's inputs (to be supplied by the client) and outputs (to be returned by the web service).
- The 'stubs'. Each client application typically employs a small piece of code generated by the client developer in the language of his choice, which acts a proxy to the web service. There is typically a one-to-one relationship between the operations provided by the proxy stub and those provided by the web service. As a proxy, the stub is responsible for preparing the client request for transport to the web service, receiving the web service response, and returning the response to the client.
- The 'stub generators'. Many web service application frameworks (e.g. Axis2, JAX-WS, Microsoft .NET) provide tools for parsing a WSDL document and generating the corresponding proxy stub for the client application developer.

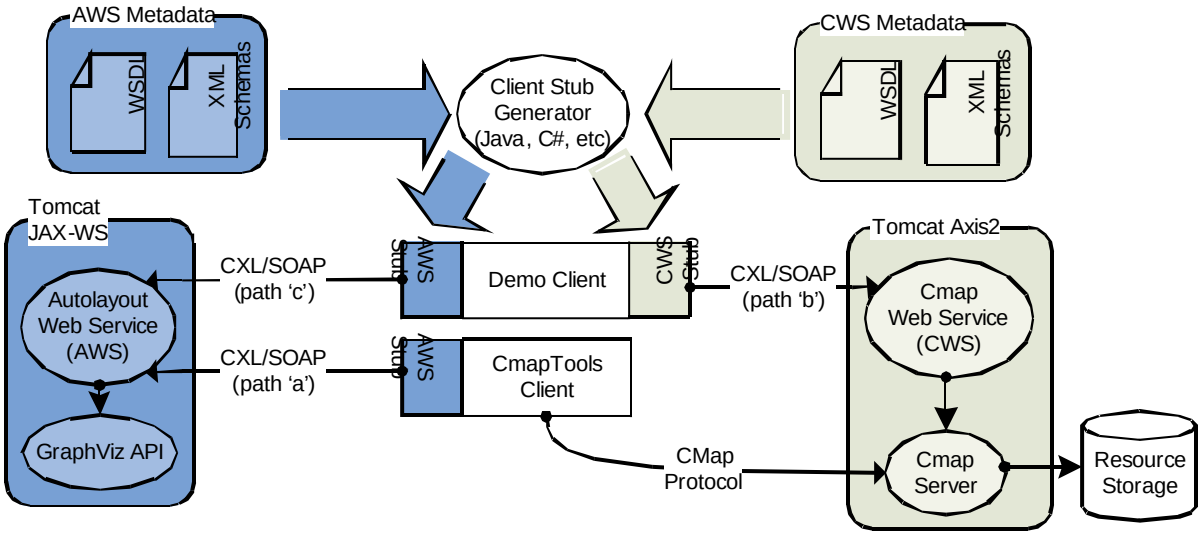

Figure 3: KEA Implementation Components.

We now point out how the three applications we developed fulfill the goals for various pathways of data exchange depicted in Figure 1. The first example will illustrate the steps that we have taken toward fulfilling the goal of implementing path 'a'. We developed a third party web service called the Autolayout Web Service (AWS), which takes a Cmap as input and returns a new Cmap that differs from the original solely in the manner in which it can be graphically displayed. This required an additional layer of functionality to be added to the CmapTools client (the AWS-interface module) that enables the client to interoperate with the AWS. The second example illustrates the implementation of path 'b'. We encapsulated a few of the most fundamental operations of the CmapServer into a web service called the Cmap Web Service. We then developed an open source reference implementation called the "Demo Client" to demonstrate how a third party client can now use web services to exchange Cmap related data with the CWS. The third and final example will show how we implemented path 'c' , namely a third party client exchanging Cmap related data with a third party web service. The various components involved in achieving the implementation of paths 'a', 'b', and 'c' are labeled as such in Figure 3. We will now proceed to explain the three applications in more detail.

Our first example, the Autolayout Web Service, is portrayed in the left hand portion of Figure 3. The AWS was designed to provide CmapTools clients with access to the latest and most advanced automated graphical layout algorithms, to enable users to better view, organize, and understand their Cmaps. Figure 4 shows an example concept map and several autolayout variants generated with AWS. Figure 4a shows a hierarchical layout, 4b a circular layout, and 4c a force-directed hierarchical layout.

We selected the GraphViz (www.graphviz.org) library of autolayout algorithms developed by AT&T (Gansner & North, 2000) as the basis for the AWS. This library is open source, written in C and C++, and is actively developed. The web services framework provides an ideal way to integrate the functionality of GraphViz with CmapTools. We abstracted the details of the programming language in which GraphViz was implemented by encapsulating the GraphViz API into the AWS. To achieve the encapsulation, we wrote a thin server-side layer that provides for bidirectional language and data translation between GraphViz (C and C++ code) and CXL. We generated a WSDL document describing the operations that we wanted to advertise to clients, and from this we generated a client stub in Java. The client stub was then encapsulated within a new CmapTools module called the

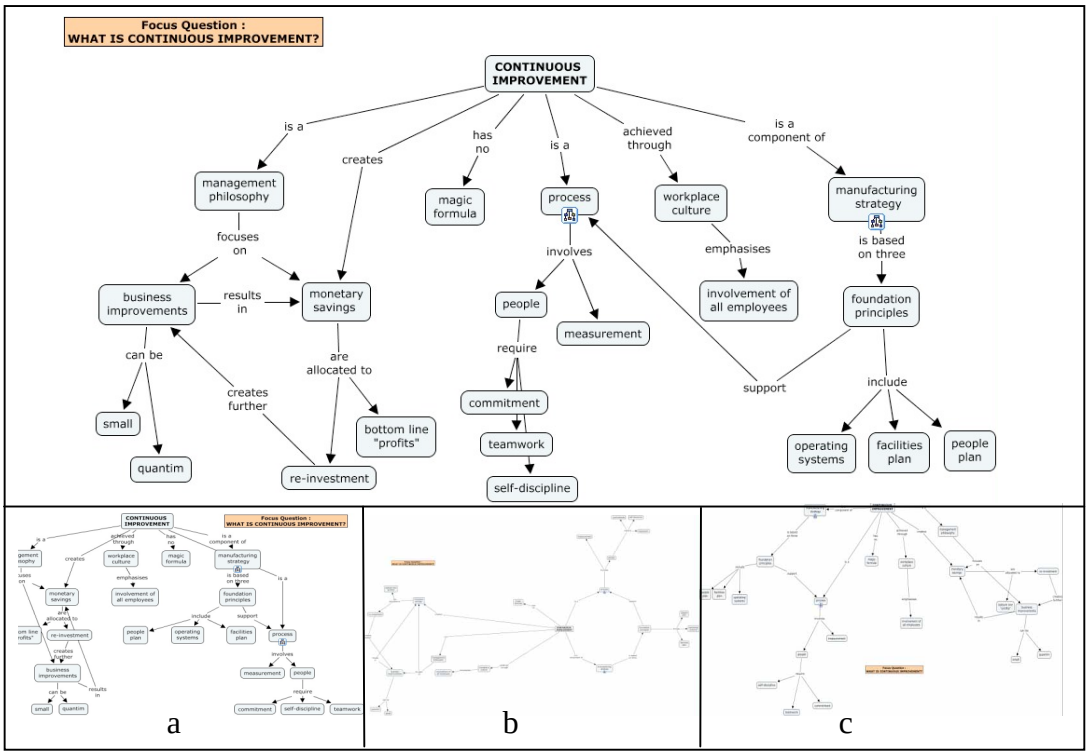

Figure 4. An example Concept Map and results from AWS.

AWS-interface Module.

The data flow between the CmapTools client and the AWS can be described as follows: The CmapTools client posts an event containing a Java-specific representation of a Cmap to the AWS-interface module. The AWS-interface module then translates this Cmap into the standardized CXL format and transports it via the SOAP protocol to the AWS. The AWS receives the CXL formatted Cmap, translates it into the GraphViz "DOT" data file format and passes it to the GraphViz API. GraphViz creates a newly laid-out Cmap as a DOT file, and the AWS translates it back into CXL. The CXL is then sent back to the client via the SOAP protocol. The AWS-interface module receives the new version of the Cmap in CXL format and translates it back into the CmapTools internal Java format.

The second example demonstrates how KEA supports Cmap-related data exchange between a third party client (the Demo Client) and the Cmap Web Service (CWS). As with the GraphViz API, the CmapServer may be viewed as an API that provides a rich set of operations for any Java developer that has access to the (proprietary) Java class definitions. Opening up this architecture entails encapsulating the CmapServer API into a web service. This was done in a manner almost identical to the means by which we encapsulated the GraphViz API. We defined a set of web service operations for the Cmap Web Service, and we designed a thin layer to direct requests and responses back and forth between the web service framework and the CmapServer API. The requests that we focused on (the operations that we wanted to export as a web service) were: (1) accessing a folder table of contents; (2) requesting a resource from the CmapServer, (3) saving a resource to the CmapServer. Once the operations were defined, we once again generated a WSDL document, accompanied by references to the appropriate XML Schemas.

We wrote the Demo Client in C# and we used the Microsoft .NET platform to generate the "CWS stub" in C#. The CWS stub is shown is shown in Figure 3 on the right hand side of the Demo Client component. The Demo Client was designed with a simple Graphical User Interface (GUI) that displays the contents of the root folder of a CmapServer to the end user. The user can then click on any other folder displayed within this hierarchy in order to see that folder's table of contents. Likewise, clicking on a resource within a folder will display the resource. In this example, the data flow would be the following: Requests made in C# are translated by the client stub into XML, which is then transported as a SOAP request to the Cmap Web Service. This service receives the request as XML and then translates it into the native Java format parseable by the CmapServer API. The CmapServer returns the requested resource (or folder table of contents) which is then translated from Java to XML by the Cmap Web Service, which in turn uses the web services framework to transport the XML data back over SOAP to the client. The client stub translates the XML back into C# and returns it to the caller. As a side note, we deployed the Axis2 web service framework along with the CmapServer into a single web application running in the VM of the Tomcat Web Server. Running all the server side code within the same Java VM made the encapsulation process a very simple one.

The third and final example demonstrates a third party client exchanging Cmaps and Cmap related data with a third party web service. In this example, the Demo Client interacts with the Autolayout Web Service. The Demo Client can retrieve a Cmap from the Cmap Web Service, send it to the Autolayout Web Service, and receive the same basic Cmap in a new graphical layout. Since the Demo Client is now also interacting with Autolayout Web Service in this example, it must employ a stub that was generated from the WSDL document specific to the Autolayout Web Service. This stub is shown on the left hand side of the Demo Client component in Figure 3.

5 Conclusions

Previously, the CmapTools architecture was relatively closed to third party developers. The only way to integrate third party code with the CmapTools client was to design a new CmapTools module, written in Java. Such a task is quite difficult, due to the prior knowledge of relevant events that the programmer must be aware of. The web services architecture described here provides a means by which previously proprietary protocols and formats may be standardized and opened up to the public, paving the way for Cmap researchers and developers worldwide to collectively create and share new and exciting capabilities in their applications.

6 References

Booth, D., Haas, H., McCabe, F, Newcomer, E., Champion, M., Ferris, C., Orchard, D. *Web Services Architecture*, W3C Working Group Note, 11 February 2004. (http://www.w3.org/TR/ws-arch/).

Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation, 6 October 2000. (http://www.w3.org/TR/2000/REC-xml-20001006/).

Chinnici, R., Moreau, J., Ryman, A., Weerawarana, S., *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, W3C Candidate Recommendation, 27 March 2006. (http://www.w3.org/TR/wsdl20/)

Cañas, A. J., Hill, G., Carff, R., Suri, N., Lott, J., Eskridge, T., et al. (2004). CmapTools: A Knowledge Modeling and Sharing Environment. In A. J. Cañas, J. D. Novak & F. M. González (Eds.), *Concept Maps: Theory, Methodology, Technology. Proceedings of the First International Conference on Concept Mapping* (Vol. I, pp. 125-133). Pamplona, Spain: Universidad Pública de Navarra.

Cañas, A. J., Hill, G., Granados, A., Pérez, C., & Pérez, J. D. (2003). *The Network Architecture of CmapTools* (Technical Report No. IHMC CmapTools 2003-01). Pensacola, FL: Institute for Human and Machine Cognition.

Cañas, A. J., Hill, G., & Lott, J. (2003). *Support for Constructing Knowledge Models in CmapTools* (Technical Report No. IHMC CmapTools 2003-02). Pensacola, FL: Institute for Human and Machine Cognition.

Eskridge, T., Hayes, P., Hoffman, R., & Warren, M. (2006). Formalizing the Informal: A Confluence of Concept Mapping and the Semantic Web. In A. J. Cañas & J. D. Novak (Eds.), *Concept Maps: Theory, Methodology, Technology. Proceedings of the Second International Conference on Concept Mapping* (Vol. 1). San Jose, Costa Rica: Universidad de Costa Rica.

Gansner, E. R., & North, S. C. (2000). An open graph visualization system and its applications to software engineering. *Software -- Practice and Experience, 30*(11), 1203-1233.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., Nielsen, H.F., *SOAP Version 1.2 Part 1: Messaging Framework*, W3C Recommendation, 24 June 2003. (http://www.w3.org/TR/2003/REC-soap12-part1-20030624/)