CGWorld-2001 - New Features and New Directions

Pavlin Dobrev[1], Albena Strupchanska[2] and Kristina Toutanova[3]

[1]ProSyst Bulgaria Ltd., Sofia, Bulgaria
pavlin@prosyst.com
[2]Linguistic Modeling Lab, CLPP, Bulgarian Academy of Sciences, Sofia, Bulgaria
albena@lml.bas.bg
[3]Stanford University, Department of Computer Science, Stanford, CA, USA
kristina@cs.stanford.edu

Abstract. This article summarizes the authors' two years experience in implementing CGWorld - a web based workbench for joint distributed development of a knowledge base of conceptual graphs, which resides on a central server. The paper discusses the new component-based architecture of the workbench. The major development during the last year was the integration of operations on Conceptual Graphs. The support of the CGIF [3] format was further improved and some new features were added.

1 Introduction

CGWorld was first introduced at ICCS 2000 [1]. The main motivation for creating CGWorld was the need of an application that allowed Internet access to a knowledge base of Conceptual Graphs (CG). The goal was to provide various facilities for remote browsing and editing of a KB that resides on a central server.

Support of different representation formats for CGs was also a high priority. We chose the graphical representation of conceptual graphs as the major medium for browsing, editing and manipulation of the knowledge base since it is easier to use by even non CG-expert knowledge engineers and end users. The other supported formats were CGIF, First Order Logic and a Prolog format.

The initial version CGWorld met many of the needs that motivated its creation. It had excellent browsing, searching and editing features for a KB of CGs. This paper describes several enhancements to the initial version, which make CGWorld a more robust and complete solution as a CG Tool. Firstly, the architecture of CGWorld was changed according to the latest developments in the area of multi tier web applications. The new component based architecture makes the integration of new and previously developed applications and features in CGWorld quicker and more elegant. Secondly, inference rules for conceptual graphs were integrated into CGWorld. The operations integrated are join, generalization, specialization, projection, type extraction and type contraction. The user can request for operations and specify their arguments through a user-friendly interface. The resulting CGs are visualized in display form.

The paper is organized as follows: section 2 mentions briefly the software base used for the implementation of CGWorld. Section 3 describes the new component based architecture in the enhanced version of the tool. It also motivates our choices and explores the advantages and disadvantages of the two architectural decisions. Section 4 concentrates on the different representation formats supported by CGWorld and the interfaces for visualization of CGs in different formats. Section 5 describes the implementation of CG operations; in section 6 we conclude and share our plans for future work on CGWorld.

2 Implementation

CGWorld is a typical web application. It uses a Browser as a presentation layer. Currently, CGWorld is implemented using HTML, Java, JavaBeans, Java Server Pages (JSP), and Prolog. JavaBeans is component

architecture for the Java programming language. JSP is a Java-based technology that simplifies the process of developing dynamic web sites. With JSP, web designers and developers can quickly incorporate dynamic elements into web pages using embedded Java and a few simple markup tags.

In our implementation we use the following additional software:

- JDK 1.3 from Sun Microsystems (http://java.sun.com).
- SICStus Prolog from the Swedish Institute of Computer Science (http://www.sics.se/sicstus). It allows for easy integration with Java.
- Tomcat - a servlet container with a JSP environment. A servlet container is a runtime shell that manages and invokes servlets on behalf of users. Tomcat was developed by the Apache Software Foundation as part of the open source Jakarta Project (http://jakarta.apache.org).

3 Present Architecture

In this section we will describe the architectural options we explored when developing CGWorld over the last two years.

Initially, as discussed in [1], we developed our own web server that allowed for the inclusion of additional components that extended functionality. Different kinds of browsing and editing of the CG knowledge base and synchronization with SICStus Prolog were developed as additional components.

The benefits of building our own Java Server were twofold. Firstly, we needed a fast implementation. Secondly, we already had developed several Prolog applications, which we wanted to make accessible through the web and the relevant decision was to reuse all previously developed programming code. The first version of CGWorld that came out in 1999-2000 was built upon this architecture [1].

A problem with this architecture arose when we wanted to further improve its functionality and implement additional features. There was a substantial overhead in developing additional components, because they were not standard and therefore a lot of time had to be dedicated to studying the interfaces the component had to implement and how to deploy the component in order to develop it.

In the last two years Java has acquired a very good reputation for development of server applications. Many Java specifications for component architectures (JavaBean and Enterprise JavaBeans) and web access to component based applications (Servlet and Java Server Pages) were developed. In accordance with the current Java technologies the new version of CGWorld uses one of the most widely distributed reference implementations of the Servlet and JSP specifications - Tomcat.

When designing a web application of any complexity, it is helpful to think of its high-level architecture in terms of three logical areas:

- The *presentation layer*, the front end which controls the look and feel and delivers results, also known as the view
- The *control layer*, which controls application flow also known as the controller
- The *application logic* layer, which manages application data, performs calculations and communicates with back-end resources, also known as the model

The three layers help us to understand our application's requirements. According to design patterns (a collection of common strategies used in software development) it is easy to recognize this three-part architecture as an implementation of the Model-View-Controller, or MVC, pattern. The MVC pattern is concerned with separating the information (the model) from its presentation (the view).

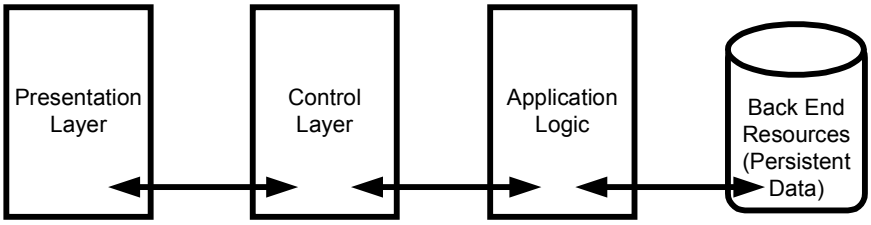

Figure 1. Web application layers

CGWorld is a typical implementation of this web architecture.

3.1 *Presentation Layer*

This tier includes the client side display elements. In CGWorld it is implemented using HTML and Java Applets. Most of the HTML pages are dynamically generated using the JSP technology. As described in [1], we implemented a CG Editor and a viewer of the Type Hierarchy as Java applets. We also implemented applets for visualization of CGs.

3.2 *Application Logic*

The application logic layer is the heart of the application, responsible for actually doing whatever it is the application is supposed to do. Currently, the application logic is implemented as a set of JavaBeans and Java classes that use an object-oriented model of the knowledge base [1] and the Prolog predicates.

3.3 *Control Layer*

The control layer determines the application flow, serving as an intermediary between the presentation layer and the application logic. This tier serves as the logical connection between the user interaction with the front-end and the application services at the back end. Currently, the control layer is implemented as a set of JavaBeans that interact with the application logic. Every conceptual graphs operation has a corresponding JavaBean that interacts with the JSP and the Prolog realization of the operation.

3.4 *Web Application Flow*

Applications, no matter the platform, are designed with a particular flow in mind. Unlike traditional applications, web-based programs are forced to deal with strange interruptions that may occur in the expected flow of a program. The user can hit the back button on the browser, hit reload, abort an in-process request, or open new browser windows at any time.

Because in CGWorld the control layer is implemented as a set of JavaBeans this automatically solves many of these problems. JavaBeans use session scope, which means in terms of a JSP specification that every user has their own instance of the bean. A user is defined as an instance of a browser somewhere in the Internet. This set of bean instances cash the user input and interact with the user and the application logic. Tomcat automatically manages the life cycle of each bean instance. When the first request from a user is received, a bean instance is created and is later reused for every subsequent request. If there are no user requests for a long period of time the server automatically destroys the bean instances.

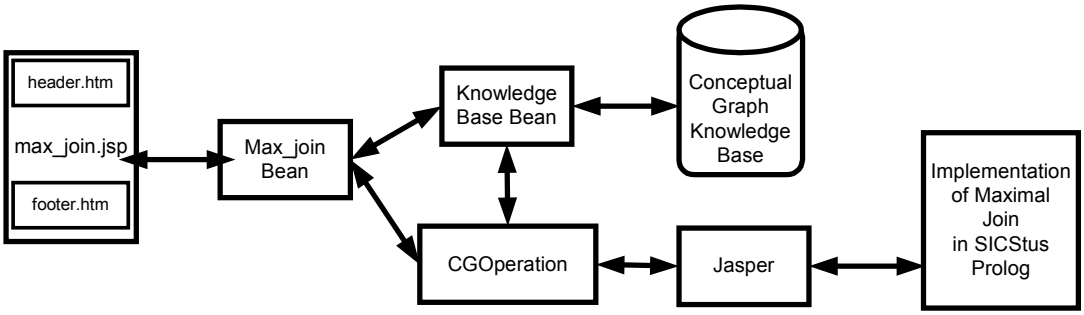

Figure 2. Implementation of Maximal Join Operation

As an example of the architecture used for implementation of CG operations, Figure 2 illustrates the Maximal Join operation. The user fills the data into a browser and sends it to the server. The data is transmitted by *max_join.jsp* to the *Max_join Bean*. If the data is not correct or not complete the user must reenter the needed data. If it is correct the maximal join operation is performed and Java receives the result through Jasper. After synchronization with the Java representation of the CG knowledge base the result is returned to the *Max_join Bean*. This result is used to generate HTML by *max_join.jsp* using the given template. This can be used as a pattern for implementation of all CG operations. Detailed explanation of the algorithms used for realization of CG operations in Prolog can be found in section 5.

4 Supporting Different Representation Formats of Conceptual Graphs

The representation forms of CGs that CGWorld supports were extensively described in [1]. These are a Prolog format, FOL (First Order Logic) format, CGIF, display form, and a Java object-oriented model (cf. Figure 3). There are also utilities for translation between the formats.

The Prolog format had existed before and was used in several CG applications [2,4,5,7]. As described in [1] on top of the Prolog representation, a Java object-oriented model of the KB was built. Using Jasper (an interface between SICStus Prolog and Java) synchronization between the Java and Prolog representations was developed. Translation to First Order Logic (FOL) was developed in Prolog. Using the Conceptual Graphs Editor of CGWorld [1] it is possible to edit CGs in display form. More information about the editor can be found in "CGWorld Editor – User's Guide" http://larflast.bas.bg:8080/CGWorld/Users_Guide.pdf. The CGs in the KB can be viewed in display form also. Translation to CGIF was implemented in Java.

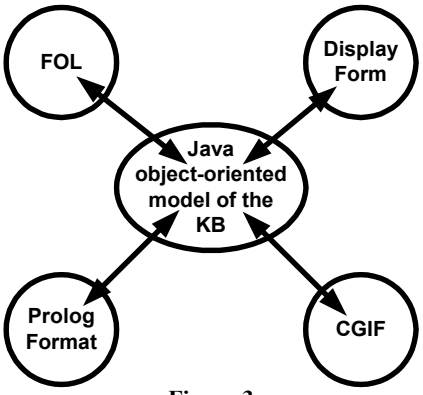

Figure 3

Several formats can be shown simultaneously when displaying information about CGs.

Figure 4 shows an example of mixing CGIF and display form. Through the interface shown in the figure the user can search for CGs containing a specified relation entered in the text field. Through the combo box the user can choose an additional representation format and the CGs will be visualized in the chosen format as well as in display form. If the user clicks on the applet containing the display form of the CG, a new page that contains the four representations of the CG – Prolog format, display form, CGIF and FOL is shown on the screen.

The support of CGIF makes exchanging conceptual information with other CG Tools easier. An example knowledge base from the area of finance in CGIF, Prolog format, and display form can be downloaded from http://www.ksl.stanford.edu/iccs2001/CGTools.

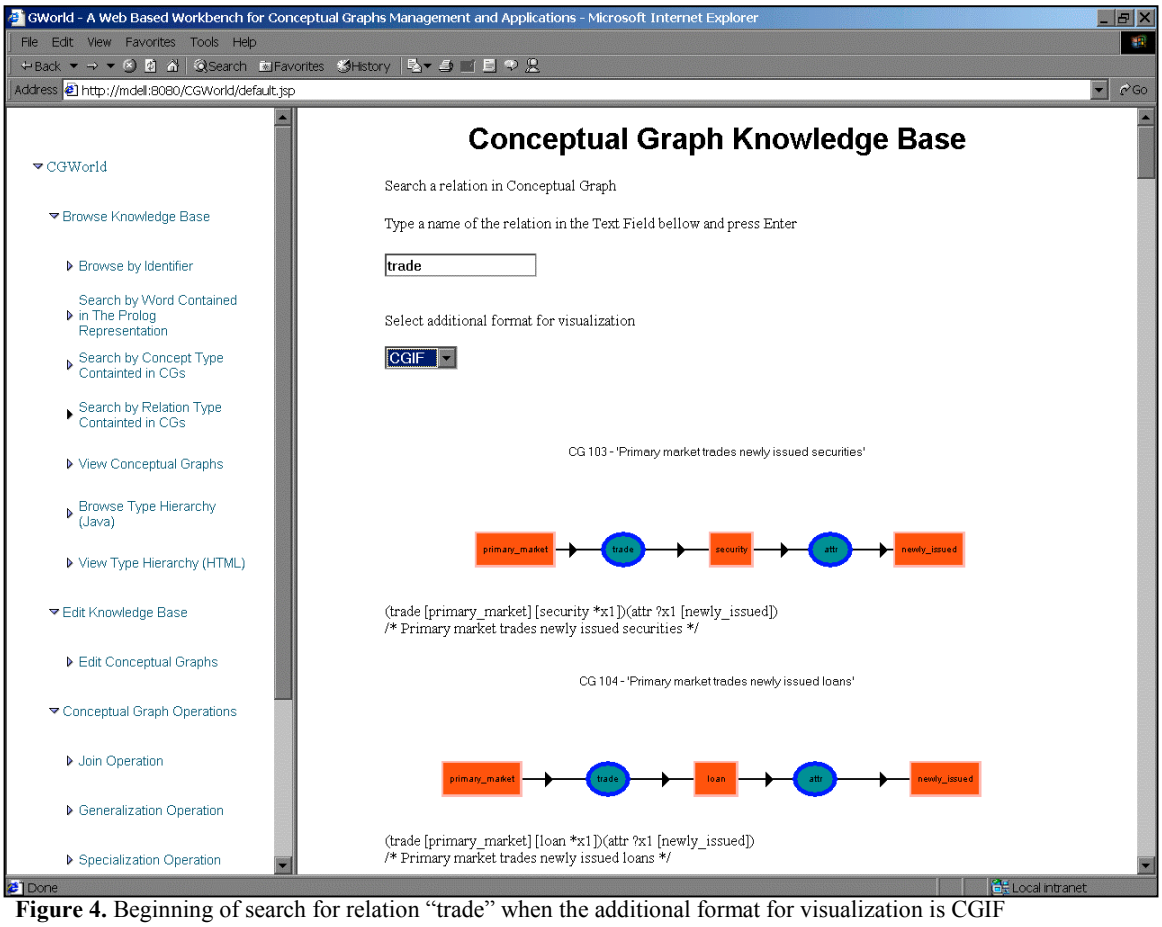

Figure 4. Beginning of search for relation "trade" when the additional format for visualization is CGIF

5 Operations on Conceptual Graphs

Basic CG operations were developed and integrated in the CGWorld workbench during the last year. They were implemented for simple graphs, graphs with identity lines and some special complex graphs. For every operation the user chooses parameters with which to perform the operation. The following is a brief description of each operation including important aspects of its implementation, if applicable, and an example of its user interface.

5.1 *Join Operation*

The implemented algorithm includes the *restrict*, *join* and *simplify* operations, defined in [6], in one operation. The implementation mainly depends on whether the concepts on which to join the graphs are provided or not.

In order to perform join on two concepts they must be equal or compatible; otherwise the join fails. Therefore the correct unification of concepts is crucial for this operation. We describe our implementation of the unification procedure next. First, the concept types are unified according to their kind (*simple* or *compound*). For *simple* concepts type unification is performed, taking into account the concept type hierarchy and if it succeeds then unification of referents is done. Unification is realized for all kinds of referents (see Table 1).

Table 1. Unification of referents when performing the join operation

Kind of referent	Internal representation of referent as Prolog list []	Internal representation of subreferent as Prolog list []
Variable	[fs(quant, lambda)]	every referent
Question	[fs(quant, quest)]	every referent
Universal quantifier	[fs(quant, every)]	every referent
Generic / Generic plural set	[fs(num, sing)] or [fs(num, plur)]	[fs(num, sing)] or [fs(num, plur)]
Definite	[fs(type, def)]	[fs(type, def)]
Measure	[fs(type, meas)]	[fs(type, meas)]
Named individual	[fs(name, Name)]	[fs(name, Name)]
Named individual	[fs(name, Name)]	[fs(refID, Id)], if a clause ind(Id, Name, _) exists
Individual marker	[fs(refID, Id)]	[fs(refID, Id)]
Individual marker	[fs(refID, Id)]	[fs(name, Name)], if a clause ind(Id, Name, _) exists
Definite Set (1)	[fs(num, plur), fs(type, def), fs(name, N1)]	[fs(num, plur), fs(type, def), fs(name, N2)], if N2 $\supseteq$ N1
Definite Set (2) (see Note1)	[fs(num, plur), fs(type, def), fs(name, N1)]	[fs(num, plur), fs(type, meas), fs(quant, Q2)] if Q2 $\geq$ number_elements(N1)
Partial Set (1)	[fs(num, plur), fs(name, N1), fs(type, meas), fs(quant, Q1)]	[fs(num, plur), fs(name, N2), fs(type, meas), fs(quant, Q2)] if N2 $\supseteq$ N1 and Q2 $\geq$ Q1
Partial Set (2)	[fs(num, plur), fs(type, meas), fs(quant, Q1)]	[fs(num, plur), fs(type, def), fs(name, N2)] if number_elements(N2) $\geq$ Q1
Special case (see Note2)	[fs(numb, N), fs(name, L1)]	[fs(numb, N), fs(name, L2)]

Note1: *If* {type_label1 and type_label2 have common subtype, different from type_labeli, i=1,2 or type_label1 = type_label2} *then* the result referent is subreferent with added fs(name, N1).
Note2: *If* {type_label1 and type_label2 have common subtype, different from type_labeli, i=1,2 or type_label1=type_label2 } *then* the result referent is: [fs(numb, plur), fs(name, L)], L= L1 $\cup$ L2.

Also the conformity operation is performed for individuals. For every individual there exists a clause ind(IndID, IndName, TypeLabel) which shows the minimal type to which the individual conforms. An individual conforms to some type label if this label is equivalent to or is a supertype of the type label from its ind clause. If a referent contains a set of individuals the conformity operation is performed for each element of this set.

Compound concepts describe contexts and have conceptual graphs as referents. They are unified only if the described contexts are in relation context/subcontext, which is determined from the context type hierarchy.

When the concepts on which to join the graphs are not provided by the user the performed operation is actually maximal join as it joins graphs on maximal common overlap [12]. This common overlap is found by searching in relations with equal names and checking the corresponding concepts for compatibility.

As mentioned above, the user can choose which of the two join operations to perform. He/she will need to specify *GraphID1, ConceptLabel1, GraphID2,* and *ConceptLabel2* for the first kind of join and *GraphID1* and *GraphID2* for the second kind. When specifying a graph ID, the user will see the graph in display form. The obtained result is also visualized in that form. An example of the user interface for the join operation is presented in Figure 5.

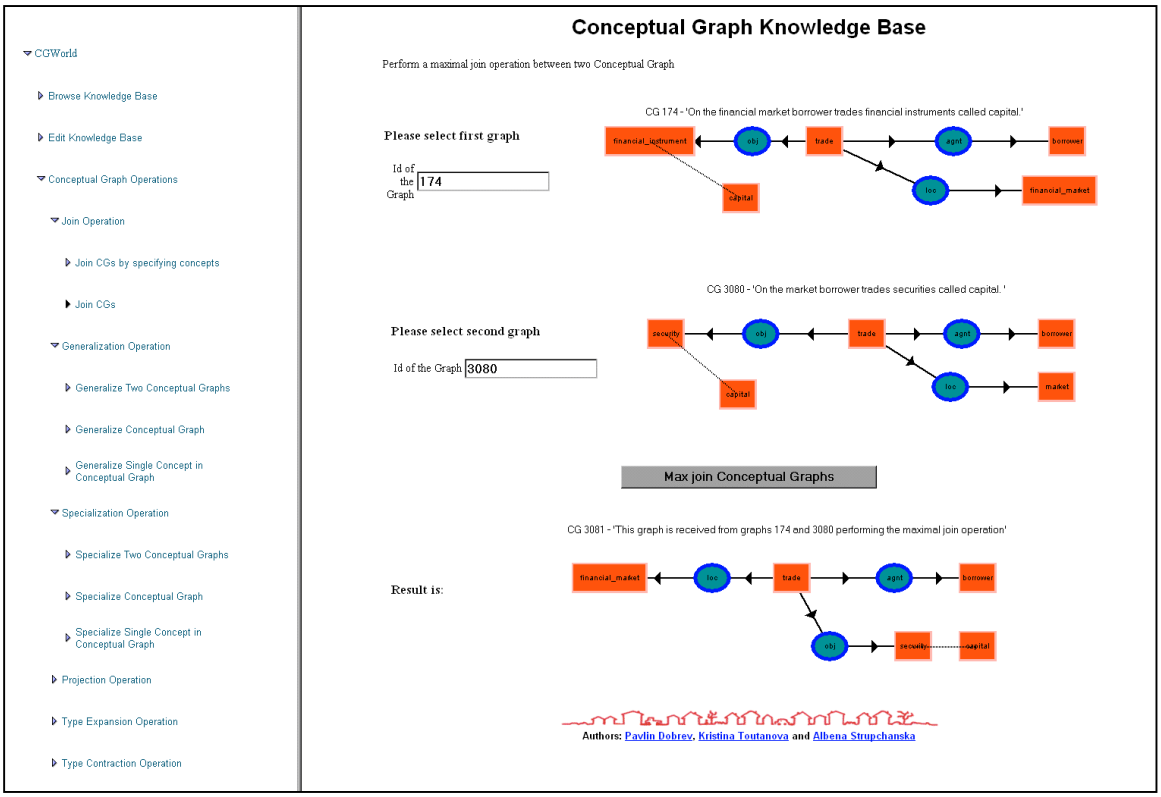

Figure 5. Join Operation

5.2 *Generalization Operation*

There are three implementations of the generalization operation, which apply to different objects of the generalization. The generalization operation can be applied to: *(i)* two conceptual graphs, *(ii)* a single conceptual graph and *(iii)* a single concept in a conceptual graph. Generalization is performed only for concepts according to the concept type hierarchy and the next step will be to extend it for relations too. The result of generalizing two conceptual graphs is the least common generalized conceptual graph built by comparing the corresponding concepts in common overlaps of the two graphs. The generalization operation preserves the truth of the assertions made by conceptual graphs, and it may therefore be useful as a part of a learning component in the LARFLAST project [2].

The CGWorld workbench also supports the opposite of the generalization operation - the specialization operation. The implementation of this operation and the ways of using it are similar to generalization but in different direction according to the concept type hierarchy.

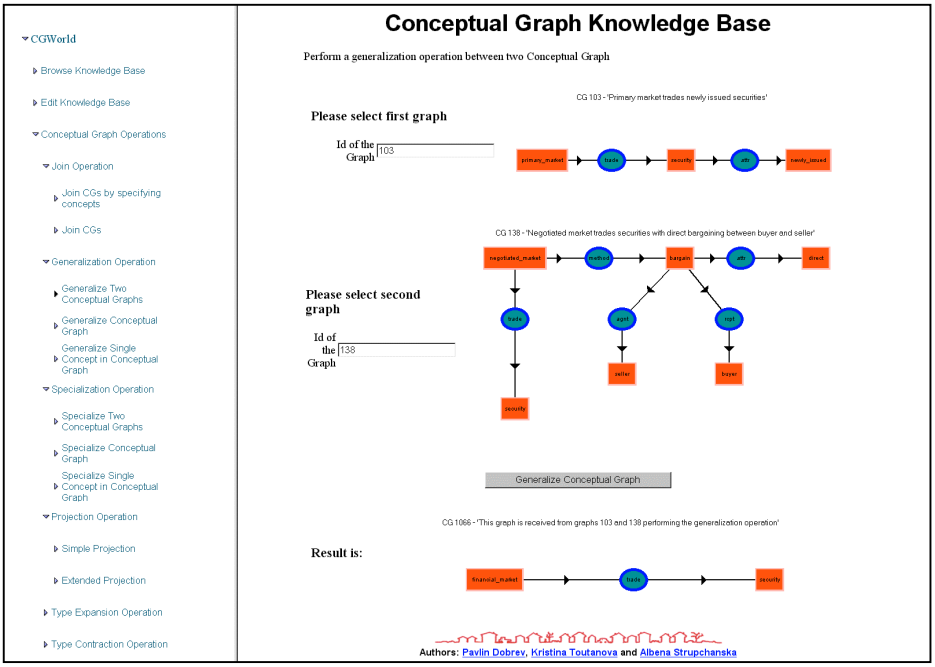

Figure 6. Generalization Operation between two Conceptual Graphs

5.3 *Projection Operation*

The projection operation is the operation used for matching two graphs, projecting a query graph to the knowledge base and as an auxiliary operation in the type contraction operation. This operation projects one graph into another and the resulting graph is not empty if and only if the second graph is a specialization of the first. The implemented algorithm follows the one described in [6, 13, 14] with one additional condition: if there exists a relation type hierarchy then $(r, r') \in \pi(G)$ if type $(r) \geq$ type (r'), $r \in G$, $r' \in G'$, $\pi(G)$: G -> G′.

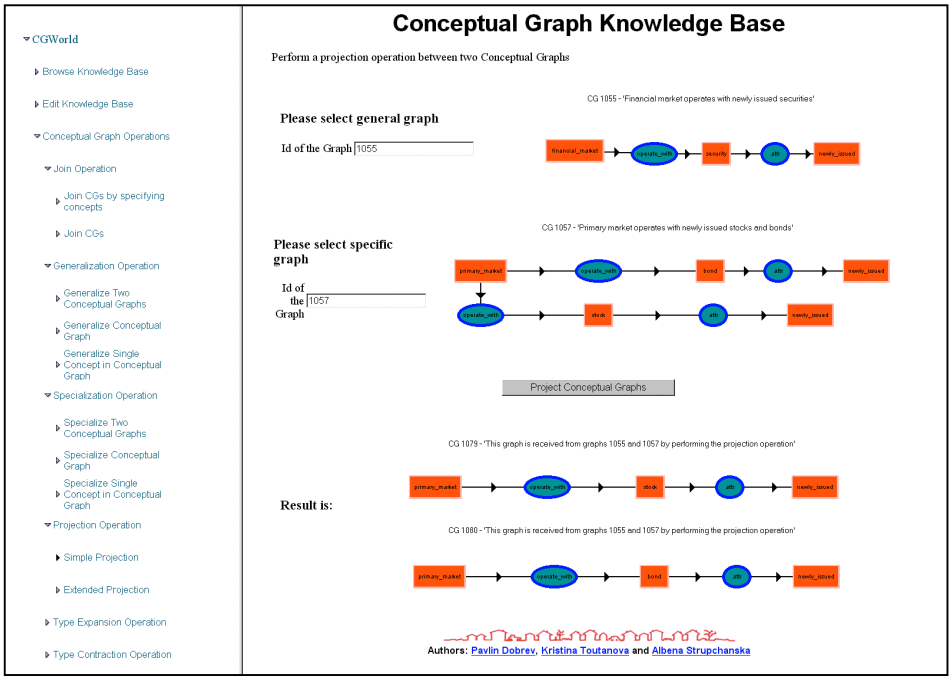

Figure 7. Projection Operation

In the case of projecting a query graph to the knowledge base, the extended projection operation was realized. If the query graph projects to the knowledge base then the result is the corresponding graph from the knowledge base. In this way the operation is useful for extracting graphs from knowledge bases.

5.4 *Type Contraction / Type Expansion Operation*

Both implemented algorithms follow closely the implementation described in [6]. It is recommended to apply them only on canonical graphs if we want the result to be also canonical. These operations take as input a graph ID (the graph to be expanded/contracted) and a type label for which there exists a type definition. The result is a conceptual graph, which is shown in display form. In the current version these operations are realized for simple graphs only but in the near future we hope to expand them to graphs with identity lines and also for relations.

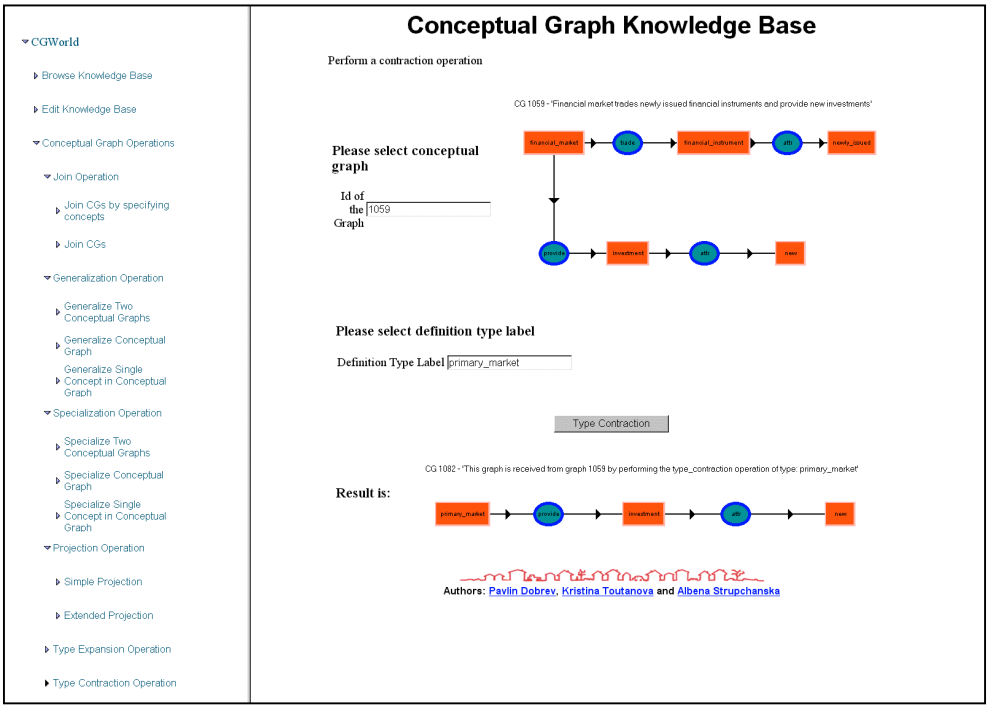

Figure 8. Type Contraction Operation

6 Future Work

In accordance with the new Java technologies the next release of the CGWorld workbench will use an application server with support of the Java 2 Enterprise Edition (J2EE). The current set of HTMLs and Java Server Pages (JSP) will be reused. The application logic that is currently developed as a set of JavaBeans will be implemented as a set of Session Enterprise JavaBeans. This will facilitate the management of user sessions. A set of Entity Enterprise JavaBeans will represent persistent conceptual objects like concept, relation, context, referent, and arc. This will allow the maintenance of large amounts of data and the control of the data integrity will be performed by the built-in mechanisms for transaction maintenance.

7 Acknowledgements

We would like to thank all researchers involved in the LARFLAST (LeARning Foreign LAnguage Scientific Terminology[1]) project and especially Galia Angelova, Ani Nenkova and Toma Nikolov that helped us with the knowledge base development.

References

1. P. Dobrev and K. Toutanova. CGWorld - A Web Based Workbench for Conceptual Graphs Management and Applications. In: G. Stumme (Ed.), Working with Conceptual Structures, Contributions to ICCS 2000, Shaker Verlag, Germany, pp. 243-256.
2. G. Angelova, A. Nenkova, S. Boycheva and T. Nikolov. Conceptual Graph as a Knowledge Representation Core in a Complex Language Learning Environment. In: G. Stumme (Ed.), Working with Conceptual Structures, Contributions to ICCS 2000, Shaker Verlag, Germany, pp. 45-58.
3. Conceptual Graph Standard Information Technology (IT) - Conceptual Graphs draft proposed American National Standard (dpANS) NCITS.T2/98-003 (http://www.bestweb.net/~sowa/cg/cgdpansw.htm).
4. G. Angelova, K. Toutanova and S. Damianova. Knowledge Base of Conceptual Graphs in DBR-MAT. University of Hamburg, Computer Science Faculty, Project DBR-MAT (funded by the Volkswagen Foundation). Technical Report BG-3-98, July 1998.
5. G. Angelova, S. Damianova, K. Toutanova, K. Bontcheva: Menu-Based Interfaces to Conceptual Graphs: The CGLex Approach. In Proc. ICCS 1997, LNAI 1257, Springer, 1997, pp. 603-606.
6. John F. Sowa, Information Processing in Mind and Machine. Reading, MA: Addison-Wesley Publ., 1984.
7. G. Angelova, K. Bontcheva: DB-MAT: Knowledge Acquisition, Processing and NL Generation Using Conceptual Graphs. In Proc. ICCS 1996, LNAI 1115, Springer 1996, pp. 115-129
8. D. Lukose. CGKEE: Conceptual Graph Knowledge Engineering Environment. ICCS 1997: pp. 598-602.
9. S. Pollitt, A. Burrow, P. Eklund. WebKB-GE - A Visual Editor for Canonical Conceptual Graphs. ICCS 1998: pp. 111-118
10. H. Delugah, CharGer – A Conceptual Graph Editor written by Harry Delugah (http://www.cs.uah.edu/~delugach/CharGer/).
11. H. Delugah, CharGer: Some Lessons Learned and New Directions. Working with Conceptual Structures, Contributions to ICCS 2000: pp. 306-309
12. J. Fargues, M. Landau, A. Dugourd, L. Gatach, Conceptual graphs for semantics and knowledge processing. In IBM Journal 30(1), 1986
13. John E. Heaton, The projection operation of Sowa's conceptual graphs. Paper UK, Loughborough University of Technology.
14. Guy W. Mineau, Introduction on Conceptual Graphs: Finding Common Generalizations and Compatible Projections, Conceptual Strucrures, 1992: pp. 295-310

[1] INCO Copernicus'98 Joint Research Project #977074

Appendix A: Installation

Installation Steps

1. Install JDK 1.3.1
2. Install Sicstus 3.8.x (the tool has been tested with versions 3.8.2, 3.8.5 and 3.8.6 evaluation copy)
3. Add jasper.jar (Sicstus java interface) to your classpath
4. Install Tomcat (the tool has been tested with versions 3.1.1 and 3.2.2)
- unzip the zip file
- define tomcat_home=[home tomcat directory]
- define java_home=[home jdk directory]
5. Put CGWorld.war in the tomcat\webapps directory
6. Start Tomcat
7. CGWorld should now be available at: http://your_host:8080/CGWorld/Index.htm
   If you are at the same computer the URL is: http://localhost:8080/CGWorld/Index.htm

Installation Troubleshooting

One typical error arises when jasper.jar is missing from the Tomcat classpath. This is because Tomcat also contains a jasper.jar file (different) and this may involve conflicts. (You can find jasper.jar in the Sicstus bin directory) Tomcat adds its jasper.jar to the classpath automatically. You have to manually add Sicstus's jasper.jar as specified in step 3 above.

Conversion from CGIF to Internal Format (CGLex)

You can convert files containing graphs in CGIF format to the internal format using the BAT file convert.bat in the root directory of CGWorld.
If you run convert.bat without parameters all files with extension .cgf from the folder CGWorld\kb\cgif will be converted into CGLex format and saved in the folder kb\cgif\converted. For every .cgf file, the tool creates one .kb file for the conceptual graphs and another type_hierarchy_…kb file for the type and relation hierarchies. Currently, the directory CGWorld\kb\cgif contains the cgif files that were prepared for the CGTools workshop at ICCS 2001. The tool supports level 1 and 2 from these files and most of the basic graphs.
The program convert.bat can also be used with some additional parameters. You can use:
convert <file_name>
When you use this command, the tool creates files CGKB.kb and Type_Hierarchy.kb (if there is a type or relation hierarchy) in the default directory (typically main CGWorld directory).
You can also use:
convert <file_name> <identifier>
This means that all identifiers for conceptual objects generated by the program will be greater then <identifier>.