A Tool to Support Multiple Visual Languages for Distributed Software Development on the Web

Roberto Flores-Méndez, Rob Kremer
Software Engineering Research Network, University of Calgary, Canada
{robertof, kremer } @cpsc.ucalgary.ca

Abstract

Concept mapping is a simple and intuitive visual form of knowledge representation. Concept maps can be categorized as informal or formal, where the latter is characterized by implementing a semantics model constraining their components. Software engineering is a domain that has successfully adopted formal concept maps to visualize and specify complex systems. Automated tools have been implemented to support these models although their semantic constraints are hard-coded within the systems and hidden from users.

This paper presents the Constraint Graphs and jKSImapper systems. Constraint Graphs is a flexible and powerful graphical system interface for specifying concept mapping notations. In addition, jKSImapper is a multi-user concept mapping editor for the Internet and the World Wide Web. Together, these systems aim to support user-definable formal concept mapping notations and distributed collaboration on the Internet and the World Wide Web.

1. Introduction

This paper is about formal concept mapping and distributed concept mapping collaboration. Concept maps are a common form of visual language used pervasively in a wide variety of applications. Concept map editors may implement concept mapping formalisms, which constrain construction of the concept maps to conform to a specific language.

Software engineering is a domain where formal concept maps have been successfully applied. Examples of their application can be found in the area of visual object-oriented modeling notations, where diagrams are used to represent objects' relationships, inheritance and collaboration, for example. Graphical systems have been implemented that help to construct and manipulate these diagrams. However, these systems do not adapt well to notation modifications, since it is common for them to define diagram constrains as part of the implementation code. Constraint Graphs, in the other hand, is a versatile system that allows users to visually specify the notations applied to the construction of formal concept maps. In addition to this application, jKSImapper is presented as a system that supports multi-user distributed elicitation of informal concept maps on the Internet. Together, these two systems allow multi-user distributed, concept mapping collaboration in the Internet and the World Wide Web.

A multi-user distributed concept mapping tool would allow software engineers to collaborate at a distance using familiar, visual notations in a shared, real-time, interactive graphical workspace. Furthermore, familiar notations could be easily extended to encompass new concepts as new requirements arise.

2. Concept Maps

One of the mechanisms for organizing and communicating knowledge among individuals are concept maps [1]. Concept maps encompass a wide variety of diagrammatic knowledge representations. They can be defined as diagrams composed of links and nodes of different types. Concept maps can graphically represent and organize arguments and thoughts, providing an alternative to natural languages as a means to communicate knowledge.

Concept maps have been applied on diverse areas such as education [2], management [3], artificial intelligence [4], knowledge acquisition [5] and software engineering [6] [7] [8].

Concept maps can be categorized as informal or formal, according to the level of syntactic constraints implemented for interpreting and organizing information. Concept maps are not formal unless they have an associated computational semantics. Therefore, a formal concept map's nodes and links types and their

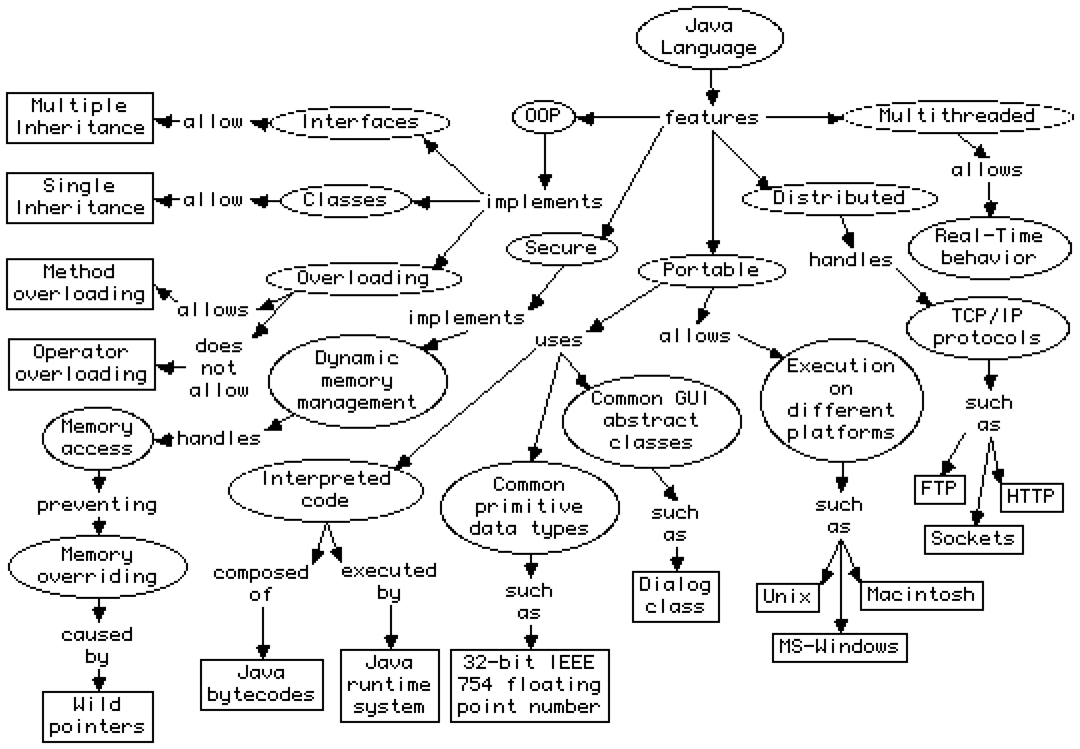
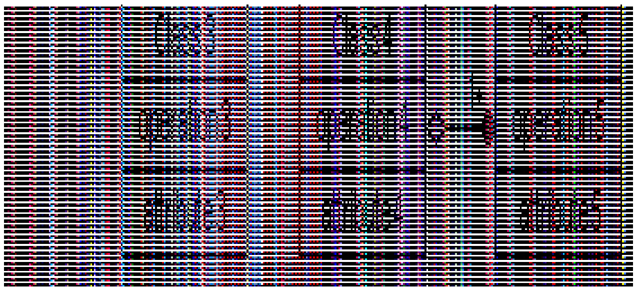


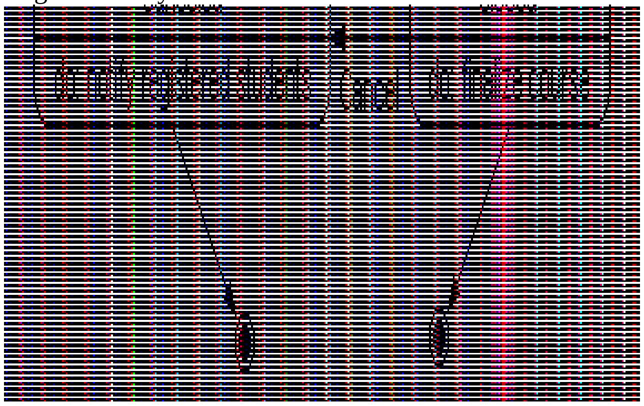
Figure 1: An informal concept map about the Java language.

interconnections must be constrained to allow for computer support [9]. However, the absence of semantic constraints does not mean that informal concept maps are not useful just because they may not be computational from a computer's point of view (see Figure 1). For humans, even informal concept maps appear to have greater "computational efficiency" than other forms of knowledge representation, such as text and logic notations [10].

3. Concept Mapping Languages in Software Engineering

Software engineering is an area where concept mapping languages can be (and have been) applied. Concept mapping models are found in the domain of visual object-oriented modeling systems, where concept maps are used to capture essential parts of systems and then applied to the analysis and design of software applications. Examples of these notations are the Object Model Technique (OMT) [6], the Booch model [7], the Coad model [8] and the Unified Method Language (UML) [11]. Figure 2 illustrates the OMT notation for a small group of classes and Figure 3 exemplifies a UML state transition diagram for a hypothetical University Registration System.

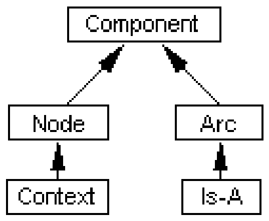




Although these examples can both be identified as concept maps, they implement very different semantic constraints and distinct graphical decorators. While it is possible to find systems that manipulate these notations (e.g., Rational Rose [13]), these applications are hard coded to manipulate those specific semantics constraints. The following section will describe a system that defines a minimum set of constraints required by concept mapping languages while allowing designers to visually define the semantics and syntax to which users are restricted during the elicitation of diagrams.

4. Constraint Graphs

Constraint Graphs [14] is a concept mapping system that allows the specification of graphical formalisms in a manner similar to typical drawing programs. As illustrated in Figure 4, the basic premise in Constraint Graphs is that all graphs contain only two basic object types: *nodes* and *arcs*, which are collectively referred to as *components*. Each of these basic types may be further elaborated into a lattice of *subtypes*, where each new subtype can introduce new *attributes* on top of those of its parent type.



The type lattice of a constraint graph is fully integrated with the graph itself. That is, there is a special subtype of arc, called the *is-a* arc that can be drawn into the graph just as any ordinary arc type (like *owns*, *has-color*, or *bigger-than*) would be. Any component that lies at the tail of an *is-a* arc is considered a subtype of the component at the head of the *is-a* arc. Of course, *is-a* adds several constraints on top of a basic arc because the *is-a* projection of the graph must be a lattice (because no type may be a direct or indirect parent of itself). These constrains include the fact that *is-a* arcs cannot form cycles and *is-a* arcs are always directed binary arcs.

A common concern is constraining the component types on which arcs can terminate. This is accomplished by simply attaching the terminals of the defining arc to the appropriate type objects. For example, if one wants to describe the *employee-of* relationship as an arc, one could define it as follows (where *legal-individual* is a person or corporation): [*legal-individual*] ←*employee-of*— [*person*], which would constrain any subtype of employee to source at a component of at least type person, and sink at a component of at least type *legal-individual*.

In addition to *node*, *arc*, and *is-a*, there is a subtype of node, called *context*, which is a labeled box that can contain other components.

For convenience, a constraint graph is divided into levels. Level 1 consists of the primitive graph types themselves – *node*, *arc*, *is-a* and *context* – and it is immutable as far as the user is concerned. The designation of the rest of the levels is left to the discretion of the formalism implementor. Generally, levels 2 and 3 should be at the system level where the basic types are defined according to the target formalism. These types should normally be considered immutable by any end users, since to disrupt them may impede interpretation of the graph. Two system levels are often used (where level 2 is hidden from the end user and used for hidden type hierarchies, while level 3 is public and used to populate the space of type identifiers for the end user). Level 4 is generally considered to be the user level, where the end user builds some specific knowledge structure. More levels are possible: for example, level 4 might be used to construct types in some specific domain, and a fifth level might be added to hold objects of that domain within some hypothetical world.

4.1. A gIBIS Example

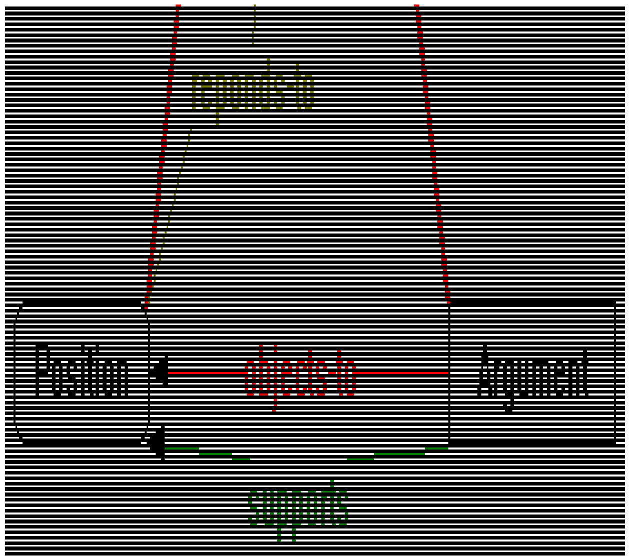
gIBIS [15] is a simple visual language used in business decision making. gIBIS (which is an acronym for "Graphical Issue-Based Information System") is a notation consisting of *typed nodes* and *typed arcs* only (*contexts* are not implemented). This notation defines three *types of nodes*.

- *Issues*: which usually represent decisions to be made;
- *Positions*: which are statements of a possible resolution of a particular Issue; and
- *Arguments*: which describe statements supporting or objecting to a particular Position.

This notation also defines seven *arc types*.

- *Questions* and *is-suggested-by*: which link an *Issue* to any other type of node;
- *Specializes* and *Generalizes*: which link an *Issue* to another *Issue*;
- *Responds-to*: which links a *Position* to an *Issue*; and
- *Supports* and *Objects-to*: which link *Argument* nodes to a *Position*.

Figure 5 shows a simple gIBIS concept map, which could have been created as part of making a decision about which type of computer an organization is going to buy. In this diagram, *Issue* nodes are shown as ellipses, *Position* nodes are shown as rounded rectangles and *Argument* nodes are shown as rectangles.



To model the gIBIS notation, it makes sense to capture the notion of a general *gIBIS node* (the union of *Issue*, *Position*, and *Argument*), and to capture the notion of a general *gIBIS arc*, specifying that it must connect a *gIBIS node* to another *gIBIS node*. Both terminals of the *gIBIS arc* terminating on the *gIBIS node* serve to restrict the terminals of any *gIBIS arc* subtype to terminate on components that are subtypes of *gIBIS node*.

Finally, the actual gIBIS nodes and arcs can be defined. The three gIBIS nodes, *Issue, Position,* and *Argument*, are first defined as top-level Constraint Graph nodes, then an *is-a* arc is drawn from each to the general gIBIS node. Immediately after the *is-a* arcs are drawn, the visual attributes (colors, shape) of the nodes change to match those of the general gIBIS node. But *Position* and *Argument* are meant to have different shapes, so the attributes of these two nodes are changed to represent the appropriate *shape* attribute. These operations yield the nodes in Figure 6.

The arcs are created in a similar way, but to actually show the *is-a* arcs in the figure would clutter up the diagram unreasonably, so a slightly different technique is used. Instead of creating the arcs as generic Constraint Graph arcs, they are created directly as subtypes of the general gIBIS arc. This has the effect of creating the appropriate *is-a* arc in the Constraint Graph, but not drawing the *is-a* arc in the interface.

All of the gIBIS components, except for *gIBIS node* and *gIBIS arc* (which are level 2) should be placed in level 3, since level 3 components will be made visible to the end user. The level 2 components will be hidden.

5. Distributed Concept Mapping

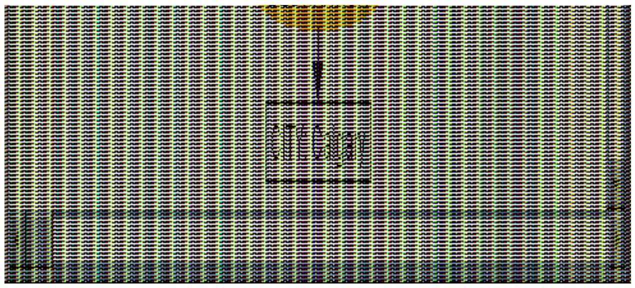
Constraint Graphs has been described as a powerful and flexible graphical system interface for specifying concept mapping notations. Although this description may imply a tightly coupled system, Constraint Graphs was developed with a great deal of attention on the software engineering aspects. In particular, it has a very modular design: the constraint engine and the user interface are actually quite independent and communicate only though a very narrow interface. This characteristic allows the graphical interface to be substituted with other interface models while minimizing the impact of such changes on the constraint engine. The objective of the present section is to describe a distributed architecture suitable to replace the currently on-site one-user graphical interface. This architecture allows multiple remote users to concurrently manipulate a shared concept map located in a centralized server on the Internet.

5.1. jKSImapper

jKSImapper [16] is a multi-user client/server application that supports distributed elicitation of informal concept maps on the Internet. This system allows users to concurrently manipulate a remotely-located shared concept map by means of a client standalone program or a web browser applet. To provide this service, jKSImapper features three programs: two client programs (named *jKSImapper* and *jKSImapplet*) and one server program (called *jKSImapper Server*). These programs were entirely implemented using the Java programming language [17]. This allows *jKSImapper* to provide a great deal of portability and integration with prevailing commercial web browsers.

jKSImapper and *jKSImapplet* are client programs that implement a drawing surface where concept maps are displayed and directly manipulated. Concept mapping components are handled using a pointing device and they are created and modified using menu options (*jKSImapplet* uses HTML-embedded JavaScript controls instead).

jKSImapper, which is illustrated in Figure 7, is a standalone program that can edit concept mapping data found locally in the client computer, accessible via a URL or provided by a *jKSImapperServer* process. Under local and URL access modes, data is non-shareable and it is manipulated locally without the assistance of any external process. In the other hand, data provided by a *jKSImapperServer* requires collaboration between clients and server, since clients are required to join a server session where concept maps are manipulated and shared (sessions are further detailed on the *jKSImapperServer* section below).



jKSImapplet is a Java applet that is embedded inside HTML pages and automatically executed when it is downloaded as part of normal web navigation. In contrast to the standalone version, *jKSImapplet* does not allow concept mapping data to be stored or read from the client computer. This makes *jKSImapplet* completely dependent of the server process (which must reside on the same server computer from where the applet was downloaded).

jKSImapperServer is a server program supporting multi-user manipulation of concept mapping data files. *jKSImapperServer* allows client concept mapping elicitation programs to retrieve server concept mapping data files, while coordinating clients' actions in order to maintain a consistent state in the concept maps being concurrently shared by several clients. When a client requests a file, the server process creates a session (if one does not exist) to which the client is connected as a new member. A session can be described as a dynamic shareable centralized state representation of a concept mapping data file, and it represents a common coordination location to a group of clients accessing a shared data file.

The three programs implement a session-oriented command broadcasting mechanism to coordinate a community of clients engaged in a session. This technique requires clients to transform user events into commands that are transmitted to the server and routed to the session on which the issuer has membership. Once their session is located, these commands are broadcast to all members (including the issuer) for execution.

Under this architecture, Constraints Graph registers with the server process as a listener service provider to which client programs submit user entries for notation checking. In a typical scenario, a user interacts with a concept mapping graphical interface, transforms user events into requests and sends these requests to the server (and to the constraints processing engine) for validation. In the event of a valid request, Constraints Graphs issues the proper resulting commands to *jKSImapperServer*, which then submit them to the issuer and all the other clients connected to the same session. In addition to commands broadcast to all clients, Constraint Graphs can also issue commands directed to a single client. Such is the case of commands resulting from invalid requests (where an error message command is send to the issuer for user feedback), and commands containing context-specific information (such as data to be displayed in components pop-up menus).

6. Conclusion

This paper has described concept maps as a type visual language that can be (and is) used in software engineering and knowledge representation. Constraint graphs is a flexible concept mapping editor that can adapt to a wide variety of languages, including those used in software engineering. This is useful in itself, but by combining it with jKSImapper, software engineering groups can manipulate concept maps over the Internet in a shared, real-time, graphical environment. This should allow distributed workgroups to collaborate using familiar visual languages in new and natural ways.

References

[1] Gaines, B. R. and Shaw, M. L. G., Collaboration through Concept Maps. Computer Supported Cooperative Learning. Bloomington, October, 1995.

[2] Novak, J. D. and Gowin, D. B., Learning How To Learn. Cambridge University Press, New York, 1984.

[3] Axelrod, R., Structure of Decision. Princeton, Princeton University Press, New Jersey, 1976.

[4] Quillian, M. R. Semantic memory. Semantic Information Processing. MIT Press, Cambridge, Massachusetts. p. 216-270, 1968.

[5] McNeese, M. D., Zaff, B. S., Peio, K. J., Snyder, D. E., Duncan, J. C. and McFarren, M. R., An Advanced Knowledge and Design Acquisition Methodology for the Pilot's Associate. Harry G Armstrong Aerospace Medical Research Laboratory,

Wright-Patterson Air Force Base, Ohio, 1990.
[6] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. Object-Oriented Modeling and Design. Prentice Hall, New Jersey, 1991.
[7] Booch, G., Object-Oriented Analysis and Design with Applications. Second Edition, Benjamin/Cummings, Redwood City, CA, 1994.
[8] Coad, P., North, D. and Mayfield, M., Object Models: Strategies, Patterns and Applications. Prentice Hall, New Jersey, 1995.
[9] Kremer, R., Concept Mapping: Informal t o Formal. Proceedings of the International Conference on Conceptual Structures, University of Maryland, 1994.
[10] Lambiotte, J. G., Dansereau, D. F., Cross, D. R. and Reynolds, S. B., Multirelational Semantic Maps. Educational Psychology Review, 1(4), pp. 331-367, 1984.
[11] Booch, G., Rumbaugh, J. and Jacobson, I., Unified Modeling Language User Guide. Addison-Wesley, Reading, Massachusetts, 1998.
[12] Rational Software Corporation., Object-Oriented Analysis and Design with UML, Rational Software Co., 1997. Available at: http://www.rational.com/products/rose/ooadwithuml4.0.zip
[13] Quatrani, T., Visual Modeling with Rational Rose a nd UML, Addison-Wesley, Reading, Massachusetts, 1998.
[14] Kremer, R., Constraint Graphs: A Concept Map Meta-Language, Computer Science Department, University of Calgary, Canada., PhD dissertation, 1997.
[15] Conklin, J. and Begeman, M., gIBIS: A Hypertext Tool for Team Design Deliberation, Hypertext '87, pp. 247-251, 1987.
[16] Flores-Méndez, R. A., Programming Distributed Collaboration Interaction through the WWW, Computer Science Department, University of Calgary, Canada, MSc. Thesis, 1997.
[17] Arnold, K. and Gosling, J., The Java Programming Language Specification, Second Edition, Addison-Wesley, Reading, Massachusetts, 1998.