To appear in: Proc. of 7th International Workshop on Groupware CRIWG 2001,
Darmstadt, Germany, 6 - 8 September 2001, IEEE CS Press.

Integration of Domain-specific Elements into Visual Language Based Collaborative Environments

Niels Pinkwart, Ulrich Hoppe, Katrin Gaßner
University of Duisburg
pinkwart@informatik.uni-duisburg.de

Abstract

This paper presents an approach for the integration of domain related elements and operational semantics into collaborative environments based on visual languages. This integration allows for supporting domain specific collaborative tasks, e.g. in the area of "collaborative discovery learning" in science education, by integrating data modelling with generic discussion support. A special focus is set on flexibility and parameterisation of the system which is achieved through providing the syntax definition of the visual language as a separate resource file.

1. Introduction

A central aim of delivering computer support to collaborative learning activities is to facilitate *co-constructive activities*. This is typically achieved through shared workspace environments which allow the co-learners to synchronously and jointly elaborate external representations. Systems may differ considerably with respect to the degree of semantics or structure that is explicitly captured by the computerised representation: Whereas the internal structure of whiteboard (drawing) tools is based on strokes, colours and geometrical shapes, concept mapping tools constitute "semantic" relations between certain objects or nodes. However, it is not clearly defined in how far the semantics of the representation is interpreted by the machine. Internally, a concept mapping tool may just be based on a graph representation which is perfectly acceptable for many purposes. On the other hand, more specific representations such as Petri Nets or visual programming languages come with a clearly defined and rich internal operational semantics. In this paper, we present some ongoing work within the European DiViLab project [1] focusing on an approach of adding a semantic structure to co-operative visual language environments without assuming a priori a given specific domain semantics.

The main objective of this approach within this project is to support a great portion of the typical laboratory activities in science education: taking group decisions about experiments, building and verifying hypotheses as well as modelling data, integrated with discussing about the results of experiments with partners. Yet, also co-operative tasks in other domains, mainly those that require both collaboration and domain dependent activities, can easily be supported due to the flexibility of the approach.

2. Treatment of domain content in existing environments

Concerning the treatment of domain content in existing visual language environments, two extremes are currently predominating: on the one hand, some specialised visual languages like system dynamics models [2] provide a complete semantic definition of each object and therefore allow for detailed model based support. On the other hand, flexible systems like Belvedere [3] or the CardBoard [4] do not interpret the content of the objects but only restrict the rhetorical or argumentative type of an object (e.g. "hypothesis", "conclusion").

Most *general purpose tools* aim at obtaining easy-to-communicate visualisations rather than at interpreting semi-formal representations. The focus is on providing interactive interfaces to represent a domain or support a certain task, not on system-internal structure and semantics. Mainly, the supported task is co-operative, e.g. discussion [5] or scientific argumentation [6] or knowledge elicitation [7]. The gIBIS System [8], is an early example of using visual languages to represent and elaborate arguments during a design process. The Sepia System [5] was developed for the co-operative design of hypermedia documents. It offers four types of visual workspaces (planning, argumentation, content and a rhetorical space), which are connected through a central database. Belvedere [3] was designed to teach students scientific argumentation. It offers two types of content objects, data and hypotheses. The system is able to check and criticise user input based on certain argumentation

rules like “do not formulate hypotheses without data suporting them”.

The work reported here elaborates on the CardBoard environment [4] which allows to create multiple visual languages by parameterising a general shared workspace environment. A special language profile specifies the syntax of the respective language, in terms of a set of relations, their argument slots, and of basic object types. To add or plug in semantics in terms of domain models or knowledge bases, an interface is provided that transfers actions from the visual language environment to a semantic module [9]. In this sense, CardBoard is of general-purpose type, yet it is relatively easy to extend with semantics.

Domain specific visual tools like STELLA [10] or Rational Rose [11] provide visual interfaces for existing model semantics, here “system dynamics” (STELLA) or UML (Rational Rose). Although these tools can be shared through general synchronisation mechanisms, there is no special focus on supporting synchronous collaboration. A recent example of a collaborative learning environment based on a domain-specific visual language is the COLER system [12]. COLER supports the co-construction of entity-relationship (ER) models for database modelling.

3. Central ideas and challenges exemplified

For man applications it is desirable to have a system which integrates multi-purpose structuring tools, e.g. for discussion, with specialised domain-related functions. This has recently been proposed by van Joolingen [13] in regard of integrating simulations with argumentation environments to support “collaborative discovery learning” in science education.

Generalising this from the viewpoint of visual languages leads to “partially formal” languages in which some objects have a specified domain-related functionality and semantics, enabling the system to provide e.g. special tools, means of analysis or domain-related support, mixed with other elements that represent generic aspects of the environment (e.g. discussion statements).

Dividing a visual language into content objects (nodes) and relation objects (edges or connections between content objects), the content objects typically contain textual information or images although many other media types are imaginable. The “meaning” or interpretation of the content of an object can generally only be derived by the system if a domain specific context is available, either predefined, as e.g. within a UML tool like [11], or dynamically assigned by the user.

Example 1: Enrichment of content objects

Imagine a brainstorming session in a group of musicians, trying to find the theme for their next song. Entering the text “ACE” in an “idea” node of a shared workspace is here meaningless for the system at first, but a context-specification like “domain = music” provides a semantic clue – probably similar to the one the users have. Having this semantic information, the system is able to offer further specific content objects like “transpose”, “play” or “show as notes” (see Figure 1).

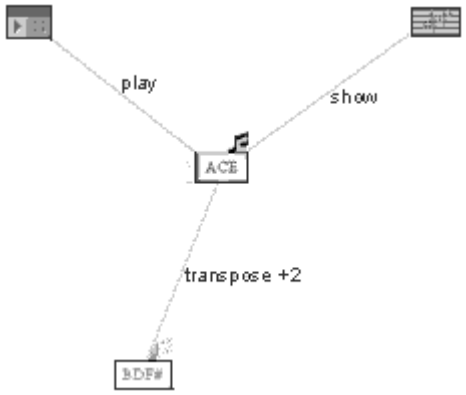
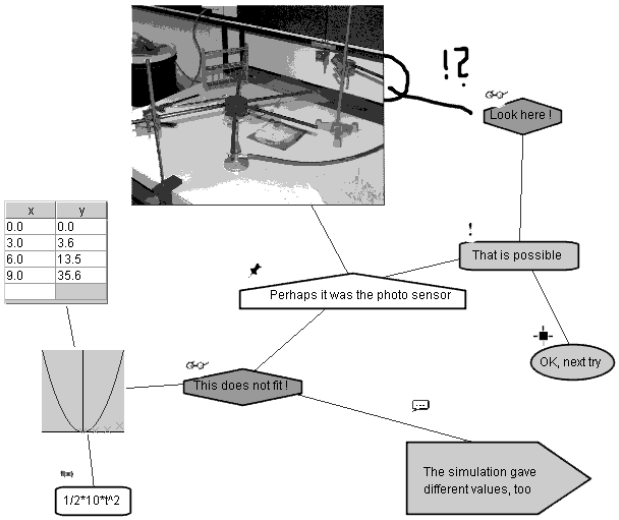


Figure 1. Domain-specific object spaces

Example 2: Enrichment of relation objects

Concerning the relation objects, we can state that in most of the general-purpose visual languages, relations between content objects are of a more or less general type like “supports” or “contradicts”, sometimes just “related to” or “associated with”. Yet, the same idea as for the content objects also holds here: having information about the domain context of content objects and the necessary input contexts of a relation object offers the possibility to integrate highly specialised tools with general environments.

Here, an example is the integration of data modelling and argumentation in the domain of experimental sciences. Figure 2 shows a possible result of a co-operative discussion about an experiment on free fall going hand in hand with the modelling of data. This example is generalisable and clearly points out the major advantage of this approach: remaining in a flexible environment that supports the overall task, the users can “plug in” the specific components they need without loosing the domain context (here: analysis of an experiment) or the social context (doing the analysis with a partner).



x	y
0.0	0.0
3.0	3.6
6.0	13.5
9.0	35.6

Figure 2. Augmenting discussion spaces with data modelling

4. Approaches for the definition of domain content in visual languages

To implement our central idea – extension of co-operative visual language environments towards a flexible integration with domain content for reasons of intelligently supporting workflow management and reflection – we have identified the following challenges in terms of representation, information and interpretation:

- How can varying but domain dependent content be *represented* flexibly?
- How can a system *interpret* externally defined context information following a "plug in" approach?
- Which *levels of interpretation* are imaginable?
- Which user interfaces are appropriate to support the users' handling the *information*?
- How should an application be structured into components to reach these aims?

These questions can be broken down into two categories - the first one dealing with the meta level of the overall system structure, the second one with the micro level around the topic "representation of domain semantics in visual languages".

4.1. System structure

The main function of the system component structure is to facilitate the handling of varying visual languages. In our approach, this is realised by the approach shown in Figure 3.

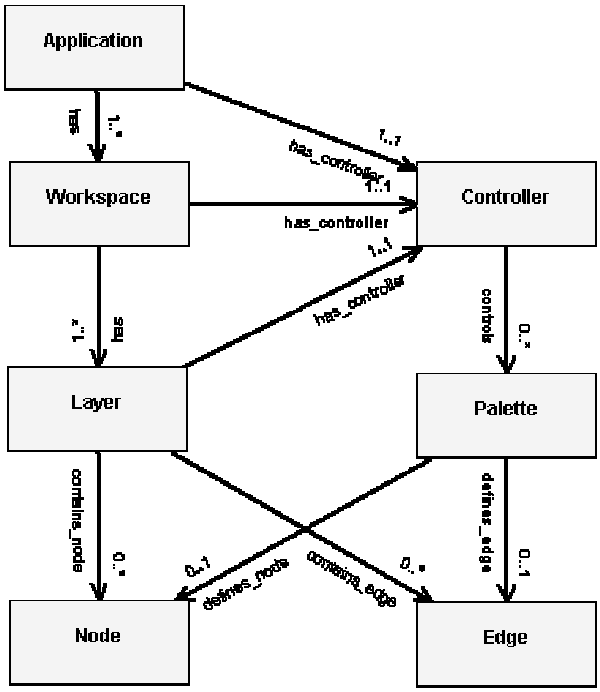


Figure 3. System and controlling structure

The environment is able to manage several workspaces represented in different windows. This offers the possibility to have private and shared sessions simultaneously or to clearly separate of independent collaborative tasks.

Each workspace can contain a number of transparent layers which can have "solid" objects like e.g. handwriting strokes or images. Similar to the workspaces, layers can be private or shared. A typical use case for this is a private handwriting layer used for personal annotations (see Figure 4).

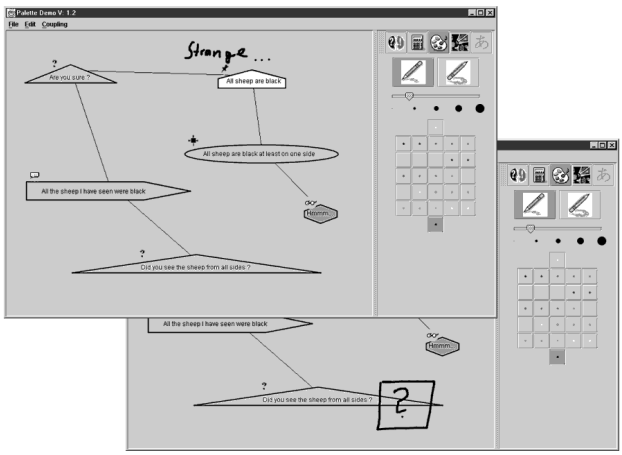


Figure 4. Layer-wise synchronisation

The controller component is the central interface between the static (application level) and the dynamic (visual language level) components of the system. It manages the assignment of visual languages to workspaces and layers and controls the options for the users, in particular the creation of new workspaces and new layers in existing workspaces.

The example below shows a (simplified) declaration of an environment named "Petri net discussion" with one predefined workspace named "Petri net editor". The user is allowed to open other workspaces ("type=extendable" in environment tag). The predefined workspace has two layers, one for handwriting and one containing two palettes (representing visual languages), petri net symbols and generic discussion elements. No more layers are allowed here ("type=fixed" in workspace tag).

```
<Environment label="Petri net discussion"
             type="extendable">
 <Workspace label="Petri net editor"
            type="fixed">
  <Layer label="Graph layer"
         mode="graph">
   <Palette location="petri.xml"/>
   <Palette location="discuss.xml"/>
  </Layer>
  <Layer label="Annotations"
         mode="handwriting"/>
 </Workspace>
</Environment>
```

4.2. Representation and interpretation of domain dependent content in visual languages

A major problem in trying to build a system that uses externally defined visual languages together with another representation of domain dependent content or models is due to the facts that (usually) the domain dependent content

- might have a complex structure (model),
- might require a complex visual or other representation (view),
- might have to be interpreted (controller).

An external definition of a visual language, however, would usually allow for easily representing the model, e.g. using XML. The controller part (and to some extent also the view) is far more difficult as it typically contains the operational semantics and thus involves active operations - something that a pure data file is not designed for. Going one step further, this leads to the question of how, if the operational domain semantics cannot be captured in external data files, the representation of the interpretation can be linked to the representation of the model.

The solution we propose relies on *reference frames*. Each object (at least those with defined domain dependent semantics) provides a link to one of these and, in order to interpret the content of an object, the system generally performs three steps:

1. look for the appropriate reference frame,
2. consult the reference frame with the current content of the object as parameter,
3. get as result the changes in the object structure.

So, basically interpretation here means interpretation in a – or through a – reference frame. Yet, there remain some possible ways of realising these frames which are compared (using XML for data and Java for code) in the following. The objects shown in Figure 5 serve as an example, where one node is used for input of function terms and one for visualising function graphs.

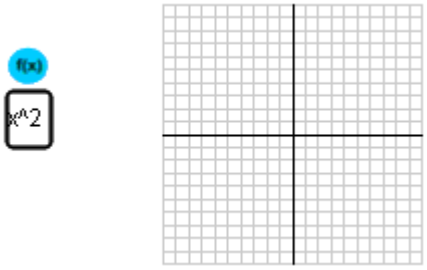


Figure 5. Example nodes

Approach A: XML file links to class files:

In this approach, the XML file contains only links to Java classes that contain the complete code (model, view and controller). The interaction between the two types of nodes is externally defined in a reference frame expressed in the class "Functions".

```
<node type="FunctionTermNode.class"
      frame="Functions.class"/>
<node type="FunctionDisplayNode.class"
      frame="Functions.class">
```

While this approach is very general and covers a wide variety of domains, the major disadvantage is the lack of usability. In order to construct a visual language for a specific domain, the user has to write Java classes which implies that "non-programmers" will not be able to use the system in the desired flexible way. But also for experienced users, the task to build user defined languages might be quite complex due to the necessity of understanding the system internal code structure up to a certain degree.

Approach B: Complete description within the XML file

Here, the classes needed for the content objects in the visual language are compiled at runtime (or, optionally: interpreted). In this approach, the reference frame is only used as an identifier so that the environment can deal with the connection of objects "belonging together" from a domain semantics point of view like a field for formula input and a graph plotting component.

In the description, there are three main parts - the view with the appropriate parameters, a reference to the model type, here MathML, and the controller. The function of the latter here is mainly to give changed content of the object to any "interested" other component, e.g. the function display.

```
<node type="FunctionTermNode"
      frame="functions">
 <view>
  <icon location="FunctionIcon.gif"/>
  <shape type="rectangle"/>
  <color r="255" g="255" b="255"/>
  <input type="string"
         content="x^2"/>
  <size width="100" height="50"/>
 </view>
 <model type="MathML"/>
 <controller>
  <submission time="onChange"/>
  <reception time="never"/>
 </controller>
</node>
```

In this approach, the representation of the display component is impossible as there is a need of parsing the MathML that the input component produces in order to plot the function graph. This is also the main disadvantage here - apart from some imaginable "easy" cases, no interpretation of domain semantics is directly possible.

Approach C: An intermediate solution

The central idea in this approach is to describe the suitable parts of the object in the data file. This will in most cases be the model (that usually has a natural XML representation) and - in some cases - the view. The controlling part is a link to the reference frame which itself points to a method in a class that realises the needed operations on the models, in this case the parsing of the MathML and the plotting of the function values.

```
<node type="FunctionTermNode"
      frame="functions.xml">
 <view>
  <icon location="FunctionIcon.gif"/>
  <shape type="rectangle"/>
  <color r="255" g="255" b="255"/>
  <input type="string"
         content="x^2"/>
  <size width="100" height="50"/>
 </view>
 <model type="MathML"/>
 <controller operation="submitModel"/>
</node>


<node type="FunctionDisplayNode"
      frame="functions.xml">
 <view class="DisplayView.class"/>
 <model type="MathML"/>
 <controller operation="receiveModel"/>
</node>


<frame type="functions">
 <operation name="submitModel"
            class="Parser.class"
            method="setTerm"/>
 <operation name="receiveModel"
            class="Parser.class"
            method="parseTerm"/>
</frame>
```

This approach seems suitable as it combines the advantages of the previously presented ones, usability and flexibility. In order to construct a visual language description that embeds domain dependent semantics, the user can describe most parts of the language in the XML file. Only the *needed* components, here mainly the interpretation of mathematical formulae, have to be written in a programming language - the system-dependent overheads are invisible to the user.

5. Collaboration support

As in many existing environments, our primary support for collaboration relies on the use of shared workspaces. Technically, this is realised using the MatchMaker server [14] offering a replicated architecture, partial synchronisation features and dynamic synchronisation. According to the system structure as outlined in chapter 4.1 and the intended flexibility of the system, the synchronisation of objects is possible in the following different ways, each related to specific use cases and outlined briefly in the following:

Coupling by application

The standard case used in most environments - the complete application (all the workspaces) are synchronised. This is useful e.g. for larger tasks involving different sessions or for different synchronised *views* on a discussion.

Coupling by workspace

In this case, the users can have multiple private and shared workspaces simultaneously which is practical when e.g. being in different user groups at the same time or in order to realise jigsaw designs for co-operative tasks.

Coupling by layer

This way of coupling applications provides e.g. the option to have private annotations on synchronised workspaces. An example with a private handwriting layer is shown in Figure 4.

Coupling by semantic unit

Here, the synchronised objects belong together semantically, either as defined in a reference frame or as selected by the user through the grouping of elements. A characteristic use case of this way of coupling might be the "publication" of a developed model (excluding the private comments) in order to discuss it in a group.

Coupling by object

This is the "lowest" level of synchronisation - here, the synchronised objects "to be discussed co-operatively" can be explicitly defined; any other objects will be private. This mode allows a highly detailed control and definition of the co-operative task and its conditions.

7. Outlook

We believe that shared workspace environments will be more and more integrated with semantic support without giving up the provision of a very general "syntactic" platform. In our current work, we are redesigning and reimplementing the CardBoard environment on a Java basis to support the specific needs of modelling and data analysis in science education. As a next step, we plan to integrate graphically defined models with complete operational semantics (based on system dynamics) with the environment presented above. Another extension focuses on augmenting workspaces with archiving and retrieval functionality.

8. Acknowledgements

Parts of this work refer to the European IST project No. 1999-12017, DiViLab.

9. References

[1] DiViLab. EU funded IST project (no. 12017) Distributed virtual laboratory. *http://www.divilab.org*

[2] Forrester, J. W. (1968). *Principles of Systems*. Waltham, MA (USA): Pegasus Communications.

[3] Suthers, D., Weiner, A., Connelly, J. & Paolucci, M. (1995). Belvedere: Engaging students in critical discussion of science and public policy issues. In Greer, J. (ed.), *Proceedings of the 9th World Conference on Artificial Intelligence in Education* (pp. 266-273). Washington DC (USA).

[4] Gaßner, K., Tewissen, F., Mühlenbrock, M., Loesch, A. & Hoppe, H. U. (1998). Intelligently supported collaborative learning environments based on visual languages: a generic approach. In Darses, F. & Zaraté, P. (eds.), *Proceedings of 3rd International Conference on the Design of Cooperative Systems* (pp. 47-55). Cannes (France).

[5] Streitz, N., Haake, J., Hannemann, J., Lemke, A., Schuler, W., Schütt, H. & Thüring, M. (1992). SEPIA: A coopoerative hypermedia authoring environment. In *Proceedings of the 4th ACM Conference on Hypertext* (pp. 11-22). Milan (Italy).

[6] Buckingham Shum, S. & Hammond, N. (1994). Argumentation-based design rationale: What use at what cost?. In *International Journal Human-Computer Studies, 40,* 603-652.

[7] Gaines, B. R. & Shaw, M. L. G.(1993). Knowledge acquisition tools based on personal construct psychology. In *The knowledge engineering review, 8,* 49-85.

[8] Conklin, J. & Begemann, M. L. (1987). gIBIS: A hypertext tool for team design deliberation. In Proceedings of Hypertext'87 (pp. 247-251). Chapel Hill, NC (USA).

[9] Mühlenbrock, M., Tewissen, F. & Hoppe, H. U. (1997). A framework system for intelligent support in open distributed learning environments. In du Boulay, B. & Mizoguchi, R. (eds.), *Artificial intelligence in education: Knowledge and media in learning systems* (pp. 191-198). Amsterdam (The Netherlands): IOS Press.

[10] STELLA. http://www.hps-inc.com

[11] Rational Rose. http://www.rational.com

[12] Constantino-Gonzalez, M. & Suthers, D. (2000). A coached collaborative learning environment for Entity-Relationship modeling. In Gauthier, G., Frasson, C. & VanLehn, K. (eds.), *Proceedings of 5th International Conference on Intelligent Tutoring Systems* (pp. 324-333), Montréal (Canada). Berlin, Heidelberg: Springer.

[13] Joolingen, W. R. van (2000). Designing for collaborative discovery learning. In Gauthier, G., Frasson, C. & VanLehn, K. (eds.), *Proceedings of 5th International Conference on Intelligent Tutoring Systems* (pp. 202-211), Montréal (Canada). Berlin, Heidelberg: Springer.

[14] Tewissen, F., Baloian, N., Hoppe, H. U. & Reimberg, E. (2000). "MatchMaker" Synchronising Objects in Replicated Software-Architectures". *In Proceedings of 6th International Workshop on Groupware, CRIWG 2000* Madeira, Portugal, 18 - 20 October 2000, IEEE CS Press.