

Visual middle-out modeling of problem spaces

Andrea Valente

*Dept. of Computer Science and Engineering
Aalborg University Esbjerg (DK)
av@aaue.dk*

Abstract

Modeling is a complex and central activity in many domains. Domain experts and designers usually work by drawing and create models from the middle-out; however, visual and middle-out style modeling is poorly supported by software tools.

In order to define a new class of software-based modeling tools, we propose a scenario and identify some requirements. Those requirements are contrasted against features of existing tools from various application domains, and the results show general lack of support for custom visualization and incremental knowledge specification, poor handling of temporal information, and little generative capabilities.

Satisfaction of the requirements proved difficult, and our first two prototypes did not perform well. A new and streamlined prototype is currently under development: it should enable some useful form of middle-out modeling. Application domains will range from visual knowledge representation to object-oriented analysis, and graph rewriting.

1. Introduction and motivation

The formalization of problem space models from informal human knowledge is a complex of activity. Model building is very common in many domains, ranging from analysis and design of software systems, to system engineering, teaching and presenting, and also in the exploration of new formalisms and theoretical concepts; modeling involves knowledge acquisition, its representation and usually requires incremental refinement.

Models are created following different strategies, however the main ones are: bottom-up (e.g. building a GUI from predefined parts), top-down (e.g. defining a detailed ontology to describe concepts in a problem space), or middle-out [2]. As an example of the latter,

consider the creation of a system model with incomplete and under-detailed parts; this partial model is then completed incrementally, by grouping and generalization of its parts into types on one hand, and detailing of part-to-part relationships on the other. Both types and relationships need to be systematically refined, re-structured during the construction and exploration of a model; for this middle-out modeling problem spaces require very flexible type systems [3]. Moreover, in [2,3] the middle-out approach to modeling is described as the most common and natural (for least when software developers are considered).

Modeling is also a *linguistic activity* [7]: when continuous and informal human knowledge about a system is partitioned into discrete elements and system behavior is expressed as discrete steps in time, a *domain specific language* is always created. Concept maps (and *text graphs* [5] in particular) can be used to represent and reason about such languages. If the informal description is expressed in a pictorial way, its discretization will induce a *visual language*.

We argue that modeling, complex and central as it is in many domains, largely lacks software tool support; and it is even more so for visual and middle-out modeling. For example a recent survey of software development teams adopting the AGILE methodologies showed that more than 80% use mainly low-tech support, such as white-boards, when modeling [4].

We propose to take a simple authoring tool (e.g. a simplified kind of MS PowerPoint) and turn it into a visual middle-out modeling tool, taking inspiration from text graphs [5], problem space types [3] and existing knowledge modeling software [6].

To achieve our goal, we first look requirements for a visual middle-out modeling tool. We contrast our findings against existing tools, then shortly discuss two unsatisfactory prototypes that we developed and the general design of a new one that should result in a

working tool. Finally some application domains and conclusion are considered.

2. What should it be like

2.1. A scenario

As a scenario we consider a designer hand-drawing a storyboard to represent informal knowledge about a real-world problem, for example how orders are managed in a restaurant. The storyboard is composed of many frames, and each frame is just an image depicting waiters, customers, dishes, tables, the kitchen, etc. (for storyboarding as a low-fidelity prototyping technique see [1]). Each frame can also be understood as a different state of an entity, instance of a *restaurant_ordering_system* type, and this type (i.e. concept) is being modeled as the storyboard gets detailed.

Various versions of the same entities are drawn in the frames, for instance tables could be present in many frames, but since they are rendered in artistic fashion, other designers might not be able to tell they are all tables. There need to be an explicit act from the part of the modeler, to declare a new *table* type in the model, and relate all drawings representing tables, to that type. This can be done by chopping off each table from its frame, and turn it into a new layer, explicitly linked to the table type. If all tables are later required to have a uniform look, which is typical of technical drawings like construction blueprints, the designer can alter the standard representation of the table type, and the effect propagates to all frames (Figure 1).

In some cases different artistic renderings of an entity in the storyboard might be meaningful: for instance a waiter could be drawn as free, in the process of collecting an order, or serving customers at their table. In these situations the designer is expressing visually the possible states of an entity, and this knowledge also needs to be made explicit in the definition of the *waiter* type. Once the waiter's available states are listed, some constraints might be added, as usual when describing protocols as state machines or transition systems.

Repeating these operations the designer will gradually move from her informal storyboard towards a more structured, layered and constrained list of diagrams. From imprecise, incomplete but aesthetically appealing and meaningful to humans, towards a visually simpler but semantically clearer notation.

The designer is also aware that visually a table contains internal details (e.g. glasses and dishes on top, nearby chairs, menus, size, orientation, ect.), while structurally it is still a single instance. Relationships

between the whole table and its parts (instances of other types) can be explained by cutting apart visual details of the image, and re-group them. The table type will change accordingly and specific types for the different parts will be created; moreover, all tables in all frames are re-structured to reflect the new knowledge of table-as-composite. Perhaps later in the design, drawing new storyboards tables will turn out to require a coarser definition; in that case the table type will be re-defined and the propagation of the change could invalidate some of the storyboards developed. The designer will be aware of that, and simply check and correct manually the inconsistencies.

At the end of her work the designer has a much more formalized storyboard, perhaps still with some pictorial and informal parts in the background of some frames. She has anyway a model of her problem space composed of few, compact and atemporal definitions. It is this model that provides the added value with respect to a purely pictorial storyboard: in fact she could now quickly instantiate a new restaurant-related storyboard, with elements similar to the original one. She has effectively generalized a visual language for her problem space, on top of a single storyboard.

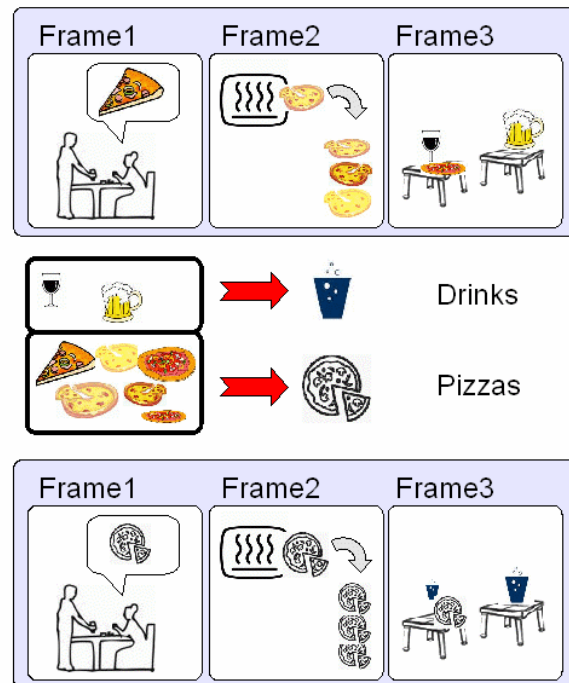


Figure 1: Storyboard for a restaurant ordering system. First frames are artistically rendered, then icons are grouped and formalized into types. The result is a more diagrammatic storyboard.

Using her new visual language, the designer can present her storyboards to non-designers, e.g. domain experts, alter them on-the-fly while discussing with them and her colleagues, and rapidly generate new ones. The visual language is defined by a collection of related types, and this knowledge is very similar to a class hierarchy of an object-oriented system or an ontology: it can be mapped onto the code for a rough prototype of a problem space simulator.

2.2. Requirements

The scenario discussed above involves visual knowledge modeling, middle-out and gradual construction of a model of the problem entities, and knowledge re-structuring with unavoidable insurgence of temporary inconsistencies.

Taking our scenario into consideration we propose some characteristics that a software modeling tool should have:

1. let users customize their visual notation, as in a presentation tool
2. support middle-out thinking, i.e. incremental and *forgiving*, to cope with inconsistencies and re-classifications
3. maintain an explicit and accessible knowledge representation; time should also be explicitly handled (via frames)
4. support generation, i.e. allow use of knowledge to create new stories as needed

3. What is available now

3.1. Related software tools

There are currently many software tools available for the modeler, that support only partially the requirements listed in the previous section.

For us spatial hypermedia and visual knowledge management are of great interest, since they also aim at bridging the gap between real-world problems and formal models. A good representative of the area of visual knowledge tools is Visual Knowledge Builder [9]. It uses *incremental formalization* to simplify the expensive and time-consuming task of defining knowledge. At the official VKB website, we found that VKB:

... allows users to initially enter their understanding of their domain, task, and solutions in less formal representations and provides computer support for the gradual formalization of this knowledge.

Many of the goals of VKB are strikingly similar to

ours. VKB is visual, but the graphic elements at disposal are simple geometric shapes, little artistic expression is possible (first of our requirements). VKB allows users to proceed incrementally from concrete examples of structures, towards more general patterns, type-like in nature. However, VKB does not deal with time as a sequence of frames, and the users knowledge is not explicit, so this tool is only in partial agreement with our third requirement.

Finally VKB seems to be more oriented towards analysis than synthesis: the fourth requirement, generative power, is completely missing.

A number of software tools support knowledge representation by means of *mind maps* and *concept maps*. Although very useful in activities like brainstorming or for presentation of interrelated concepts, we find this kind of tools too *shallow*: there seems to be nothing *behind* the drawings. Usually no knowledge about the domain described by the map is represented in any form inside the tool.

A tool that shows much about the linguistic aspects of modeling is the *text graph editor* [5]; very useful for defining and reflecting on the learning of complex concepts, the strongest limitation that we can see in the *text graphs* is their inability to express the evolution of the formalization: i.e. changes in time are not directly visible in a text graph. Also for the text graph editor, as for the VKB, no generation is possible from the acquired knowledge.

Another very inspirational tool of recent development is Scratch [12]:

... a new programming language that makes it easy to create your own interactive stories, animations, games, music, and art ...

The tool is targeted at young people (from 8 years up), it is visual, with a friendly user interface, and natively handles multimedia. However scratch is a programming environment, and was not constructed with modeling nor knowledge management in mind; moreover time is not expressed directly, as in a frame-based presentation tools. Even so scratch can be used for rapid sketching of storyboards, and covers part of the scenario we presented in section 2.1.

In the domain of graph-rewriting systems [8] some tools exist that could be used for the activities that we are considering, but unfortunately they require a clear and complete initial formalization of both terms and operations in the problem space. The specifications are normally written in tool-specific languages, and the syntax they work with is typically textual and resembles algebraic notations, like calculi.

In the object-oriented community, there are many modeling tools, mostly based on UML or similar diagrammatic notations. For example *BlueJ* [10], a simple Java IDE (Integrated Development Environment) meant for beginners, enables students to create objects on-the-fly, by direct manipulation. The objects are instantiated and visually displayed as boxes in a special *object bar*. Methods can be called on the boxes, and the students can inspect the fields of each instance, via an inspection window that reminds of a visual debugger.

Clearly class-related knowledge is as explicitly represented in this tool (requirement three) and some time-dependent object-level knowledge too, which is related to the first and second requirements. The generation of objects from classes (i.e. stories from knowledge) is also possible, but it is rather volatile: students cannot store the operations they executed on a set of objects as stories, and retrieve them later (this is only partially possible, when creating a unit test from the objects in the object bar [10]).

The whole approach of BlueJ is strongly rooted in design and implementation of object-oriented programs, and will result too rigid, detailed and technical for most of the modeling activities we are interested in supporting. With a related tool, Greenfoot [11], the users can attach icons to classes, and represent objects as active actors in a bi-dimensional world. While more *visual* than BlueJ, also in Greenfoot the knowledge has to be expressed by a cut down UML class-diagrams and detailed code.

In the ontology-building domain, one of the best tools is Protégé [13], developed by Stanford University. With Protégé ontology experts can acquire knowledge about a domain, structure it and instantiate ontologies. The workflow of ontology editors is not usually middle-out, but rather top-down: most of the concepts in the ontology need to be defined in detail before any of the reasoning tools can be deployed. In fact population and maintenance of ontologies are among the most studied and complex tasks in this research area, together with concept re-classification. Being targeted at experts, the user interfaces of ontology editors tend to avoid customizable visual represents, and prefer standardized modeling diagrams and languages (e.g. OWL [17]).

Many techniques for population of concepts exist, such as PANKOW [14] or KnowItAll [15], and in some cases even automatic approaches are shown to perform well. Even if we are not interested in automatic modeling, perhaps some of these techniques could be applied to *suggest* formalization while

middle-out modeling.

3.2. Our prototypes

In the past years, we tried to both precisely define the core problems related to visual middle-out modeling, and to construct a working prototype. Given the complexity of dealing with all our requirements, and especially the second one, about *incremental specification* and *forgiveness*, we feel that even a partial implementation of our new tool should prove of great significance. Unfortunately the tool itself has been proven difficult to implement: we have so far developed two prototypes, that provide important insights circa the architectural needs, the power and user-friendliness of the tool. A central question during design and implementation was how to stay in between a *dumb* graphic editor and a *formal* knowledge management tool.

The key idea of the first implementation is to script every user action, so that she later generalize, formalize whatever she did, and reuse it. The result is a simple visual editor, with few predefined graphic shapes, and only a single frame is available. The tool, coded in Java, is actually Turing-complete, but the visualization cannot be customized (a part from simple grouping, which supports an unwanted bottom-up modeling approach); moreover, the user needs to be a programmer to handle her scripts. We conclude that scripting all actions results in too low-level scripts, difficult to read, to clean and reuse and the focus on modeling is lost in this prototype.

The second prototype is implemented as an HTML page with embedded javascript; it has an even simpler visualization system, reduced to a table of boxes, reminiscent of a spreadsheet. The user can create boxes, each containing an integer value or a string; once created a box has a name, a bi-dimensional position, an initial value, and it exists in all frames of the storyboard. If the box changes position or value (by direct manipulation) in the current frame, the change is propagated automatically in all following frames; at the same time a new assignment command is added at the end of the script generating the current frame. In this way every frame is the result of the execution of a short script, that alters the boxes in the previous frame.

The main point of this prototype is to study how users interact with frames and how semi-automatically maintain more meaningful and higher-level scripts. It works fine for some of the activities in the scenario at section 2.1, but the lack of structured values makes it impossible to cover most of the requirements about knowledge management. There are no objects and no classes, no explicit knowledge representation of

problem space entities, and the user is still too dependent on scripting.

We are currently starting a new prototype that is centered around freehand storyboarding and explicit declaration of classes of visual entities. Users will start creating a project, and within the same project there will be multiple storyboards. Each storyboard in turn will be composed of many frames, representing different states of the system under definition. For each project a single palette will exist, holding a coherent set of related types (i.e. concept definitions), used in all the storyboards and frames of that project.



Figure 2: Design of the user interface of the new prototype. Every project will have one palette and will contain a number of related storyboards.

A project can be considered a (visual) language to discuss a specific problem space, with the palette of types providing all definitions of terms and available events. The user will proceed middle-out, creating and altering definitions in the palette and checking their consistency across all the storyboards.

Artistically drawn icons without internal structure will represent individual object states (i.e. basic values), while structured diagrams will represent composite objects. This decomposition of icons into an aggregation of sub-icons should be the graphic equivalent of what is done textually in the text graph editor [5].

In our tool, a type will be a summary of what a group of values can do over time. Consider a bank-related project with a storyboard that depicts an empty bank account as a skinny piggy-bank, and after some money is deposited the piggy-bank looks fatter. It will be possible to select all icons representing piggy-banks in different frames a define a type in the palette (Figure 2). The *bank-account type* will show a summary of all possible states of an account instance, i.e. all piggy-bank icons. Events can possibly be drawn to connect the states, to express constraints about instances evolution over time: for example the skinny piggy-bank should be the initial icon for any instance of the bank account type, and the fatter icon can only occur

after a *deposit-event* has happened.

We believe that to correctly support middle-out modeling, the new tool will need to cope with the duality between objects changing in time and a-temporal type definitions.

Considering all aspects of a visual middle-out tool, we think that when fully implemented and functional our latest prototype should be relevant for a number of application areas, among which:

- visual knowledge management
- object-oriented analysis and design
- algorithm animation and rewriting-based calculi
- teaching with interactive presentations
- language-oriented programming [16]

4. Conclusion and future work

We argue that visual and middle-out modeling are not properly supported by software tools; in fact in many occasions developer teams and analysts rely on low-tech tools.

In order to define a new class of software-based modeling tools, we propose a scenario and from that extract some requirements. Our requirements are contrasted against features of existing tools from various related areas like visual knowledge representation, object-oriented analysis, visual programming, graph rewriting and ontology construction. We found general lack of adequate support for custom visualization and incremental knowledge specification, poor handling of temporal information, and little generative capabilities: most of the tools have mainly an analytic purpose.

Developing software that supports middle-out modeling turned out to be an elusive task. The two prototypes we created so far proved insufficient to meet our requirements, but we believe that they provided valuable insights on what the new kind of modeling tool could do. The latest prototype, currently under construction, will be similar to a simple presentation tool and have a palette of concepts, common to all entities in all slides of a presentation. We believe that such a simplified application should be enough to enable some useful form of middle-out modeling.

Future work includes the implementation of a complete and working modeling tool, and the old and new prototypes need more testing with problems from different fields, to better assess their expressiveness and usability.

- [1] Helen Sharp, Yvonne Rogers, Jenny Preece, *"Interaction Design: Beyond Human-Computer Interaction"*, 2nd edition (March 23, 2007), Wiley.
- [2] L. S. B. Raccoon and Puppydog P. O. P.: "A middle-out concept of hierarchy (or the problem of feeding the animals).", *SIGSOFT Softw. Eng. Notes* 23, 3 (May. 1998), pages 111-119.
- [3] William Van Lephien, Kenneth M. Anderson: "Extending types to modelling problem-space entities", *Journal New Review of Hypermedia and Multimedia*, volume 12, issue 2, December 2006, pages 143-164.
- [4] Scott W. Ambler, "Agile Adoption Rate Survey: March 2007", *Online article*, visited September 3rd 2008.
- [5] Esko Nuutila and Seppo Törmö, "Text Graphs: Accurate Concept Mapping with Well-Defined Meaning", *Proc. of the First International Conference on Concept Mapping*, Pamplona, Spain, Sep. 14-17, 2004, volume 1, pages 477-485.
- [6] Shipman, F. M., Hsieh, H., Maloor, P., and Moore, J. M., "The visual knowledge builder: a second generation spatial hypertext.", in *Proceedings of the Twelfth ACM Conference on Hypertext and Hypermedia*, Aarhus, Denmark, August 14 – 18, 2001, HYPERTEXT '01. ACM Press, New York, NY, pages 113-122.
- [7] Luiz Marcio Cysneiros and Julio Cesar and Sampaio Prado Leite, "Non-functional requirements: from elicitation to conceptual models", in *IEEE Trans. On Soft. Engineering*, 2004, volume 30, pages 328-350.
- [8] H. Ehrig, G. Engels, H.J. Kreowski & G. Rozenberg, "Handbook of graph grammars and computing by graph transformation. Volume 2: Applications, Languages and Tools", World Scientific Publishing Co., Inc. (1997).
- [9] F. Shipman, J.M. Moore, P. Maloor, H. Hsieh, and R. Akkapeddi, "Semantics Happen: Knowledge Building in Spatial Hypertext", *Proceedings of the ACM Conference on Hypertext*, 2002, pages 25-34.
- [10] Patterson, A., Kölling, M. and Rosenberg, J., "Introducing Unit Testing With BlueJ", *Proceedings of the 8th conference on Information Technology in Computer Science Education (ITiCSE 2003)*, Thessaloniki, 2003, pages 11-15.
- [11] Michael Kölling and Poul Henriksen, "Game Programming in Introductory Courses With Direct State Manipulation.", *Proceedings of ITiCSE'05*, Lisbon, Portugal, June 2005, pages 59-63.
- [12] J. Maloney, L. Burd, Y. Kafai, N. Rusk, B. Silverman, M. Resnick, "Scratch: a sneak preview", *Proceedings of the Second International IEEE Conference on Creating, Connecting and Collaborating through Computing*, 2004, pages 104- 109.
- [13] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, and S. W. Tu, "The evolution of Protégé: an environment for knowledge-based systems development.", *International Journal of Human-Computer Studies archive*, volume 58, issue 1, Jan. 2003, pages 89-123.
- [14] P. Cimiano, S. Handschuh, S. Staab, "Towards the self-annotating web.", *Proceedings of the 13th World Wide Web Conference*, pages 462-471, 2004.
- [15] O. Etzioni, M.J. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D.S. Weld, A. Yates: "Web-scale information extraction in KnowItAll (preliminary results).", *Proceedings of the 13th World Wide Web Conference*, pages 100-110, 2004.
- [16] Martin Bravenboer, Arthur van Dam, Karina Olmos, Eelco Visser: "Program Transformation with Scoped Dynamic Rewrite Rules", in *Fundamenta Informaticae*, IOS Press (2006), volume 69, number 1-2, pages 123-178.
- [17] W3C, "OWL Web Ontology Language", at <http://www.w3.org/TR/owl-features/>, visited September 9th