In Procs of the OOPSLA'93 Workshop on Visual Object-Oriented Programming, Washington DC, 47-48, October, 1993.

# A Visual Programming Environment for an Object-Oriented Prolog

John C. Grundy
Department of Computer Science
University of Waikato, Hamilton, New Zealand
jgrundy@waikato.ac.nz

John G. Hosking
Department of Computer Science
University of Auckland, Auckland, New Zealand
john@cs.aukuni.ac.nz

The Snart Programming Environment (SPE) provides a visual programming and program visualisation environment for Snart, an object-oriented Prolog. SPE supports graphical and textual views of Snart software development with full consistency management between all views.

## 1. Rationale for Research

Both graphical and textual representations of object-oriented programs are useful. Graphical views of object-oriented software development support: analysis-level diagrams (such as major class generalisation relationships); design-level diagrams (such as method calling protocols); browser support during implementation; and both static and dynamic program visualisation. Textual views are generally most appropriate for detailed class interface specification, method implementation, and detailed class documentation.

Integration of these textual and graphical views allows programmers to choose the most appropriate representation for a phase of software development. It also allows reuse of views during different phases of development (for example, using analysis-level hierarchy views for program browsing or static visualisation). This view integration requires an environment to maintain view consistency automatically, so software developers are not presented with incorrect information and do not try to modify incorrect program data.

## 2. The Snart Programming Environment (SPE)

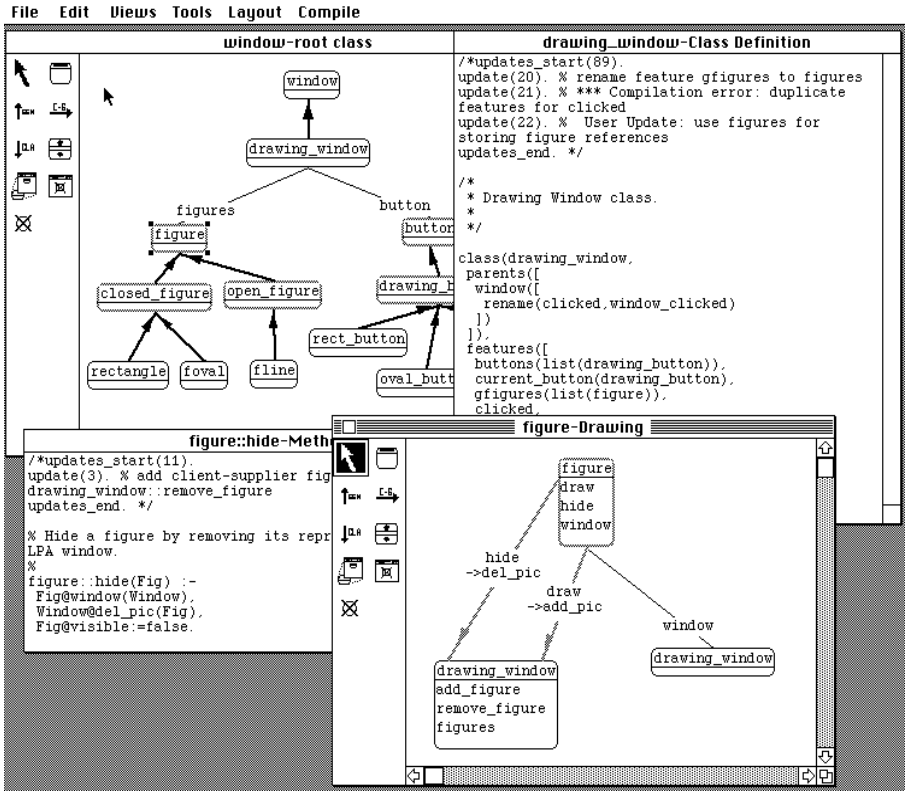


Fig. 1. An example of SPE views during the development of a simple drawing program.

Fig. 1. shows a screen dump from SPE during the development of a simple drawing editor program. One graphical view shows an analysis-level diagram representing important generalisation and aggregation structures. The second shows a design-level diagram describing method calling protocols when rendering figures in a drawing window. One textual view shows a detailed class interface for the `drawing_window` class while another shows two method implementations.

Graphical views are edited interactively using a tool palette, menus and dialogues. Textual views are free-edited and then parsed to update shared program information. This contrasts to other systems which employ restrictive structure-oriented editing approaches, which have yet to gain general acceptance by most programmers. Graphical view composition and layout is programmer-defined, allowing diagrams to have the most desired appearance. Shared information can be expanded into views, automatically laid-out and then repositioned as required.

Graphical view consistency is maintained by updating graphical structures directly. Deleted program components are rendered in a different colour to allow programmers to determine how the view should be modified. Textual view consistency is achieved by unparsing update records into a header comment for view components (as shown in Fig. 1.). Programmers can have SPE automatically apply some updates to the component's text, implement the update themselves, or reject the update.

This technique allows modification to information not directly updateable to be indicated. For example, a client-supplier connection added between two classes in a design-level diagram can not be directly applied to a method implementation textual view. Programmers are informed of such a change in the textual view via an update record, however, and can then add an appropriate feature call (see Fig. 1.). Update records in SPE also support compilation error reporting, user-defined update records, and a component modification history. Update records for graphical view components can be stored and shown in a dialogue if this is desired.

SPE supports high-level analysis of object-oriented software, detailed design diagrams, implementation using textual views, and maintenance at any level. Textual and graphical views can also be constructed to document and browse a software system, and be used for static program visualisation to reduce the cognitive complexity of software. Cerno, a visual debugger for SPE, has also been developed which supports dynamic program visualisation using object diagrams.

## 3. The MViews Model

SPE is implemented using MViews, which provides a model and object-oriented framework for constructing integrated software development environments (ISDEs). Use of multiple textual and graphical views of information with consistency management is common to most ISDEs. A set of abstractions for modelling and constructing these multi-view editing facilities thus aids development of ISDEs.

MViews uses object dependency graphs to represent the structure and static semantics of programming languages and software systems. Software is described using program graphs, comprised of elements and relationships, and a base view uses these program graphs to form a canonical software system representation. Subset views of this base view are constructed which represent partial views of software development. Subset view components are rendered and edited in graphical or textual forms using display views.

Components are modified by graph operations, which generate update records describing the exact change to a component. Components have zero or more dependent components, and when modified broadcast update records to their dependents. Update records are used by MViews to maintain textual and graphical view consistency, implement a generic undo/redo facility, document a component's modification history, provide graphical component constraints, support attribute recalculation for language semantics, and support lazy update record processing and update record composition.

MViews provides specification languages for describing the structure and modification semantics of an environment and for visually describing the appearance of display views, display view components and dialogues. An object-oriented architecture allows a new environment to be designed using MViews abstractions and a framework of Snart classes is specialised to implement the environment.

## 4. Current and Future Research

SPE is being enhanced to support more analysis and design views, including dataflow analysis, inter-feature communication diagrams and textual class contracts, and to include dataflow diagram method implementation. Cerno is being enhanced to support dynamic visualisation view generation from class diagrams and vice versa.

MViews has been reused to produce an entity-relationship diagrammer with textual relational database schema, a dialogue painter with textual constraint specification views, and SPE and Cerno have been reused to define a common building model editor. MViews is being enhanced to support flexible version control with update records and to provide distributed, shared software system representation. We plan to extend SPE with these facilities to support distributed, collaborative object-oriented software development.

**John G. Hosking: Brief Biographical Sketch**

**Position and Affiliation**: Senior Lecturer in Computer Science, University of Auckland, Auckland, New Zealand

**Email:** john@cs.aukuni.ac.nz

I have been working actively in the development of object oriented languages and systems for almost 10 years, and visual programming environments for object oriented programming for the last 4 years. I am participating in the following on-going projects, together with colleagues Rick Mugridge, John Grundy, John Hamer, Robert Amor, Stephen Fenwick, and Shaun Blackmore:

**MViews:** an objedct-oriented framework for developing integrated software development environments. this provides base support for multiple textal and graphical views with consistency between views. MViews has been used in the development of SPE, Cerno, and our CBM work (see below); an entity relationship diagrammer (which generates and maintains consistency with relational schema); and a dialog box editor (which, again, maintains consistency with a textual dialog box specification language). MViews is also being used by another group in the development of HyperPascal, a Pascal-like procedurally-oriented visual language.

**SPE:** the Snart Programming environment (described in the extended abstract). This is derived from Mviews by specialising the MViews framework in two steps. The first step is the creation of a generic visual OO Programming environment called IspelM. The second is specialisation of IspelM for programming in Snart, an OO Logc programming language (see below).

**Cerno:** Cerno is a visual debugging system for Snart (although it could be adapted for use by other OOPLs) which is also implemented using the MViews framework. Cerno provides multiple views of an executing Snart program, each kept consistent with the underlying program execution state. Views contain icons representing objects and their state (which can be modified using Cerno), together with connectors representing relationships between objects, such as feature references. The information in object icons can be quite flexibly defined, eg all features satisfying specified criteria, and layed out using either predefined object view types (such as a list of features + values, or a bar graph for collections of numbers), or by defining new types using a simple, but powerful and extensible, icon layout language. Predefined view layouts, or "templates" can be created to allow interesting program views to be reused during other executions of the same program. Current extensions include the ability to translate from templates into SPE class diagram views and vice versa; the creation of "replayable" program animations, and appropriate methods for visualising flow of control.

**Kea:** Kea is a strongly-typed object-oriented functional language. Its execution environment incorporates a one-way consistency manager, which propagates the effects of changes to user input. This permits programmers to program as if inputs are unchanging, yet have changes automatically coped with by the environment. This mechanism is combined with an external interface model, providing a per-object interface, and X-Window based forms and plan-drawing systems. The result is something similar to facilities provided by Garnet/KR, but implemented in a somewhat "cleaner" fashion. In addition, Kea incorporates classifiers, which provide a mechanism to dynamically extend the class membership of an object after creation in a controlled and type-safe fashion.

**Snart:** Snart is an object-oriented extension to Prolog. The extensions are fairly imperative in nature, providing objects with methods and assignable state and code which integrates cleanly with standard Prolog predicates. Snart was originally developed both to implement MViews, and also to act as an example language to develop an MViews-based environment for (resulting in SPE). However, Snart is an interesting language in its own right. It incorporates an imperative version of the Kea classifier mechanism; support for multiple class and object spaces; and support for persistency and object tracing. The latter two make use of the classifier mechanism to transparantly turn an object into a persistent and/or tracable one. Snart is also used as the basis for developing a constraint-based OOPL using a novel local propagation algorithm.

**Systems for code of practice compliance checking:** We have developed a number of large OO systems using Kea for assisting building designers to check compiance of their designs against construction codes and regulations. The needs of these systems motivated both the development of Kea, the visual programming environment work, and the Common Building Model work (see below).

**Common building model:** Arising from the code compliance work, we are applying some of the results from our programming environment work in another domain, that of computer integrated building construction. We are attempting to provide a model allowing multiple building industry professionals (Engineers, Architects, etc) to collaborate in the design of a building, communicating results via a centrral repositry of information. This project brings together most of our other work: an SPE variant is being used to describe/program the common building model and the views of that model required for individual professionals and diesign tools; Snart is being used to implement the models and the intermodel communication; a variant of Cerno is being used to view and edit instances of the building according to the different model requirements; the constraint-based OOPL is being used to write a flexible OO Plan Drawing system; and we hope to interface some of the Kea based tools for use with instances of the common model.

**Selected publications:**

Grundy, J.C. and Hosking, J.G.: Constructing multi-view editing environments using MViews, accepted for publication in *Proc. IEEE Symposium on Visual Languages*, to be held in Bergen, Norway, August 1993.

Grundy, J.C. and Hosking, J.G.: Integrated software development in SPE, accepted for publication in *Proc. 13th New Zealand Computer Society Conference*, to be held in Auckland, August 1993.

Amor, R.W., and Hosking, J.G.: Multi-disciplinary views for integrated and concurrent design, accepted for publication in *Proc Management of Information Tecnology for ConstructionFirst International Conference*, to be held in Singapore, August 1993.

Hosking, J.G. , Hamer, J. and W.B. Mugridge: Classifiers in OO Languages, *TOOLS'92 Workshop*, UTS, Stdney, December, 1992.

Grundy, J., and Hosking, J.G. Mviews, A framework for developing visual programming environments, *Technology of Object-Oriented Languages and Systems TOOLS 9 Proc of the 9h International Conference Sydney*, Prentice Hall, 129-137,1992.

Amor, R.A., Hosking, J.G., Groves, L.J., and Donn, M.R. : Design tool integration: model flexibility for the building profession. *Proceedings Building Systems Automation-Integration 1992 Symposium: Computer Integration of the Building Industry*, Dallas, Texas, June,1992.

Amor, R.A., Hosking, J.G., Mugridge, W.B., Hamer, J., Williams, M.: ThermalDesigner: an application of an object-oriented code conformance architecture. *Proceedings CIB Joint International Workshops on Computer Integrated Construction and Computers and Building Standards*, Montreal, Canada, May, 1992.

Grundy, J., Hosking, J.G. and, Hamer, J.: A visual programming environment for object-oriented languages, in Korson, T., Vaishnavi, V., and Meyer, B. (eds) *Technology of Object-Oriented Languages and Systems TOOLS 5 Proc of the 5th International Conference Santa Barbara*, Prentice Hall, 129-138,1991.

Mugridge, W.B., Hamer, J. and Hosking, J.G.,: 'Multi-Methods in a Statically-Typed Programming Language', in America, P (ed) *Lecture Notes in Computer Science 512: ECOOP '91 European Conference on Object-Oriented Programming*, Springer-Verlag, pp 307-324, 1991.

Hosking, J.G., Mugridge, W.B., Hamer, J.: An architecture for code of practice conformance systems, in Kahkonen and Bjork (eds) *Computers and Building Regulations*, VTT Symposium 125, VTT Espoo, Finland, pp171-180, 1991.

Hosking, J.G., Hamer, J. and W.B. Mugridge: Integrating functional and object-oriented programming, in Bezivin J., Meyer, B., Potter, J., and Tokoro, M. (eds) *Technology of Object-Oriented Languages and Systems TOOLS 3 Proc of the 3rd International Conference* , Sydney, 1990, pp. 345-355.

**Selected technical reports:**

Mugridge W.B. and Hosking J.G. : Object-oriented schema integration for a common building model, Auckland Uniservices Ltd, March 1993, 40pp.

Fenwick, S.P. and Hosking, J.G.: Visual debugging of object-oriented systems, Auckland Computer Science Report No 65., Department of Computer Science, University of Auckland, 1993.

Hamer, J., Hosking, J.G. and W.B. Mugridge: Static subclass constraints and dynamic class membership using classifiers, Auckland Computer Science Report No 62., Department of Computer Science, University of Auckland, 1992.