A Persistent Hyper-Programming System

Graham Kirby, Ron Morrison & Dave Munro
Division of Computer Science
University of St Andrews
North Haugh, Fife KY16 9SS, UK
{graham, ron, dave}@dcs.st-andrews.ac.uk

Richard Connor & Quintin Cutts
Department of Computing Science
University of Glasgow
Lilybank Gardens, Glasgow G12 8QQ, UK
{richard, quintin}@dcs.glasgow.ac.uk

Abstract

We demonstrate the use of a hyper-programming system in building persistent applications. This allows program representations to contain type-safe links to persistent objects embedded directly within the source code. The benefits include improved efficiency and potential for static program checking, reduced programming effort and the ability to display meaningful source-level representations for first-class procedure values. Hyper-programming represents a completely new style of programming which is only possible in a persistent programming system.

1. Hyper-programming

Persistent programming languages were developed in an effort to reduce the burden on the application programmer of organising the transfer of long-term data between volatile program storage and non-volatile storage. Previously, application data which was to be retained between activations had to be written explicitly to a database or file system, and later read in again to the application space. The flattening and rebuilding of data structures that this required involved a significant programming overhead, and an increased intellectual effort since the programmer had to keep track of a three way mapping between program representation, database/file representation and the real world. The introduction of orthogonally persistent languages meant that any program data could be made persistent simply by identifying it as such, with all transfers between memory hierarchy layers handled transparently.

The treatment of source programs as strongly typed persistent objects, which is made possible by the use of a Persistent Object System (POS) as the support platform, permits a new approach to program construction. Hyper-programming involves storing strongly typed references to other persistent objects within a source program representation. Thus the source code entity is represented by a graph rather than a linear text sequence. By analogy with hyper-text this is called a hyper-program. It may be considered as similar to a closure, in that it contains both a textual program and an environment in which non-locally declared names may be resolved. The difference is that now the environment is explicitly constructed by the programmer who specifies persistent objects to be bound into the hyper-program at construction time.

The support of hyper-program construction techniques by a POS provides a number of advantages:

- Program succinctness: textual descriptions of the locations and types of persistent components used may be replaced by simple embedded references.
- Increased execution efficiency: checking the validity of specified access paths to other components is factored out when they are embedded directly in the source program. Checking of type consistency may be performed at compilation time rather than execution time.
- Reliable access to components: where a textual description of a component is replaced by a direct reference, the underlying referential integrity of the POS ensures that the component will always be accessible by the program. By contrast, where a textual description is used it may be invalid by the time the program executes, even if it was valid when the program was constructed.
- Automatic source code retention: the hyper-program notation may also be used to represent procedure closures,with encapsulated state. This source representation can be recorded by the POS when the procedure is created, and permanently associated with the procedure by recording a reference to it in the closure value.

We demonstrate a prototype hyper-programming system based on the Napier88 persistent language. At its core are a hyper-program editor and an object browser. The programmer uses the browser to locate objects of interest in the persistent store, and then embeds links to those objects into the hyper-program under construction. The editor displays buttons within the program text to denote the links; when a button is pressed the corresponding object is displayed by the object browser. We show how these techniques may be used to construct and alter persistent applications.

2. Conclusions

The demonstration illustrates the new technique of hyper-programming which has been made possible by implementing a programming environment entirely within a POS. Papers on the subject and details of how to obtain the prototype system are available at:

```
http://www-ppg.dcs.st-andrews.ac.uk/
```