

The SIMPLE Language

Robert M. Akscyn
Computer Science Department
The University of Waikato, New Zealand
ra33@cs.waikato.ac.nz

SIMPLE is a hypermedia-based programming language designed for extending the functionality of hypertext-based applications, such as web browsers. The language uses the concept of a hypertext link as an intrinsic part of the language, allowing software systems to be represented as multi-level hierarchies of hypertext nodes, which are only accessed by running processes as they are dynamically needed.

The language is embedded within the Expeditee Spatial Hypermedia system: a follow-on to the ZOG and KMS hypertext systems (developed 1972 through 2006), and is now being developed at the University of Waikato in New Zealand.

The prime goal of the SIMPLE system (language, plus Expeditee-based interpreter) is to reduce the complexity of programming to the point where average users can program. The motivation for this goal is to support the needs of individual users to customize their work environment to better suit the suite of tasks they perform. The ability to customize is essential to the Expeditee approach to knowledge management, in part because it attempts to fold most applications (email, document processing, spreadsheets, programming, etc.) within a single conceptual data model: spatial hypermedia.

To address this goal, the SIMPLE language has only one construct: a procedure call. All language statements, including those added by the user, are procedure calls; single-line commands (i.e., the procedure name followed by its parameters).

Many of the concepts of modern day programming languages, most especially those found object-oriented programming languages are absent by design. For example, notions such as classes, objects, functions, user-defined data types, even the concept of "a program" are not part of the conceptual model for SIMPLE. All variables are reference variables (there is no "call by value").

The rationale underlying the design of SIMPLE is based on several decades of experience developing hundreds of applications in ZOG and KMS. A particular finding from that experience is that users find it easy to work with a large lexicon of terms (rich semantics) when the programming language syntax is kept ultra-simple (as that is how natural languages are structured).

For example, there are no punctuation symbols in SIMPLE such as semicolons, parentheses, square brackets, or curly braces. Thus there is no syntactic

nesting; all whole/part relationships being represented explicitly by links.

All SIMPLE variables begin with the character "\$" followed by a one- or two-letter code for data types ("b"=boolean, "c"=character, "i"=integer,), then a period, and finally whatever name the user chooses to refer to the variable.

An example SIMPLE statement is:

```
"CreateItem $fp.Cur $ip.Cur $i.X $i.Y $s.Text $s.Action"
```

which has the effect of creating a new text item (with an action) located at coordinates X,Y in frame \$fp.Cur (already 'opened') and then assigning the newly-create item to the variable \$ip.Cur.

The SIMPLE interpreter can begin execution code at any hypertext node (called a "frame", following the use of that term in ZOG and KMS). Top-level frames are typically dominated by pseudo-code comments representing high-level abstractions, with the leaf and near-leaf nodes making use of the language intrinsics.

All statements in a code frame are executed, and if any have links to subordinate frames, those are further processed in a depth-first fashion. The one principal exception to this regime is for statements that express conditionality (e.g., the "IF" statement). In the case of IF (and several dozen variants of "IF") the link to subordinate frames (if it exists) is only followed if the conditional expressed in the statement is true.

Current implementations of the SIMPLE interpreter (written in Java) access random frames with an average response time less than 1/20th of a second. Statements are accessed, parsed and implemented on the scale of 40,000 statements per second.

Expeditee programs are often small (a handful of frames), start execution within a 1/20th of a second, and usually finish within seconds. By contrast conventional programs are often still trying to load themselves at the several seconds mark. Thus the functionality of SIMPLE code is more often down at the level of features (quickly get in, get out) than at the level of whole programs, which, by contrast typically subsume dozens to hundreds of features.

As with ZOG and KMS, the system is designed to be shared by multiple users, who can collaborate on developing software, akin to code repositories like CVS and SVN (and their accompanying IDE's, such as Eclipse).

A second prime objective of SIMPLE is scalability: supporting the development of large-scale structures of code. Like web sites, Expeditee databases (and thus SIMPLE software systems) can grow arbitrarily large. We have tested systems with over ten million procedures without noticing any significant increase in response time (i.e, average response time is still under 1/20th of a second).

A canonical use of SIMPLE programs is to process a tree of frames that represents some system under design/development (code, sections of documents, data) and convert the private contents to other formats that are then used by specialized programs (e.g., LaTeX). The underlying motivation of this approach is to shield the user from the complexity of application systems, allowing them to instead concentrate on the logical structure and substance of their knowledge work (rather than the busy work of syntax and fussing with formats).

Currently the SIMPLE language implements over 200 of the nearly-900 intrinsic commands in the KMS "action language". As one would expect this re-implementation takes advantage of the lessons learned from the ZOG/KMS experience and thus new opportunities for a number of additional simplifications are being explored. This poster will illustrate the history, structure, and current research results of the SIMPLE language, including demonstrations of large-scale programmed systems with hundreds of millions of lines of code.