ALSO: A Language for Extensible Multi-user Systems

Andrew C. Myers
andru@lcs.mit.edu
MIT Laboratory for Computer Science

1 Introduction

The language ALSO is designed to support safely extensible multi-user servers. It was originally designed for writing MUD servers (programmable multi-user virtual worlds), but it is also applicable to other servers. For example, it provides much of the key functionality of languages like Safe-Tcl and JavaScript. ALSO supports even more flexible MUD systems than MOO [1], and can be used to implement other kinds of servers too (e.g., HTTP, SMTP). ALSO contains several uncommon or unique features that make it easier to safely extend a running system. This note provides an overview and rationale for ALSO'S design. More details are available [2]. A running ALSO server can be experimented with at telnet://also.lcs.mit.edu:4201.

2 Overview

ALSO is designed to be familiar in those areas where being different yields no benefit. For example, its syntax is similar to that of C++ and Java, though there are some compatible extensions (e.g., semicolons are optional and it supports gcc-like block expressions). ALSO has first-class lambda expressions and lexical scoping, but functions can be conveniently declared in a C-like syntax. The language is not statically typed. Functions may return multiple results, and may return either normally or with an exception. The following example is the Fibonacci function:

```
fib(x) = (
  if (x < 0) fail(range-error, x)
  if (x < 2) 1
  else fib(x - 1) + fib(x - 2)
)
```

Computation is typically performed on objects. Like SELF [3], ALSO has prototype-based inheritance; that is, objects inherit directly from other objects. Typically, *exemplar* or *template* objects fill the role of classes in class-based systems. Objects can contain references to each other, and are stored in a garbage-collected heap.

Andrew Myers is supported in part by ARPA contract N00014-91-J-4136.
Web: `http://www.pmg.lcs.mit.edu/~andru`

ALSO is currently executed by a fairly fast interpreter (about as fast as Perl), but the execution model is compatible with compilation. In particular, it has an eval operator that, like the same operator in Scheme and Lisp, explicitly compiles program text into code. Only eval can add new code to a running server.

This section has discussed similarities between ALSO and some popular programming languages. Now we will consider some differences.

3 Identifiers

Most interpreted languages (e.g., Scheme, Tcl) have a *read-eval-print* loop that inserts identifier bindings into a global namespace. A global namespace is inappropriate for a multi-user system, since it leads to name clashes — users may want different names for the same objects, or the same names for different objects. An extreme example is users who speak different languages, but the ALSO model provides support for even this case.

In ALSO, each user (in fact, each separate invocation of eval) can supply their own naming environment for interpreting identifiers in code. This environment is then used consistently for interpreting and reporting identifiers. The naming environment can be an object that performs arbitrarily complex computation to locate identifiers, which provides a powerful hook into the workings of the compiler.

Since ALSO is designed for extensible systems, it is always possible to examine compiled code (i.e., function values) in source form, using the sprint operator. This operator can report the code using identifiers consistent with the user's naming environment, which is possible because compiled code contains a precis of the naming environment used to compile it. An inverse naming environment, supplied to sprint, then can be used to back-map identifiers mentioned in the precis.

There are limitations to customized name environments, of course. Because ALSO is lexically scoped, local variables are not mapped using the compilation environment. However, there is no ambiguity of interpretation in this case.

4 Transactions

Requests for different users are processed by an ALSO server in separate, serializable transactions, so they do not interfere with one another. ALSO allows nested transactions, where the failure of an inner transaction does not necessarily affect the containing transaction. Transactions are integrated with exceptions in a convenient and unobtrusive manner. The "try" statement may be used to execute any block of code within a nested transaction. For example, try (x = 1/0) will attempt to divide 1 by 0. Since the contained code will terminate with an uncaught exception (divide-by-zero), the transaction has no effect except to return any values attached to the exception.

Since run-time errors and access control failures are treated as exceptions, this model makes it easier to run code that may contain bugs. If the code encounters an unexpected run-time error, it will generate an exception that terminates the current transaction. Running this code is therefore less likely to leave objects in a partially-modified or inconsistent state. Transactions provide an additional level of confidence about importing untrusted or possibly buggy code.

In MUDs, other users routinely extend the system by adding new code that provides behavior for items in the virtual world. Interacting with these items often causes one to invoke the code, often at unexpected times. Transactions are particularly helpful in this situation, but no other MUD language supports nested transactions. Transactions may be helpful for extensible web servers, too.

5 Objects

As the only mutable data type in ALSO, objects are very important for programming. However, the object model has some unusual features. In virtually all object-oriented languages, an object is a map from symbols to values. An object (or class) is a local namespace, and symbols are used to look up methods and instance variables. To satisfy the goals of Section 3, ALSO takes a unique approach: Objects map from *values* to values. The identifiers used to name components of an object may vary from user to user, so objects do not impose a namespace on their users. In addition, since objects are implemented as hash tables, they conveniently supersede arrays, associative arrays, and sets.

An object is a collection of mutable *properties*, which are key/value pairs. A property key may have any type; typically, property keys are themselves objects. The compilation environment contains the user's mappings from identifiers to property keys, so different users may use different identifiers to access the same properties. When a property key is an object, other information may be associated with it, such as documentation or an ACL.

Objects can also be used to implement modules: encapsulated implementations that provide a public interface. The public interface is accessed through properties, whereas private module identifiers are stored in local variables captured by a closure. For example, the following code implements a module with two public procedures, incr and value, that access an internal counter:

```
counter is (
    var cnt = 0    // private!
    new (
      incr() = (cnt = cnt + 1),
      value() = cnt
    )
)
```

counter is bound to the result of the new expression, which uses cnt from the same scope. External code can access cnt only through counter.incr() and counter.value(). Evolution of the system is then supported by the mutability of counter (subject to access control constraints).

6 Queries

ALSO supports a simple query model in the form of *invertible properties*. If a property key is designated as invertible, then the "?" operator may be used to find all objects that map that key to a particular value. Common object properties like location, type, and owner are usually invertible. For example, the expression owner?andru would produce a list of all objects owned by andru. Querying is fast, because the current ALSO implementation precomputes all query results, incrementally updating its tables as assignments are performed to invertible properties.

This simple query model has greatly simplified existing code because it supports one-to-many relationships. For example, MUD systems often find all objects in a particular location, or all objects of a particular type. Generally these two particular requests are hard-coded into the system, denying wider use of the functionality. The "?" operator makes this kind of request robust and efficient.

7 Future Work

ALSO provides several uncommon or even unique features that are well-integrated and aid writing robust code in a multi-user environment. However, more work is required for wider acceptance of ALSO. The access control model is currently being refined, and performance, while not bad, is being improved by the addition of on-the-fly compilation. A good debugger and an integrated GUI toolkit such as Tk would also help.

References

[1] Pavel Curtis. *LambdaMOO Programmer's Manual*. Available at `ftp://parcftp.xerox.com.edu/pub/MOO`, May 1996.

[2] Andrew C. Myers. ALSO*: The Language*. Available at `ftp://ftp.pmg.lcs.mit.edu/pub/also/-manual.ps.gz`, March 1994.

[3] David Ungar and Randall B. Smith. SELF: The power of simplicity. In *OOPSLA '87 Conference Proceedings*, October 1987.