Annual Report for Contract Number N00014-88-K-0641

For the period: 1 October 1987 - 30 September 1988

1Oct87 - 30 Sept88

ONR Graduate Fellowship
N00014 - 88 - K - 0641
Dr. Andre van Tilborg

Miró Research Description
Mark Maimone

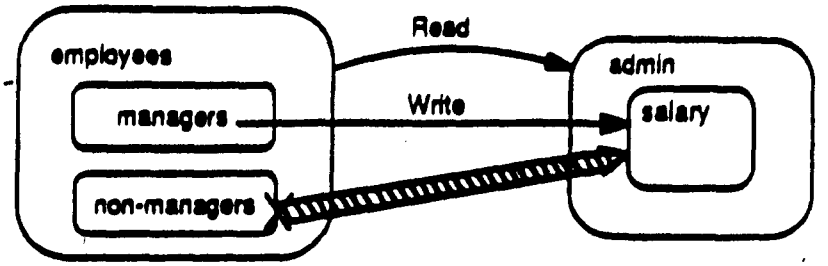
1 Introduction

The heart of the proposed research lies in the formal specification of complex software systems. We are interested in specifying not just the functional correctness of a system, but also its behavior imposed by concurrency, fault-tolerance, security, and real-time constraints. Current specification techniques are inadequate for describing such behavior for realistic, large-scaled systems. We need to combine isolated research results from the areas of formal specifications and formal models of concurrency, and more significantly, to extend them in order to specify properties that are as critical as functional correctness.

The novel aspect of this research is to exploit the benefits of visual languages to specify system behavior. We intend to design a visual specification language; to give a formal semantics to the language; to build a rich set of tools for presenting and manipulating visual specifications; and to demonstrate the suitability of the language to the specification of a wide class of system properties.

2 Language Design and Semantics

Our present design allows users to draw different types of *boxes* and *arrows*, in the formation of *pictures*. A box denotes a set of objects and an arrow between boxes denotes a relation which holds between the objects in those boxes. The same visual notation may be given different interpretations depending on the application domain of interest. The picture below, interpreted in the security domain, shows three boxes denoting sets of users, two boxes denoting sets of files and three arrows denoting different access right relations.



Here all employees have read access to administrative files, some managers have write access to salary files and no non-manager has any kind of access to salary files (assuming that the cable represents any access type). There are still many open language design issues including: finding appropriate visual notation for data types, eliminating naming problems (such as aliasing), representing infinite and non-planar structures and resolving ambiguities, such as those that arise through the use of "negative" arrows.

To give the language formal semantics we rely on the mathematical notions of Venn diagrams and graphs. We intend to factor out that part of the semantics which is common to all domains from the domain-specific parts. For example, in the above picture under a shared memory model of concurrent processing, one can interpret the boxes on the left as sets of processes, the boxes on the right as shared objects, and the arrows as operation invocations. I am developing semantics for the language, initially in its application to the security domain. These semantics are sufficient to recognize the cases of ambiguity that can arise in a well-formed picture. An interesting open semantic issue is how to define the semantics of our visual language in a visual way, rather than denotationally as we now do.

3 Future Work

I intend to develop futher the formal semantics for Miró, to influence the language design, and to provide specific tools for language users. I will present a summary of semantic results at the IEEE Visual Language Workshop this October, and have been asked to submit these results for publication in the IEEE "plenum" volume on visual languages. The language has already evolved beyond the current level of the formal semantics, as it now includes a constraint language for pictures. I will develop rigorous semantics for these constraints, and intend to apply to language to domains outside of security.

Miró Visual Specification Language

- Named after Joan Miró (1893 - 1983), Spanish artist

Miró Project Goals:

- Visual Language Design
- Formal Language Semantics
- Software Tools (editor, semantics checker)
- Applications: Security, Concurrency

Security Application:

- Language consists of boxes and arrows (Venn diagrams, relations between users and files)
- Access Matrix is the underlying model

Proposed Software Tools

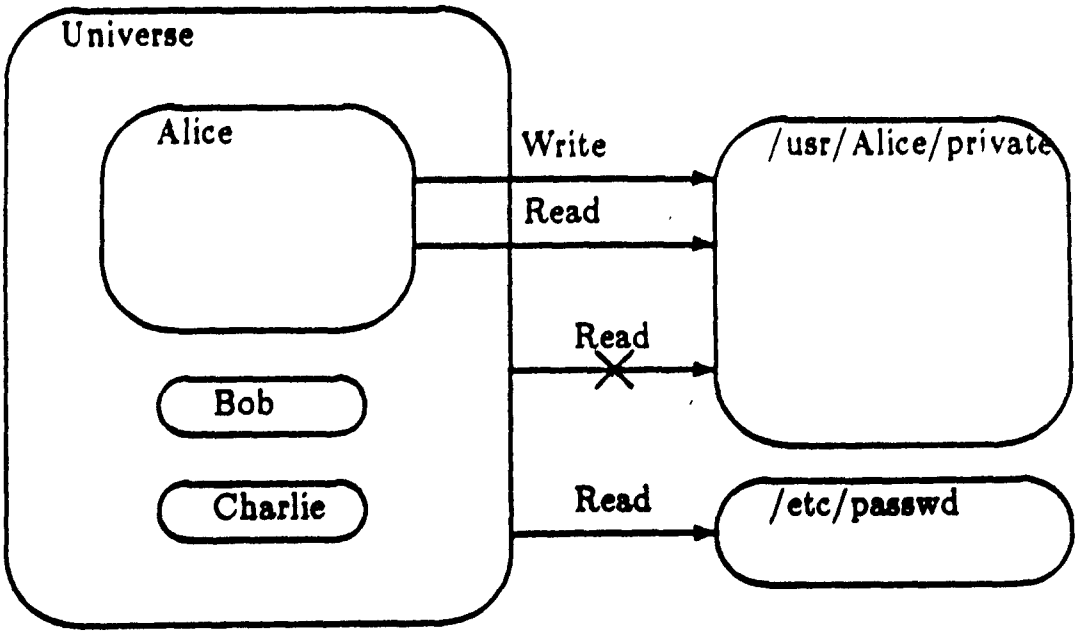
- Editor - create/modify Miró specifications; will be based on the GARNET project (device-independent graphical application package)
- Syntax Checker - verify well-formedness constraints
- Picture Inference - build a Miró picture from an existing file system
- Enforcer - make the current file system conform to the Miró specification

The editor and a Postscript tool are being developed now.

Personal Achievements:

- Formal semantics for a subset of the language
- Presentation of semantic results at IEEE Workshop
- Prelimiary work on editor tool

A Simple Miró Picture ...



... and Its Interpretation

	/etc/passwd			/usr/alice/private		
	read	write	execute	read	write	execute
alice	pos	neg	neg	pos	pos	neg
bob	pos	neg	neg	neg	neg	neg
charlie	pos	neg	neg	neg	neg	neg

Stewart M. Clamen
clamen@cs.cmu.edu
Contract # N000M-88-K-0641
Dr. André Van Tilborg (Supervisor)

FY88 Summary Report

Project Goals

A distributed system consists of multiple computers (called sites) that communicate through a network. Distributed systems are typically subject to site crashes and communication link failures. A crash renders a site's data temporarily or permanently inaccessible, while a communication link failure causes messages to be lost. A failure is detected when a site that has sent a message fails to receive a response after a certain duration. The absence of a response may indicate that the original message was lost, that the reply was lost, that the recipient has crashed, or simply that the recipient is slow to respond.

The goal of Avalon is to create a set of linguistic constructs designed to give programmers explicit control over transaction-based processing of atomic objects for fault-tolerant applications. These constructs are being implemented as as extensions to C++, Common Lisp, and Ada^{TM}. The constructs include new encapsulation and abstraction mechanisms, as well as support for concurrency and recovery. The decision to extend an existing language rather than to invent a new one was based on pragmatic considerations. We felt we could focus more effectively on the new and interesting issues of reliability and concurrency if we did not have to redesign or reimplement basic language features, and we felt that building on top of a widely-used and widely-available language would facilitate the use of Avalon outside our own research group.

We are currently implementing Avalon/C++, superset of C++, augmented by primitives that support distribution, concurrency, reliability and fault-tolerance. C++ was chosen over C because its object-oriented style of programming lends itself well to a distributed programming approach. The Avalon run-time system relies on two systems currently under development at CMU. Mach, a Unix-like operating system with support for distributed computation, is used to provide communication among the various Avalon processes, and to support process-level concurrency. Camelot, a machine-independent, high-performance, distributed transaction facility, is used to support the fault-tolerance and reliability we desire.

Past Accomplishments

The preliminary Avalon/C++ language design is complete, and we have nearly completed our work on a preprocessor to translate Avalon/C++ code to C++ code. The parser and type checkers are working, and the compiler's semantic phase, which transforms Avalon constructs into calls to the Camelot distributed transaction primitives and the Mach communication primitives, is nearly complete. The only remaining major portion of the system that has yet to be integrated is the facility for starting servers from within user programs; a facility which will be incorporated in the very near future.

My own work has focused on the Avalon/C++ communication mechanisms. Communication among the various processes that make up an Avalon/C++ application is achieved via a Remote Procedure Call (RPC) mechanism. In this way, the actual (network message) transmission of values between the two communicating processes (client and server) is masked by a traditional programming construct. For an RPC to operate correctly, the system must be able to translate the abstract representation of the arguments and return values into and out of some transmissable representation. In an effort to further conceal this message transmission from the programmer, Avalon/C++ automatically generates these translation routines. A preliminary version of the communication support is mostly done, its completion hinging on the the repair of a deficiency in the underlying Mach system.

Future Goals

This fall will see the integration of the support for remote server initialization into our existing Avalon/C++ compiler. Much energy will be spent removing any remaining bugs in the system, and there will be efforts to reduce the size of the resulting application programs.

Over the next year, I also plan to investigate Avalon/Lisp, an extension of Common Lisp to support reliable distributed programs. One possible application of Avalon/Lisp is the management of large, long-lived knowledge bases. The Avalon primitives could be used for replication, to ensure that the data remains highly available, and for reliability, ensuring that site crashes do not cause the data to become lost or inconsistent. The Lisp model permits several novel approaches to managing large objects, including sending procedures to data rather than the other way around. Because of its interactive nature, Avalon/Lisp promises to be an effective testbed for rapid prototyping of reliable distributed systems.

The major issue in this investigation will be to determine an appropriate model of concurrency. Concurrency in Lisp has been modeled in a number of ways over the past several years. Futures (as implemented in Halstead's Multilisp) and Qlisp (as developed

by McCarthy and Gabriel) are but two examples. Some experimentation will be necessary to determine which model would lend itself best to a distributed environment. Another important issue is the form exceptions and exception handling should take in a Lisp environment, an issue that has not been the object of much research. Exception handling is a particularly important issue in a distributed system where programs must be designed to tolerate failures.

Stewart Clamen

clamen@cs.cmu.edu

Avalon Project Design Goals:

- What are the *right* primitives and language extensions?
- Are the right primitives fast enough?
- Do the language extensions match the existing programming paradigm?
- What is the program supposed to do? Can one prove it?
- What neat algorithms can one implement in the new language(s)?

Past Achievements:

Implementation of Avalon/C++ as a preprocessor for C++. Personal projects as part of implementation include:

- Automatic generation of code for transmission of arguments to remote procedures (RPCs).
- Support for tracing of server operations. (Debugging Support)
- Development of Avalon/C++ example: Atomic Counter (supporting concurrent *increment*, *decrement*, and *test-for-zero* operations).

Short- to Medium-Term Objectives:

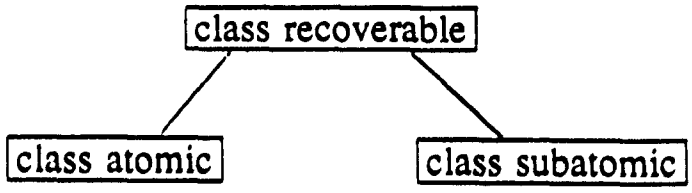
Debug Avalon/C++ implementation.

Design of reliable, distributed computing environment for CommonLisp (Avalon/Lisp) .

Approach:

1. Examine possible semantics for Avalon/Lisp:
   - Explicit (user-controlled) vs. implicit (automatic) transaction processing,
   - Explicit vs. implicit recoverability of program data and code,
   - Methods for exception handling.
2. Model various approaches to assist in examining differences. Critical differences include: ease of programming, similarity to programming models, etc.

Avalon/C++ Type Hierarchy



Class Hierarchy Detailed

```
class recoverable {
 public:
  virtual void pin();
  virtual void unpin();
};
```

```
class atomic: public recoverable {
 public:
  virtual void write_lock();
  virtual void read_lock();
};
```

```
class subatomic: public recoverable {
 protected:
  void seize();
  void release();
  void pause();
 public:
  virtual void commit(trans_id& t);
  virtual void abort(trans_id& t);
};
```