Infrastructure for the Design and Rapid Deployment of Telecommunications Applications

Wan-Teh Chang

Prof. David G. Messerschmitt

Prof. Edward A. Lee

Department of Electrical Engineering
and Computer Sciences
University of California at Berkeley

Overview

Objective: proliferation of telecom applications

- **Design** telecom applications faster
- **Introduce/deploy** telecom applications faster

Design: telecom applications are

- Concurrent programs
- Control-intensive
- **Design environment:** specialized model for control, and mixing models of computation

Deployment:

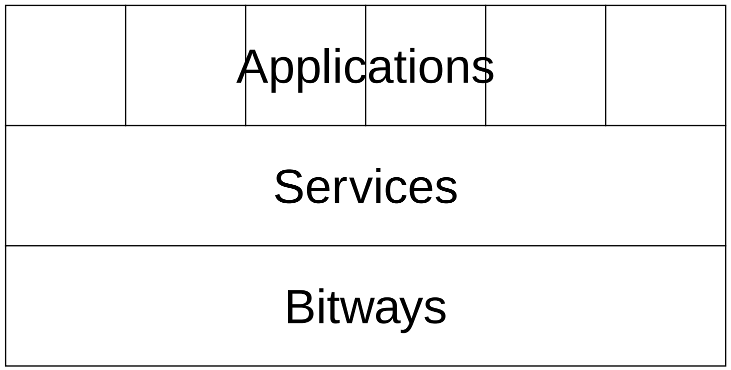
- Architectural constraints
- Economic barrier
- **Dynamic deployment:** architecture, session establishment protocol

Terminology

Networked applications:

- **Client-server:** video on demand, WWW browsing, and file transfer
- **Peer-to-peer:** telephony, video conferencing, electronic mail, and voice mail

Peer-to-peer networked applications include general telecom applications (often called telecom services) and collaborative applications.



Design Environment: Motivation

Telecom applications are

- **Heterogeneous in design styles: signal processing, control**
- **Intricate distributed control**

Objective: a design environment for telecom applications that supports

- **Better abstractions for specifying control**
- **Mixing signal processing and control**

Models of Computation

A system is organized into modules or components.

Modules are written in a high-level programming language (e.g. C++): host language

Modules interact with each other according to a model of computation (MoC): coordination language

MoCs are domain-specific, intuitive.

Examples of MoCs:

- Dataflow
- Discrete-event
- Synchronous reactive (Esterel, Lustre, Argos)
- Finite-state machines

Heterogeneous Approach

Heterogeneous approach: **combines small, specialized models of computation**

- **Achieves generality**
- **Automatic synthesis and formal verification**

Mixing concurrency models

Mixing hierarchical finite-state machines with concurrency

Ptolemy and Tcl/Tk

Ptolemy is a simulation and rapid prototyping environment.

- **Software modules in Ptolemy can be parameterized and interconnected to form systems**
- **Models of computation can be mixed**
- **Simulation and code generation (C, DSP assembly) capabilities**

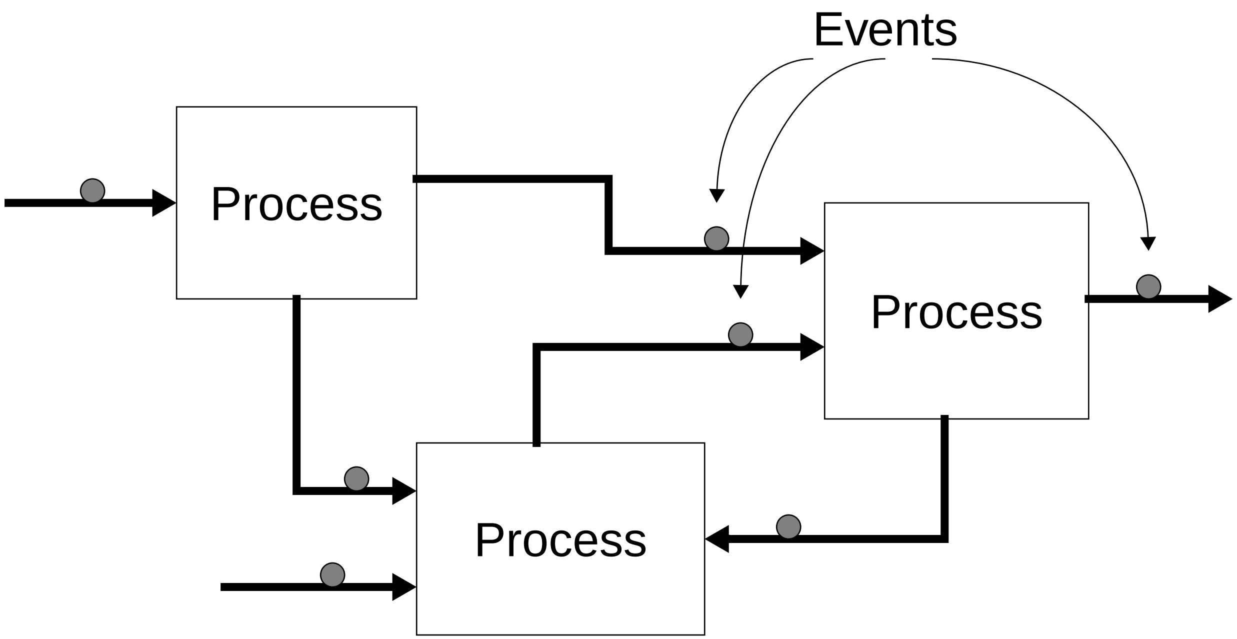
Tcl is an embeddable interpreted command language.

Tk is a windowing toolkit based on Tcl.

Concurrency Models

Concurrency: processes (modules)

Communication: events

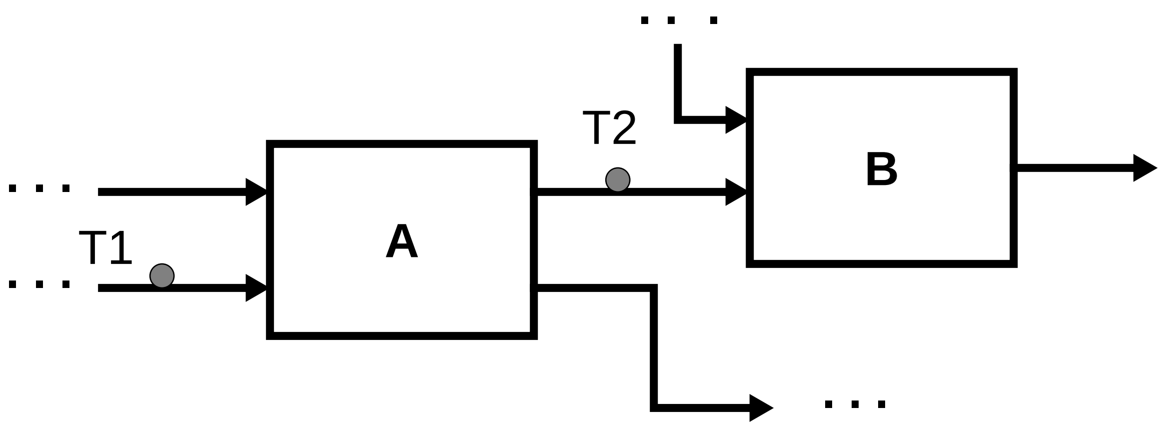


Discrete-Event

Discrete-event (DE): events have time stamps.

- **Events are totally ordered by time stamp.**

The DE simulator sorts the events by time stamp and process the events in chronological order.

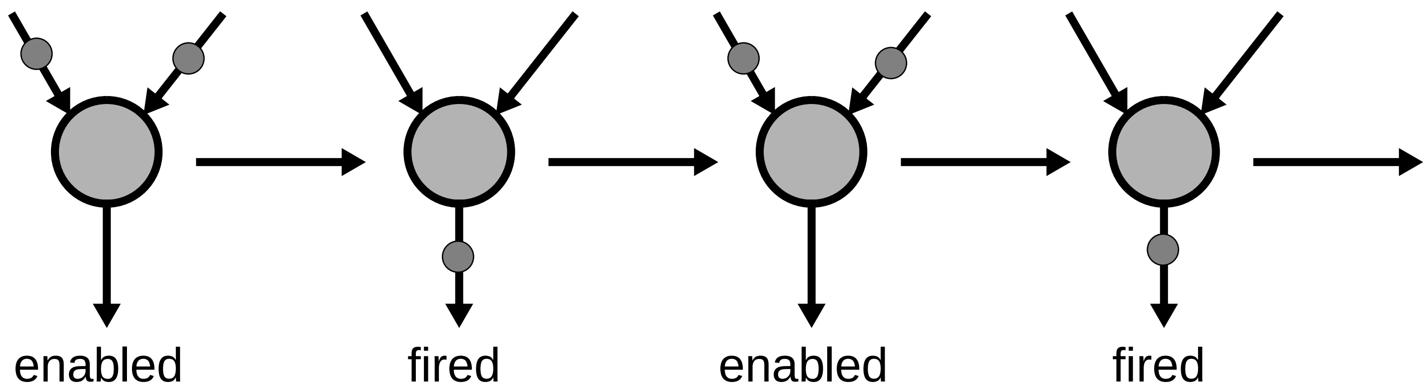


Block A fires at time T1, generating an event with time stamp T2 (T2 >= T1). Block B fires at time T2.

Synchronous Dataflow

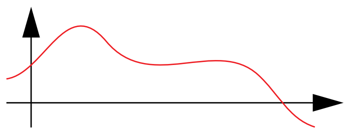
Synchronous dataflow (SDF): a block consumes a fixed number of tokens and produces a fixed number of tokens in each firing.

- **Good for modeling multirate digital signal processing.**
- **Block firings are partially ordered, sequenced only by data dependency.**

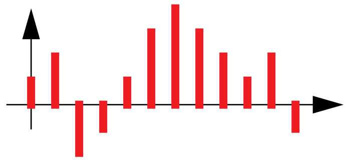


Repeated firings of a dataflow block

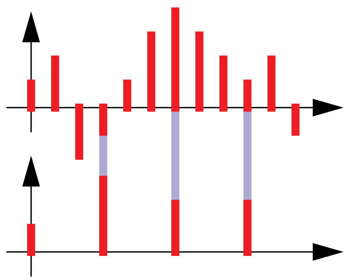
Modeling Discrete-Event Systems



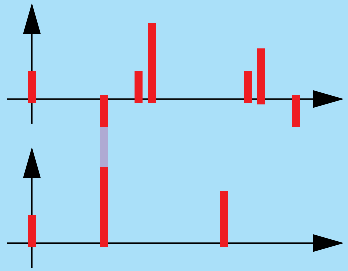
continuous time



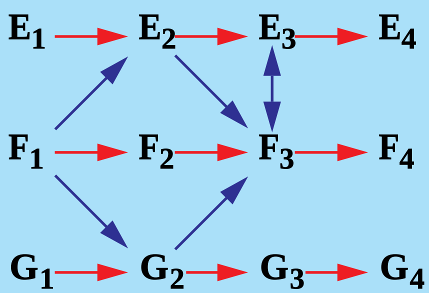
discrete time



multirate discrete time



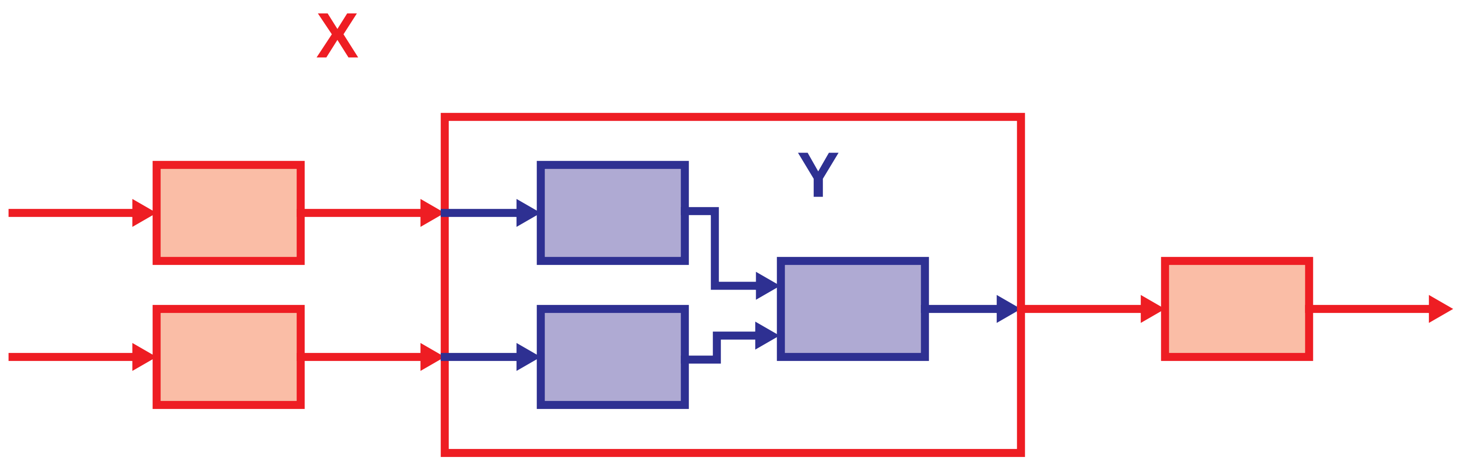
totally-ordered discrete events



partially-ordered discrete events

Mixing MoCs by Hierarchical Nesting

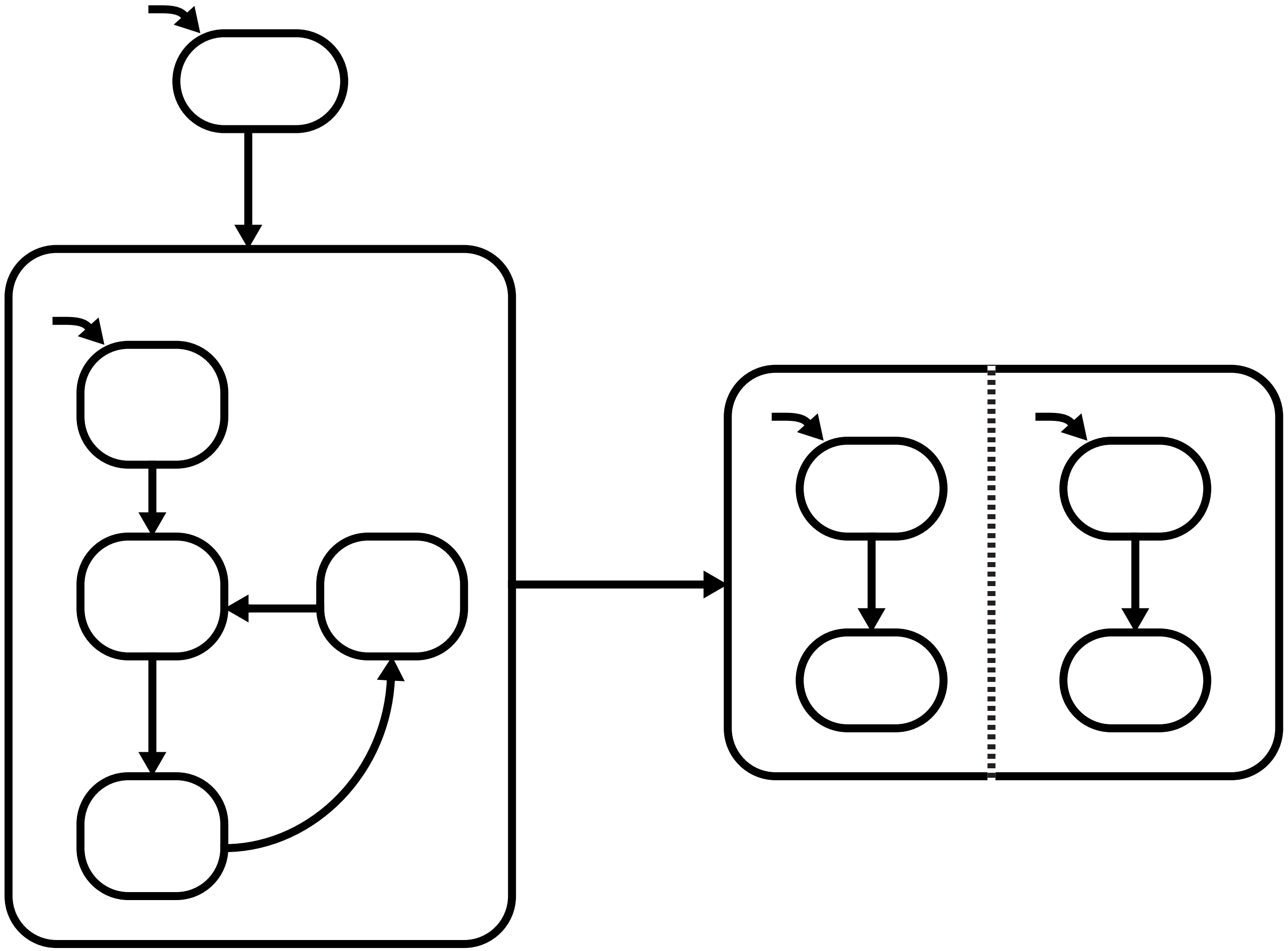
- MoCs are mixed hierarchically.
- Blocks in the MoCs have discrete firings.
- Key constraint: each MoC have a well-defined quantum of computation.
- Execution proceeds as a sequence of quanta of computation with time stamps



A subsystem of MoC Y embedded in MoC X as a hierarchical block

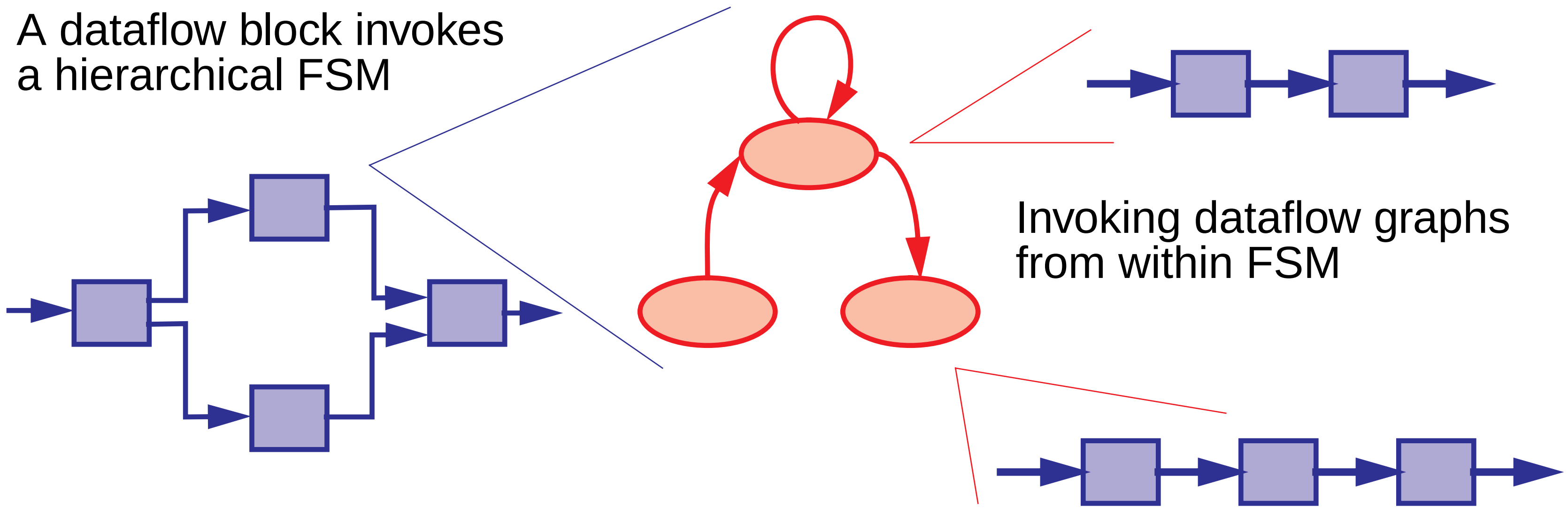
Abstractions for Control

Statechart: finite-state machine + hierarchy + concurrency



Mixing Hierarchical FSM with Concurrency

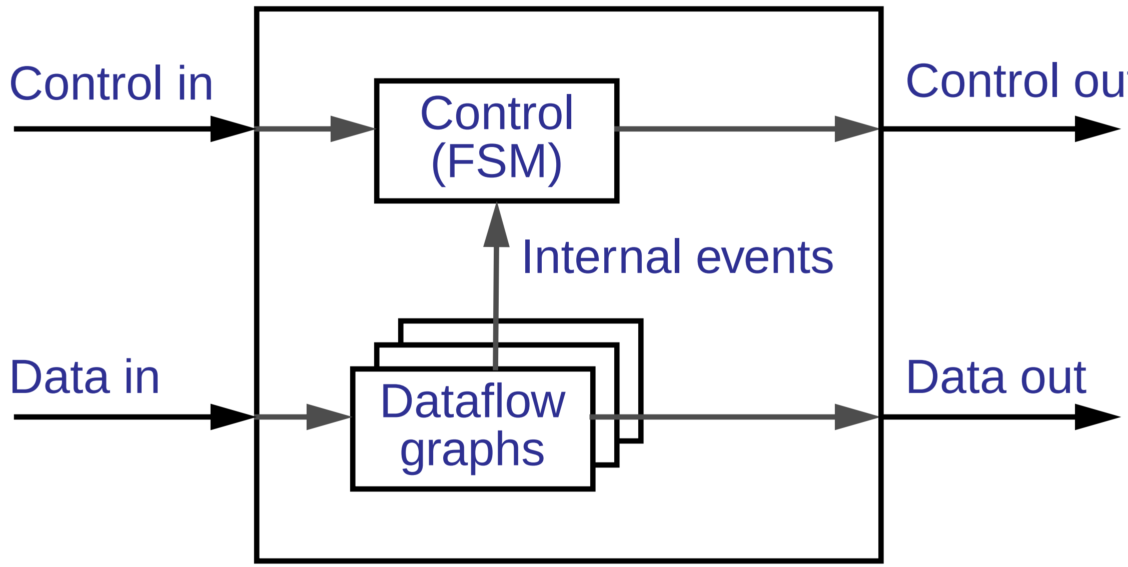
- **Sequential behavior (finite-state machine), hierarchy, and concurrency are orthogonal semantic components.**
- **Hierarchical finite-state machines (FSMs) can be nested with different concurrency models (SDF, synchronous reactive) to get (essentially) variants of Statecharts.**



FSM Controls Invocation of Dataflow Graphs

A block is replaced by one of a set of dataflow graphs.

The choice of dataflow graph is controlled dynamically by a FSM.



Summary of Hierarchical FSM

- **Defined semantics of hierarchical FSM**
- **When combined with concurrency, well suited for specifying complex control functionality**
- **Graphical editor for state transition diagrams**
- **Simulation and code generation (C, Tcl)**
- **FSM controls signal processing tasks specified as dataflow graphs**

Rapid Deployment: Motivation

Two obstacles to rapid deployment of new networked applications:

- **Architectural constraints: network-based applications**
- **Standardization at application level**

Major economic barrier to deployment of peer-to-peer applications:

- **Network externality problem: early users derive little benefit from the applications**

Solution

Applications defined in user terminals, and increasingly in software.

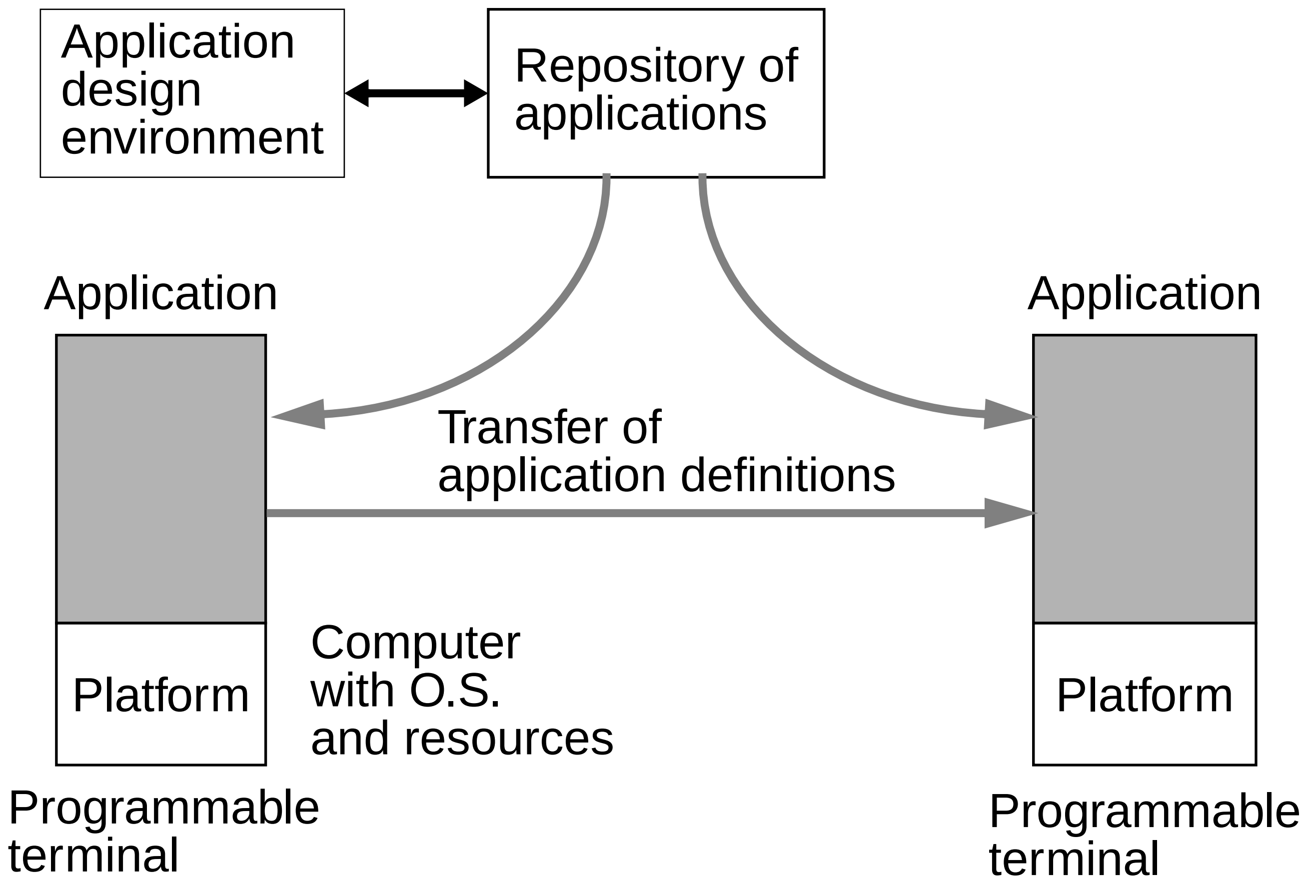
Network deployment: software-defined applications can be distributed via the network.

- **Web browsers, document viewers, audio players, etc. on the Internet use this distribution mechanism.**
- **Users still have to anticipate the need and install the application software.**

Dynamic deployment: transfer application definition at session establishment (and during the session).

The Dynamic Deployment Approach

- **Platform**
- **Application definition language**
- **Protocol for transfer of application definitions**



Dynamic Deployment: Discussion

- Limit standardization to infrastructure elements
- Downloadable software definition of remainder of application functionality
- Functionality similar to LAN/fileserver, but new problems introduced
- Bypass network externality problem: a community of interest consisting of all networked platforms
- Requires software definition of application and high-speed network

Dynamic Deployment: Issues

Security: executing application definitions from external sources

- Application definition language must be a **high-level language** with restricted functionality
- Authentication of trusted sources

Hardware/O.S. independence ==> **high-level language**

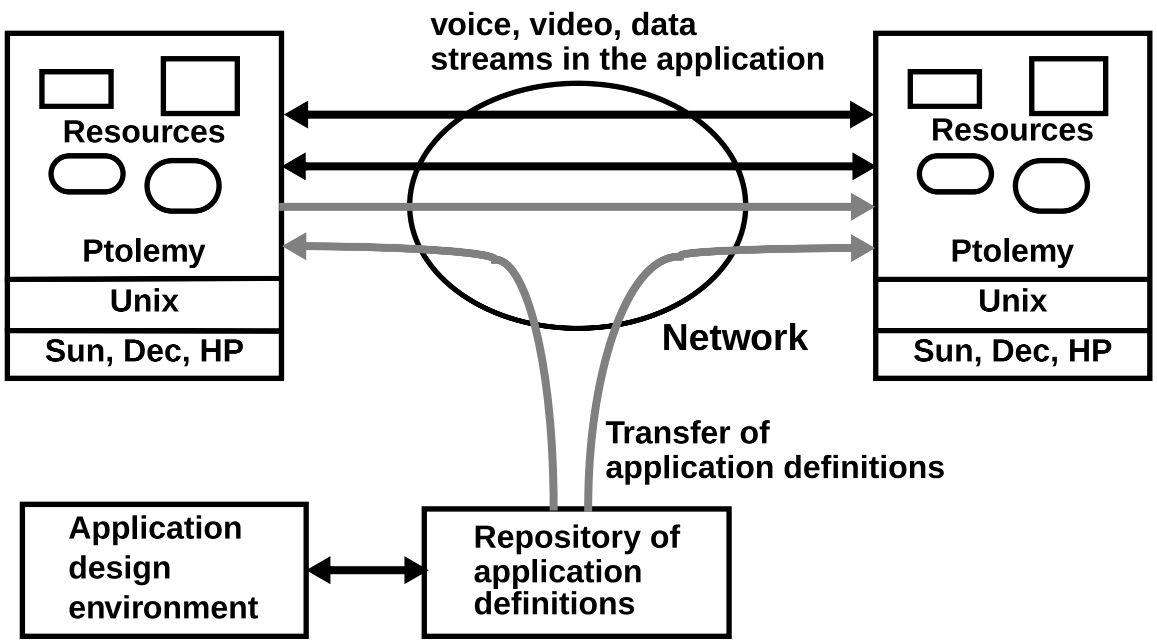
Performance:

- **Session establishment time:** application program size, network bandwidth, interpretation/compilation
- **Run-time:** interpretation overhead

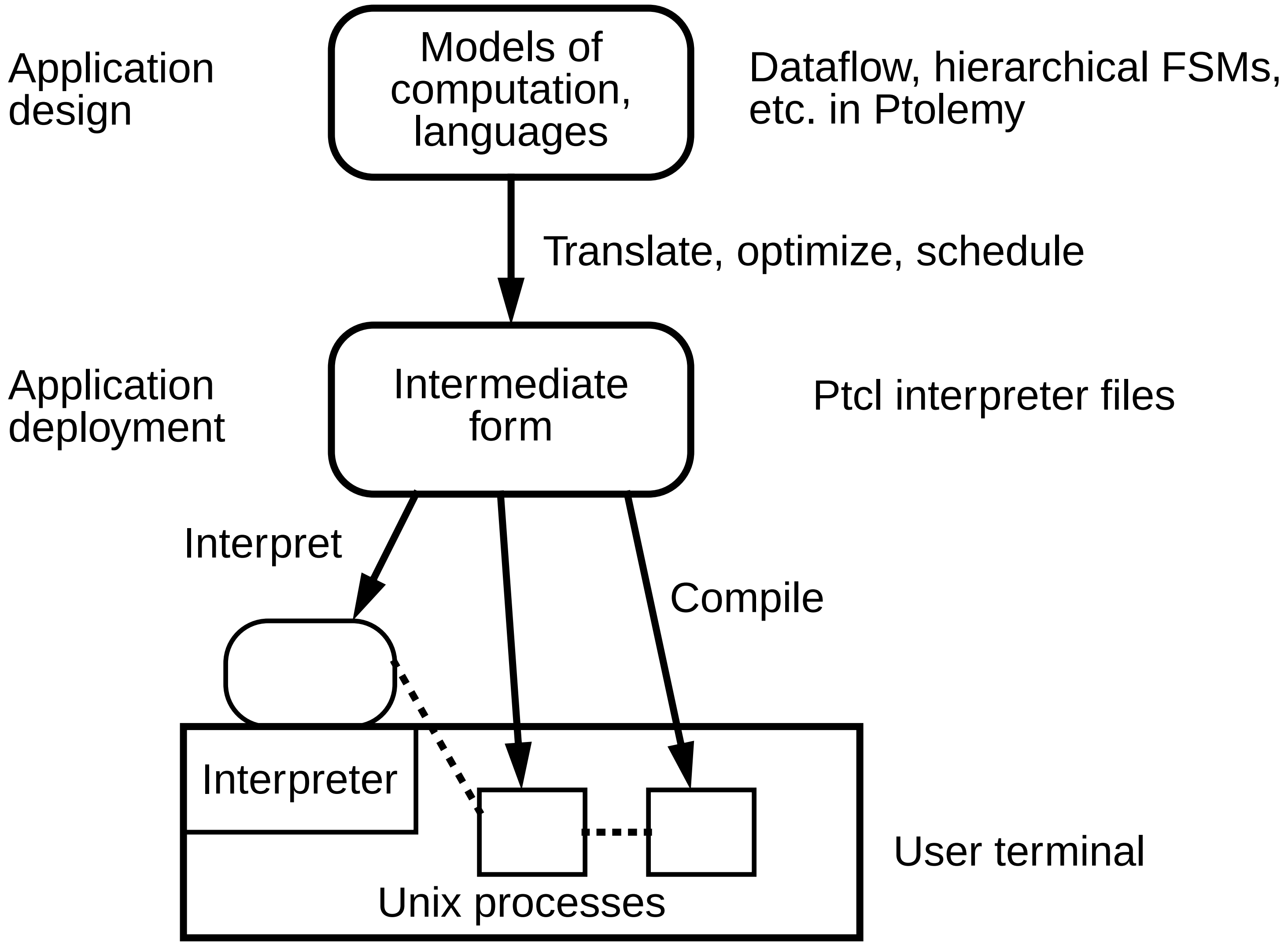
Pricing and charging; licensing

Prototype of Dynamic Deployment Approach

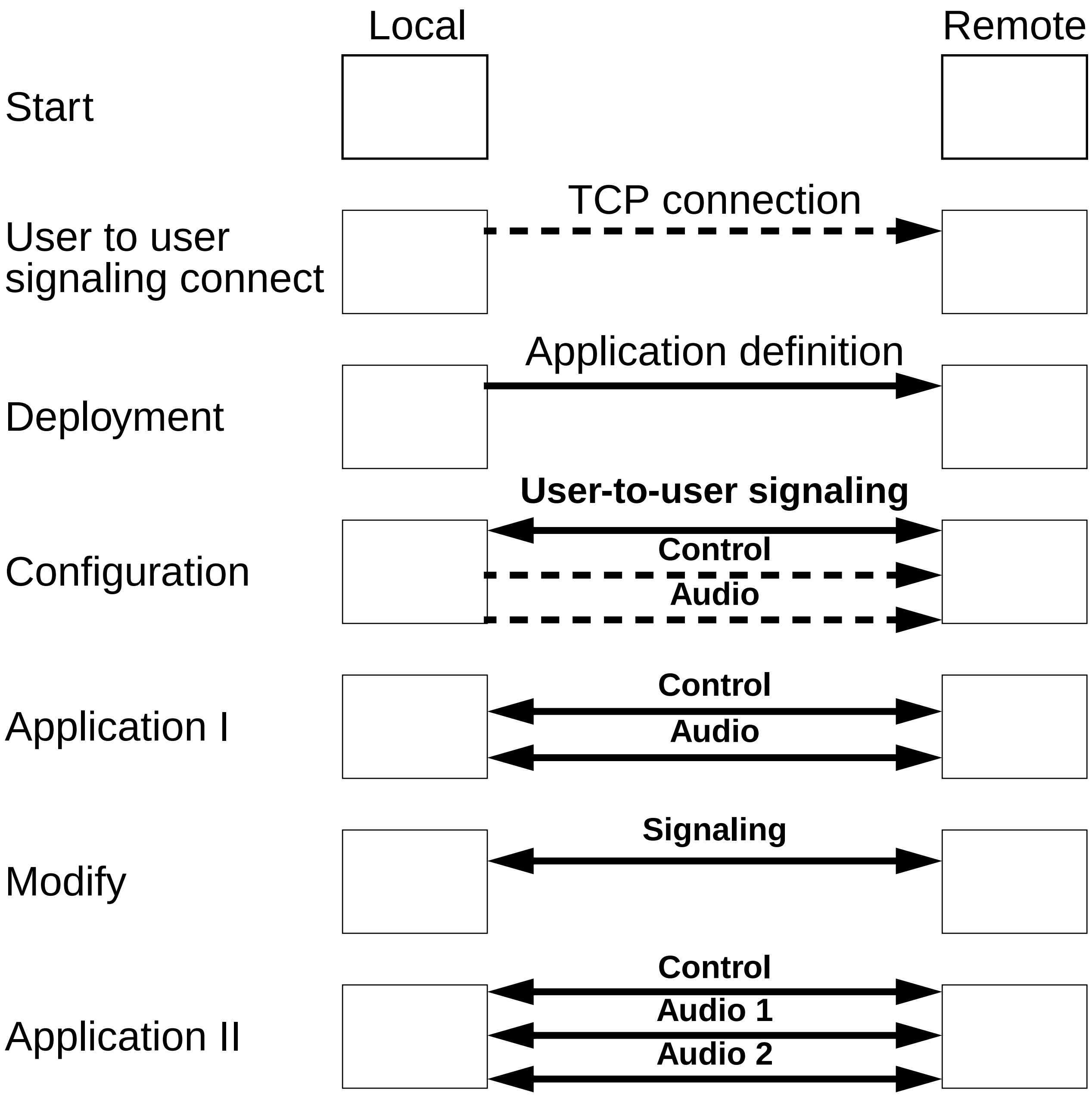
- **Platform: Ptolemy running on Unix workstation**
- **Application definition language: Ptolemy interpreter language Ptcl**
- **Protocol for transfer: Tcl-DP**



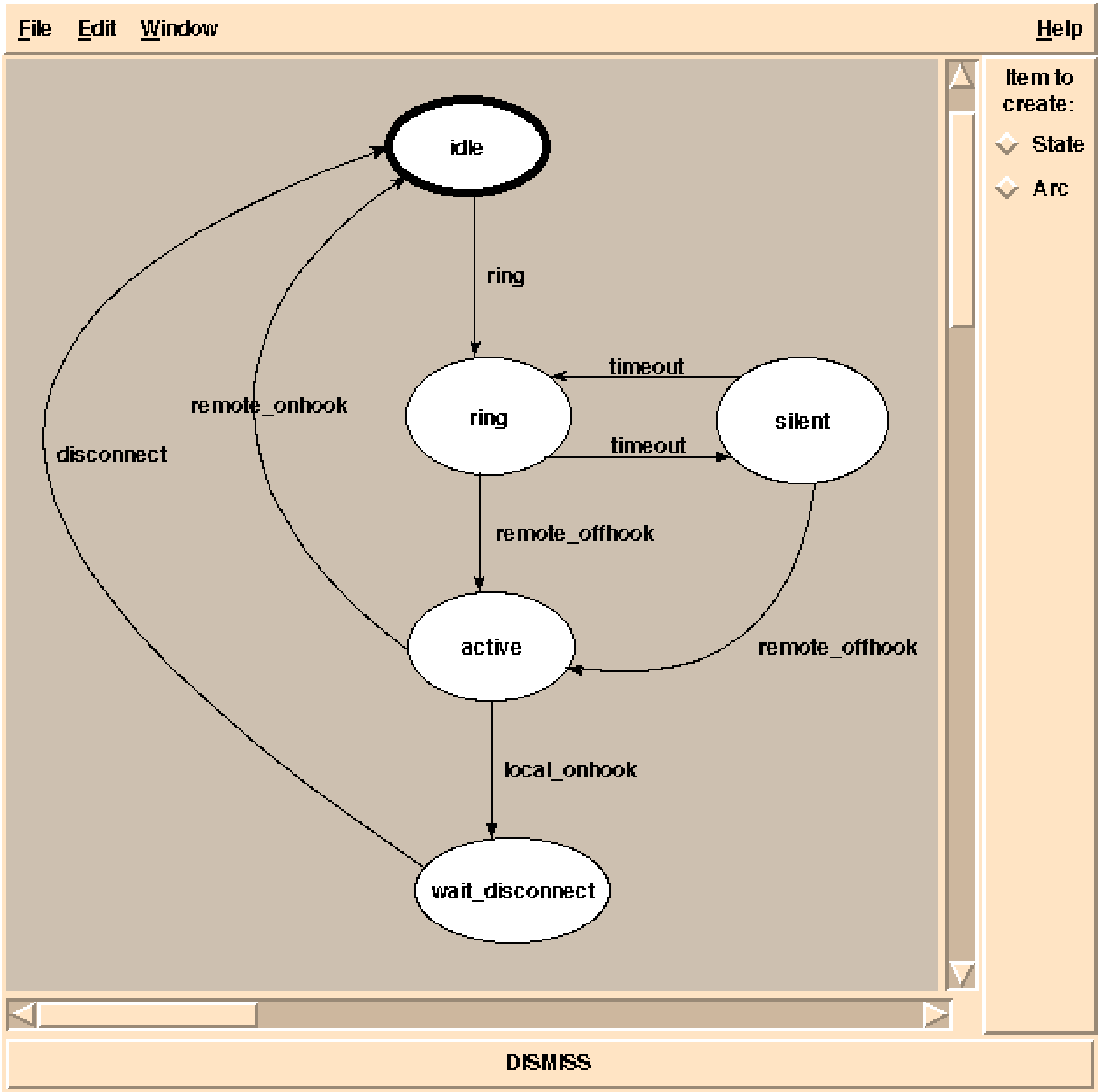
Application Design and Deployment Flow



Steps in Dynamic Deployment

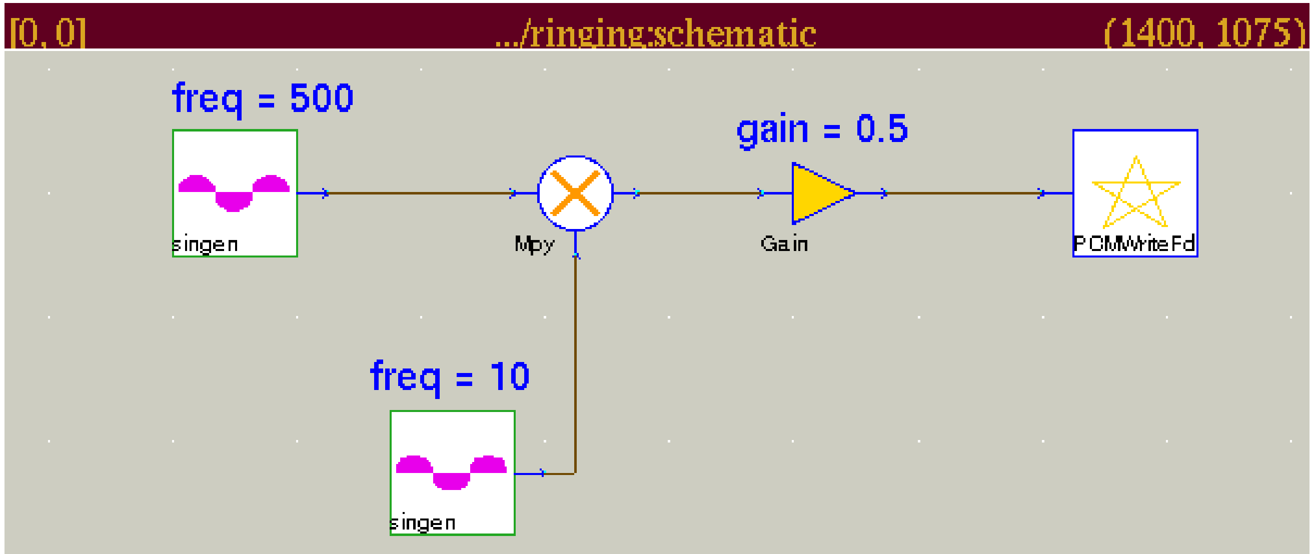


Software Emulation of Telephone

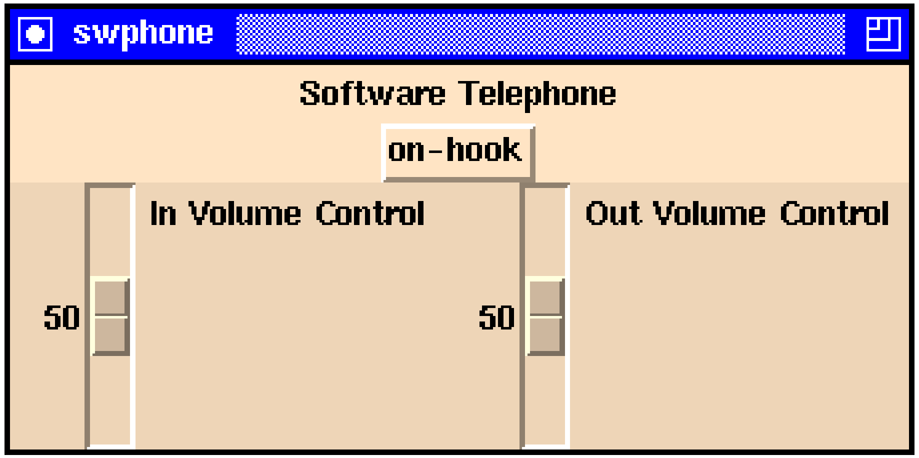


Software Emulation of Telephone (cont'd)

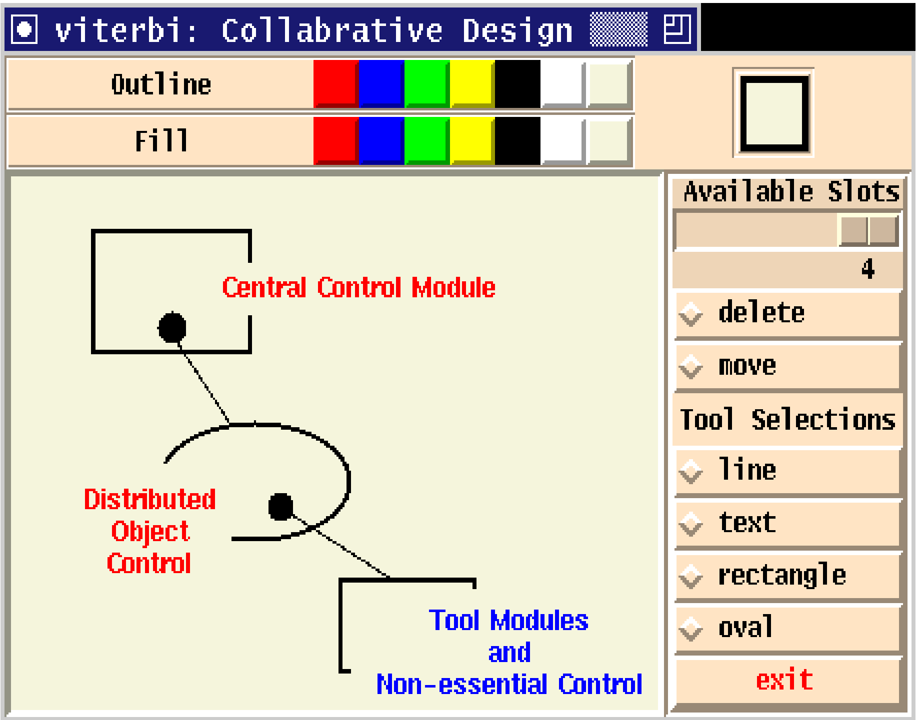
Ptolemy system for generating the ringing tone



GUI



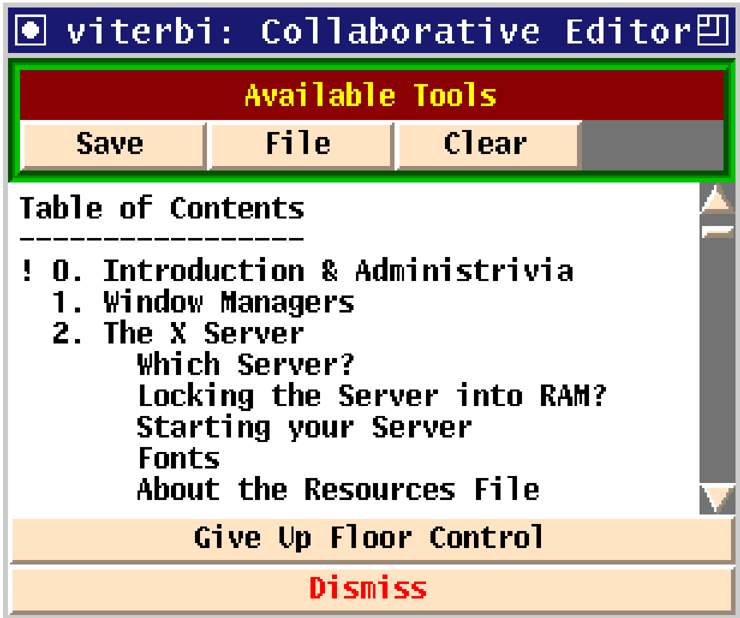
Collaborative Design Applications



Shared whiteboard

– Dynamic deployment of features

Collaborative editor



Related Work

- **Safe-Tcl: Enabled Mail (Safe-Tcl scripts embedded in email messages)**
- **The Telescript language of General Magic: agents (mobile programs)**
- **The Java language of Sun Microsystems: Java applets in Web pages**

Conclusions

Heterogeneous approach

- Impose small, specialized models of computation on a programming language
- Combine models of computation
- We can mix dataflow, discrete-event, synchronous reactive, and hierarchical FSM models.

Dynamic deployment approach

- Avoid standardization of actual application
- Limit network externality problems
- We have a real working prototype we can demo