CONCERT
A Software Architecture for Building Distributed Education Sessions

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040 Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

Background

Distributed Education

Conventional view: *video conferencing*.

- Audience sited in various remote locations.
- Presentation is recorded and multicast.
  - Audio and video.
  - Special lines (ISDN, ATM) or Internet (MBone).
- Transmitted information:
  - Lecturer's speech and gestures.
  - Images of blackboard drawings/slides.
- Backchannels
  - Feedback from audience.
  - Group discussions.

What you see or hear is (all) what you get.

Distributed Computer-Supported Education

Surface view: *shared screen*.

- Lecturer's user interface is monitored.
- Screen drawing operations are multicast.
- All participants receive identical screen copies.
- Input control may be switched among participants.
- *Shared X*, ...

Screen images as a supplement to video.

Distributed Computer-Supported Education

Beyond the surface: *shared tools*.

- Based on *logical* structure of computer-supported sessions.
- *Concert* of multiple software *instruments*.
  - Slide presentations.
  - Domain specific application software.
  - Audio, video.
- Lecturer *conducts* use and interaction.

Focus on the <u>underlying objects</u>.

Coordinating Distributed Education Sessions

Very challenging task!

- Coordinatin of concurrent activities.
- Real-time interaction.
- Multi-media objects.
- Currently: almost unfeasible.
  - Computer experts.
  - Parallel programming.
  - Network programming.

Goal: break down complexity.

Requirements

Benefits of Computer Support

Two basic advantages:

1. Improve didactical value.
   (a) Computer support for application domain.
   (b) e.g. Mathematica© for mathematics.
2. Participation of non-local users.
   - Wide-area networks (Internet or special lines).
   - Collect and utilize remote teaching experience.
   - Widen target audience of presentations.
   - Overcome participation obstacles (physically handicapped).

How to organize sessions?

Session Development

- Prepare basic materials:
  - Texts, images, animations, programs, hypermedia documents, ...
  - Like in any computer-supported course.
- Coordinate their use:
  - Invoke browsers, viewers, interpreters. ....
  - Manage causal and temporal relationships.
  - Handle inputs and outputs.

Support coordination tasks as much as possible!

Session Operation

- Start system on lecturer's host.
  - Possibly equipped with audio/video devices.
- Additional hosts request session access.
  - Define their *role* in the session.
  - Simple sessions: single lecturer plus multiple audience hosts.
  - Complex sessions: supervisor, lecturers, floor managers, active participants, passive participants.
- Load and execute part of session program corresponding to role.
  - Load media objects.
  - Invoke applications, invoke applications, synchronize interaction.

Simplify routine management tasks as much as possible!

Session Components

- Audio and video.
  - Audio carries most of the information!
- Electronic whiteboards.
  - Handwritings and drawings.
- Slide shows.
  - Sets of text documents augmented by graphics/animations.
  - Active multimedia documents.
- General applications.
  - Mathematica notebook frontend and execution kernel.
  - Present mathematical documents and execute mathematical programs.
- Feedback from audience!
  - “Raise hand” and ask questions.

Integrate standard applications as much as possible!

Summary

Need support for:

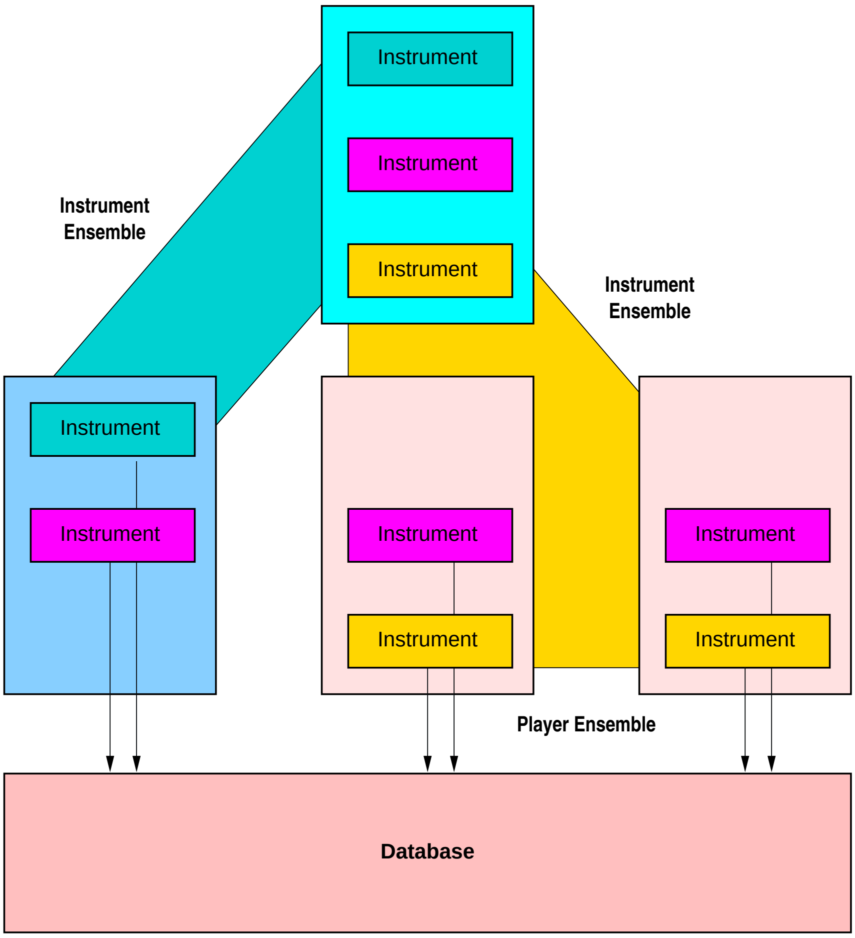
- Developing sessions.
  - End-user programming!
  - Developing, testing, debugging.
- Performing sessions.
  - Relief from standard management tasks.
  - Control of session access.
  - Focus on *single lecturer* scenarios.
- Integrating components.
  - Open interfaces.
  - Flexible usage.

The CONCERT Architecture

Overview

- Software architecture.
  - Mental model providing organizational framework for designing sessions.
  - Application programming interface for concrete session programs.
- Middleware for building education sessions.
  - Above low-level network and communication system.
  - Below education session.
- Aimed at distributed *education sessions.*
  - No general distributed application framework!

Mental Model



Basic Concepts

- Concert
  - Instance of an education session in action distributed among multiple *hosts* and involving multiple *players*.
- Score
  - Concert program according to which players operate.
- Instrument
  - Application component that executes independently as an asynchronous process.
- Player Ensemble
  - Set of *players* that handle same set of instruments and are treated alike.
  - Sets of *instruments* of the same kind.
- Database
  - Global network database contains all objects required by instruments (documents, programs, ...).

Instrument Ensembles

Concert consists of performance of multiple instruments ensembles.

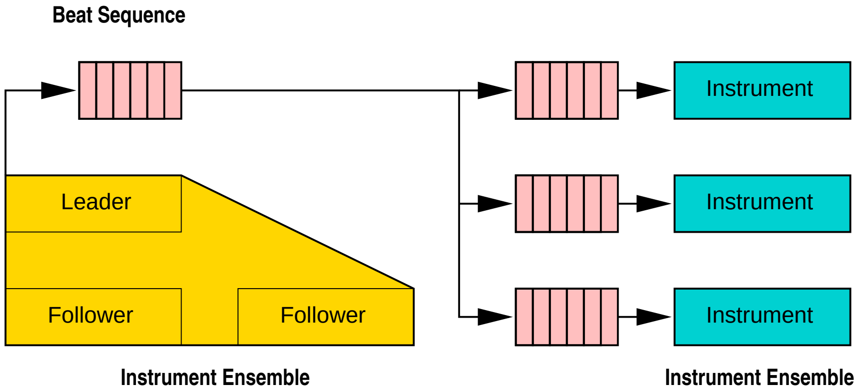
- *Leader* mode:
  - One member of the instrument ensemble (*leader*) controls operation of all other members (*followers*).
  - Presentation of slides, program runs, ...
- *Independent* mode:
  - Activities of all members of an instrument ensemble proceed independently.
  - Exercising on an application program. independently
- Each instrument runs as a parallel process.
  - Start and termination triggered by score.
  - Synchronization by beat sequences.
- Mode switches.
  - Score switches ensemble from leader mode to independent mode.
  - Score designates new ensemble leader.

Beat Sequences

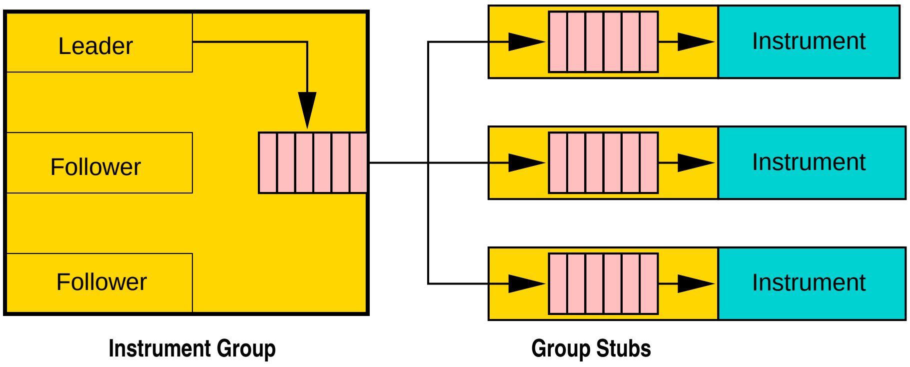
Synchronization of different *instrument ensembles*:

- Sequence of (named) signals associated to ensemble.
- Generated by leader instrument.
- Instrument may wait for
  - next beat from some particular ensemble, or for
  - beat with a specific name.
- Each instrument receives individual *copy* of beat sequence:
  - Ensemble mates are not punished by slow connection to some instrument.
  - Late comers are not punished and receive identical copy.

Beat Sequence (Abstract View)



Beat Sequence (Concrete View)



Communication

Basic kinds of communication:

- Score to instrument ensemble:
  - Passing (same) arguments on activation.
- Leading instrument to follower:
  - Followers replicate activities of leader.
  - Based on *instrument-specific* protocols.
- Between *different* instrument ensembles:
  - Synchronization by *beat sequences*.
  - Communication by objects in *network database*.

Runtime System

Basic components:

- Concert score
  - Program downloaded and interpreted by each concert player.
  - Java classes generated from score.
  - Compiled to Java VM byte code.
- *Concert manager*
  - Runs on some host, lists for participation requests, and transmits corresponding scores.
- *Concert clients*
  - Runs on players' hosts connect to concert manager and retrieve scores.

Score Classes

Java classes generated from high-level concert score.

- Instrument classes:
  - `monitor`: lecturers' monitor.
  - `slides`: slide presentation.
  - ...
- Control classes:
  - Concert session.
  - Access control.
  - Player ensembles.
- Object generation and linkage:
  - Link player ensembles to session and access control.
  - Link instrument ensembles to player ensembles.

Concert Instruments

```
class Instrument
{
  ...
  start();                // start process
  stop();                 // terminate process

  async();                // independent mode
  leader(Instrument);     // back to leader mode

  String wait();          // wait for next beat
  wait(String)            // wait for named beat
  ...
}

class SlideViewer extends Instrument
{
  ...
  hide();                 // hide presentation
  show();                 // show presentation
  ...
}
```

Control Classes

```
Concert               // session
{
  ...
  run()
  ...
}

class Control        // access control
{
  ...
  askPassword()
  addDomain(Domain)
  ...
}

class Ensemble       // player ensemble
{
  ...
  Ensemble(Concert)
  ...
}
```

Linking Ensembles to Concerts

```
// session object
MyCourse myCourse = new MyCourse("Computer
Algebra I");

// access control objects
Control pwdHost = newControl();
pwdHost.askPassword();
Control localDomain = new Control();
localDomain.addDomain("risc.uni-linz.ac.at");

// player ensembles
Ensemble lecturer = new Ensemble(myCourse,
pwdHost);
Ensemble localAudience = new
Ensemble(mycourse);

// instruments
Monitor monitor = new Monitor("monitor",
lecturer, all);
Audio audio = new Audio("audio", lecturer,
remoteAudience);
Slides slides = new Slides("slides", lecturer,
audience);
```

Composing CONCERT Scores

Executing a CONCERT Score

- Instrument and control classes generated.
- Objects created and linked.
- `Concert` method controls execution.
- Sequence of method invocations on *instrument ensembles*.
  - Activation/termination of ensemble
  - Waiting for (named) beats.
  - Switch of operation mode.
  - Instrument-specific operations
- Only beat synchronization specified.

Single thread of control visible!

A Textual CONCERT Score

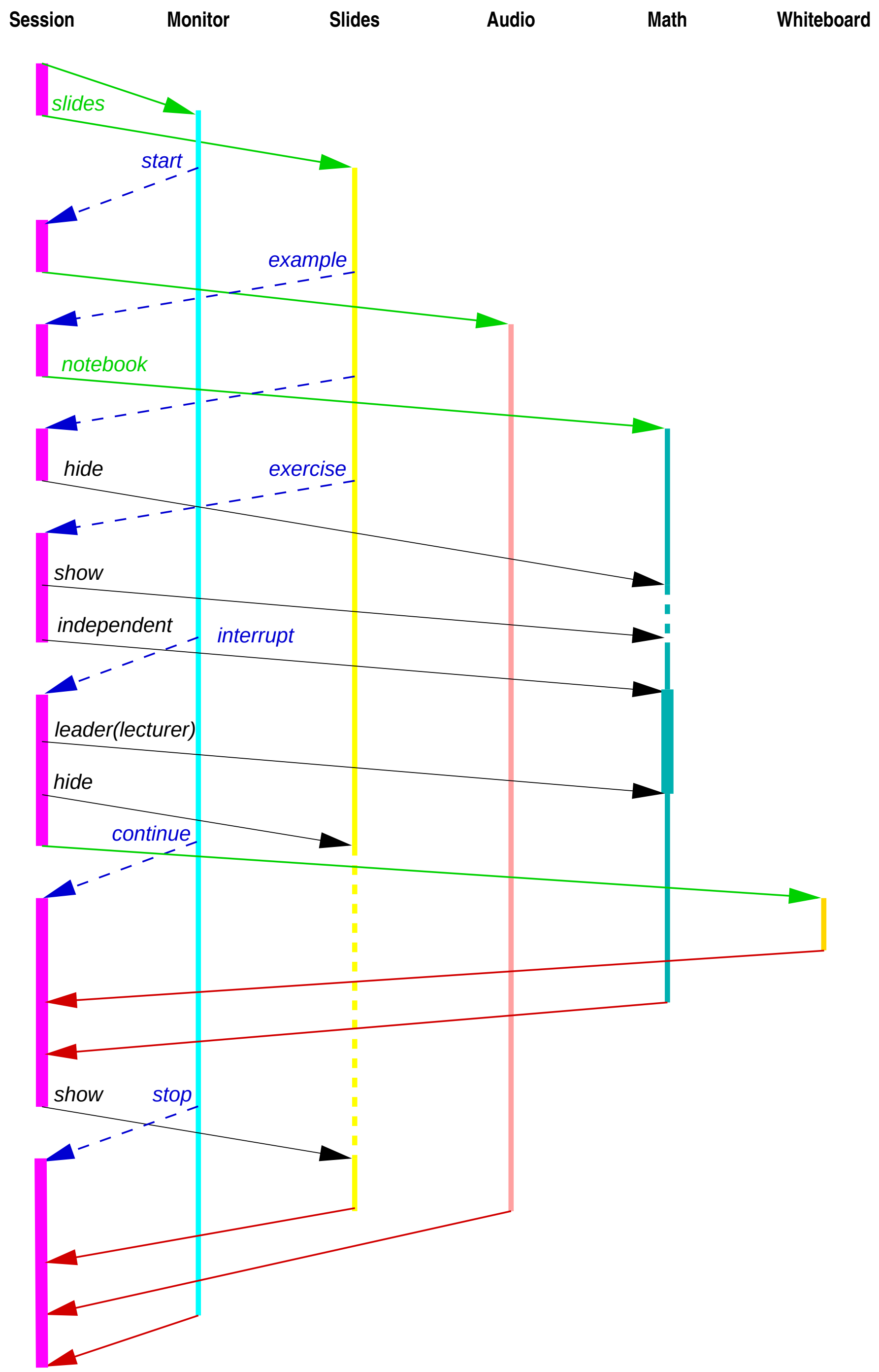
```
run()
{
  monitor.start();
  slides.start("slides");

  monitor.wait("start");
  audio.start();

  slides.wait("example");

  math.start("notebook");
  slides.wait();
  math.hide();

  slides.wait("exercise");

  math.show();
  math.async();
  monitor.wait("interrupt");

  ...
}
```

Developing a Concert Score

- SPMD programming style.
  - No asynchronous distributed programming.
  - Still too low-level for end-user.
- *Visual* representation of concert scores.
  - Instrument ensembles as vertical bars.
  - Green (thick) arrows for ensemble activation.
  - Blue (dashed) arrows for beat synchronization.
  - Black (thin) arrows for general operations.
  - Red arrows for ensemble termination.
- Support by simulator/debugger.
  - *Off-line* development of scores.

First step towards really *composing* scores!

A Visual CONCERT Score



Building CONCERT Instruments

General Issues

1. Interaction between leading instrument and its followers,
2. Changing the leading instrument of an ensemble,
3. Switching execution mode of an instrument ensemble to independent and back,

Solution strategies:

- Basic services provided by CONCERT middleware.
- Beat sequences for synchronization among instrument ensembles.
- Network database for communication among isntrument ensembles.
- Instrument-specific protocols for intro-ensemble communication.

Interfacing with Java

Various possibilities provided by Java:

- Synchronous execution of external proceses ("system calls").
- Message passing via sockets.
- Java IDL for CORBA-compliant services.
- Java JNI for integrating C/C++ object code.
- Java RMI for interfacing with remote Java objects.
- JavaBeans for developing Java componentware.
- Java DBC for SQL database connectivity.

The Java side is not the problem!

Interfacing with External Applications

- Closed World
  - Most applications designed for interactive use.
  - Command line arguments, environment variables (startup time), resource properties (windows environment), standard I/O, file I/O, special hooks (callback programs).
  - Embedding difficult to impossible.
- Dynamic linking
  - Componentware.
  - CORBA/OpenDoc, DCOM, JavaBeans, ...
  - Embedding easy (but not widely used).
- External Links
  - Facilities for communication with external programs.
  - Mathematica's *MathLink*.
  - Embedding not easy but possible.

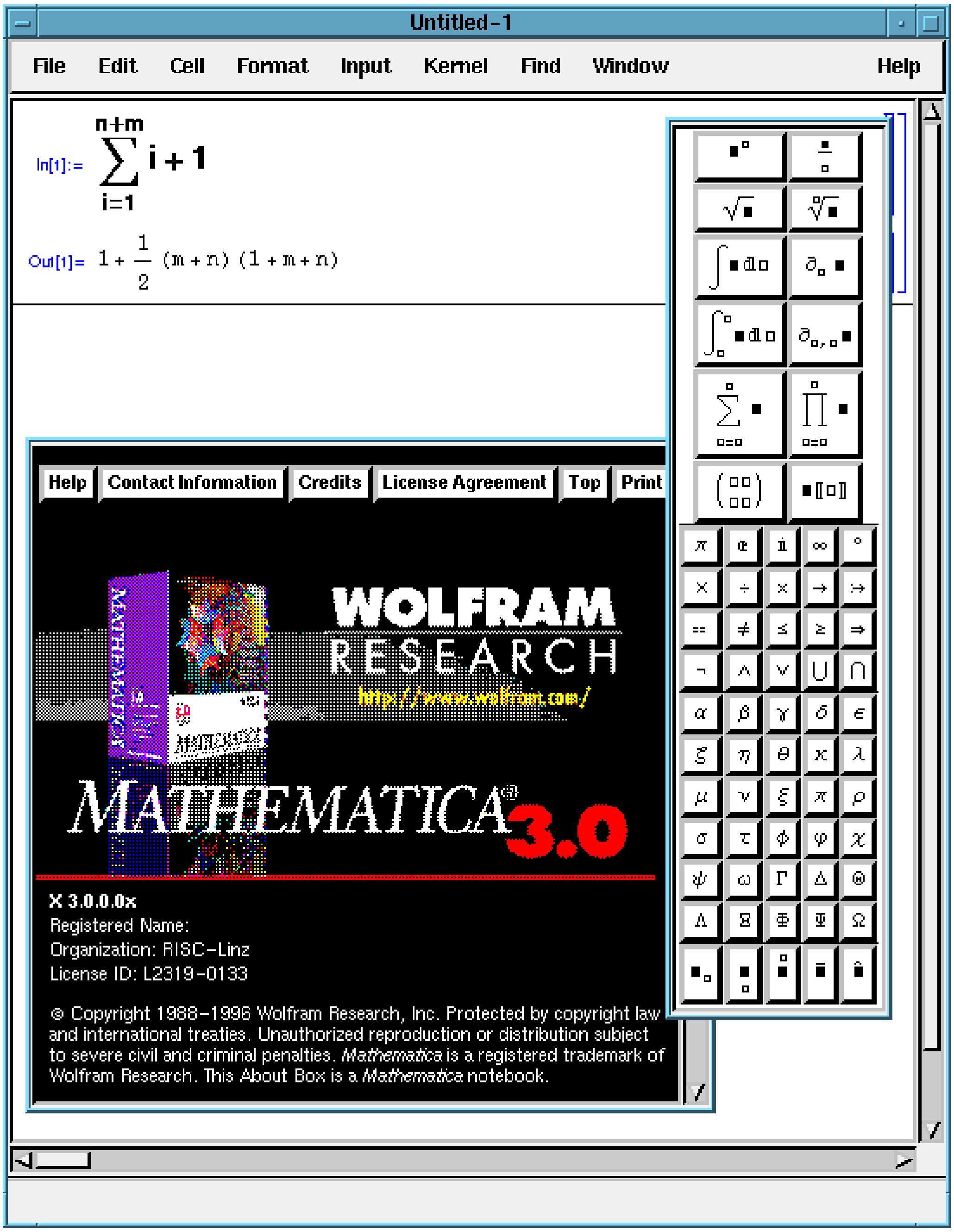
The external side is the problem!

Example: Mathematica

Basic system structure:

- Frontend process.
  - Interaction with user.
  - Notebook manipulation.
- Kernel process.
  - Actual computation.
  - Simple text-base interface.
- Communication by *MathLink*.
  - Message passing model.
  - C-based programming library.
  - Implemented by sockets.

Mathematica Frontend

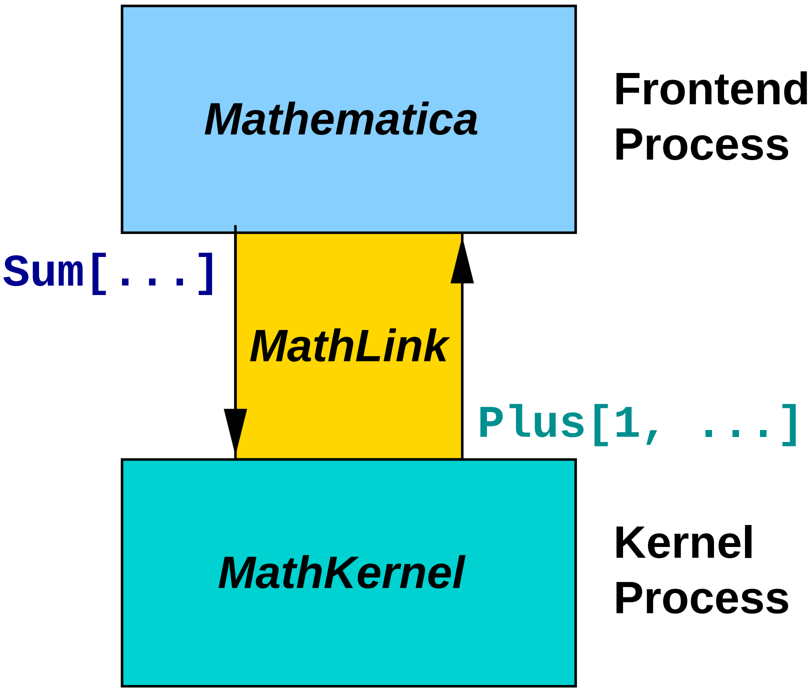


Mathematica Kernel

```
iris!21> math
Mathematica 3.0 for Silicon Graphics
Copyright 1988-96 Wolfram Research, Inc.
-- Motif graphics initialized --

In[1]:= Sum[i+1, i, 0, n+m]

                    (m + n) (1 + m + n)
Out[1]= 1 + m + n + -------------------
                             2

In[2]:=
```

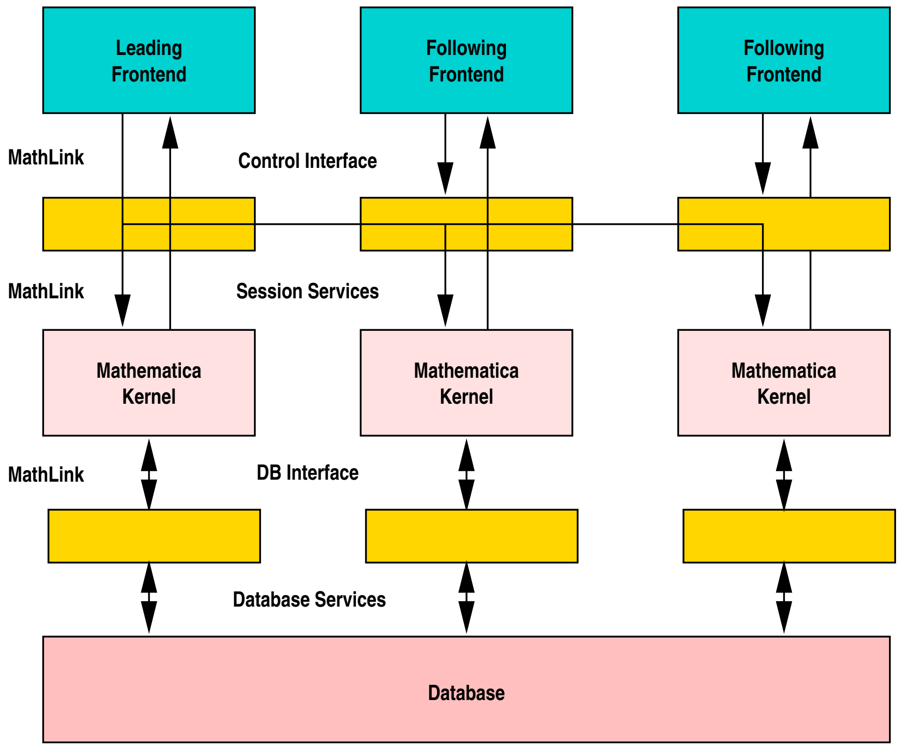
MathLink

```
iris!24> ps -ef | fgrep math
schreine 29681 /usr/.../Mathematica -topDirect
schreine 29685 /usr/.../MathKernel -mathlink
```



Mathematica as a CONCERT Instrument

Intercept communication between frontend and kernel:



Conclusions

- Software architecture for distributed education systems.
  - Not surface (audio/video/shared screens).
  - Logical structure (shared tools).
- Single lecturer scenarios:
  - Support for session development.
  - Support for session operation.
- Current plans:
  - No funding support for the “big thing”.
  - But: implementing one particular instrument!
  - Distributed Education Environment based on Mathematica

Systems for distributed education are urgently needed!