CONCERT
A Software Architecture for Coordinating Education Sessions in Distributed Environments*

Bruno Buchberger
Bruno.Buchberger@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/buchberg

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

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

RISC-Linz Technical Report 97-04†

February 17, 1997

Abstract

This paper sketches an environment for distributed education that supports the lecturer in preparing and performing a computer-supported session over the network ("virtual classroom"). This task is particularly challenging because it requires the coordination of *concurrent activities* operating in *real-time* on *objects of different types* located in a *distributed network environment*.

We describe the CONCERT software architecture that can be implemented with realistic efforts in the mid-term future and allows non-expert users to perform such a task. Since technology is rapidly evolving, we focus on the design of an abstract architecture under which various technologies can be integrated and on top of which different programming interfaces can be built. This architecture can be implemented in form of an application framework i.e. of an object-oriented software library into which application/system-specific components can be plugged.

Our investigation is accompanied by a detailed analysis of the current state of the art (i.e. technology already in existence or projected to be available within one year) which can deliver components for the proposed framework and serve as the basis of its implementation.

*Supported by IPA.

†This paper is a linearized version of the hypertext document available at http://www.risc.uni-linz.ac.at/people/schreine/papers/ipa97. The printable PostScript file is available at http://www. risc.uni-linz.ac.at/people/schreine/papers/ipa97/report-main.ps.gz

Contents

1 Introduction

We describe the design of a system that supports a lecturer in preparing and performing education sessions in which the audience participates from remote locations via networked computers. Most current projects on "virtual classrooms" and "tele-teaching" focus on an straight-forward extension of the traditional class room model by broadcasting a lecture with video-conferencing technologies [MUN96][CAE96].

Our approach is different because we base it on the logical structure of a modern computer-supported education session. In our vision such a session is a concert of multiple software instruments that concurrently perform various tasks with the lecturer conducting their use and interaction. Audio may be one such instrument, a slide presentation another, the demonstration of an application program with the possibility of running interactive exercises by the audience is a third one.

In such a scenario, the application of video (while nice if available) plays a secondary role since most of a session's information contents is actually provided by the other software instruments. Consequently computer-based distributed education sessions may cope with much tighter constraints on network bandwidth than distributed classrooms fundamentally based on video technology.

However, organizing education sessions of this kind is a challenging task that requires the coordination of concurrent activities operating in real-time on objects of different types located in a distributed environment. Currently this is hardly feasible for computer experts with experience in network programming. Our goal is to break down the complexity of this task to the level of preparing an animated slide show with the help of the presentation tools available today.

Since technology is rapidly evolving with many competing systems and standards promoted by different vendors, the system design is based on an abstract reference architecture named CONCERT under which various technologies can be integrated and on top of which different coordination languages can be built. The CONCERT architecture can be implemented with realistic efforts in the mid-term future based on already available systems and technologies yielding an application framework i.e. an object-oriented software library into which application/system-specific components can be plugged.

The rest of this paper is organized as follows:

Preparatory Work (Section 2) presents our research work and results that have preceded and prepared the CONCERT design.

Coordinating Education Sessions (Section 3) sketches our vision of a distributed education sessions setting the requirements for our system design.

The CONCERT Architecture (Section 4) describes our design of an abstract reference architecture for coordinating distributed education sessions.

Building CONCERT Instruments (Section 5) discusses how to integrate existing software systems as framework components.

Developing CONCERT Tools (Section 6) sketches various software tools that support the construction and the performance of distributed education sessions.

Implementing CONCERT (Section 7) discusses how to implement the abstract architecture using available technologies and systems.

Conclusion (Section 8) summarizes the main aspects of our design.

Appendix Technologies for Distributed SystemsA contains a review of relevant technologies that have influenced our design; we have also delegated most references to related work to this section.

2 Preparatory Work

Our design of the CONCERT architecture for distributed education has been shaped by the results of many former activities of our own and of other researchers at RISC-Linz (the Research Institute for Symbolic Computation in Hagenberg/Linz, Austria):

- We have developed various systems and languages for coordinating concurrent activities on parallel/distributed systems with focus on applications in symbolic computation.
- We have gained substantial expertise in the use and further development of interactive mathematical software systems in university and college education.

Both aspects are documented in the following subsections.

2.1 Coordination Programming

Research on parallel and distributed computing at RISC-Linz has its origins in the 1970s [Buc78]. In the frame of this research, various programming languages have been developed and used to coordinate concurrent activities of symbolic computation systems on multi-processors and computer networks. Among the more recent activities are the following:

pD pD [Sch94a] [Sch95] [Sch96a] is a para-functional language that has been developed as a high-level programming interface for the parallel computer algebra kernel PACLIB (see below). Parallelism may be expressed by several forms of annotations; the compiler generates C code with PACLIB task creation and synchronization statements. Via an external interface the programmer may call SACLIB functions from the functional layer thus using pD as a parallel *coordination language* for computer algebra applications. The performance of the generated code is practically the same as for manually implemented PACLIB programs.

PACLIB PACLIB [HSN+92] [SH93b] [SH93a] [SH93c] [Sch94b] [HNS95] is a shared memory parallel variant of the runtime kernel of the computer algebra library SACLIB [BCE+92] for symbolic and algebraic computation. PACLIB provides very efficient light-weight tasks, parallel garbage-collection, non-determinism, and lazy task creation. Sequential SACLIB code may be freely embedded in PACLIB with the same efficiency as in the sequential system thus using PACLIB as a *coordination system* for SACLIB algorithms. A variety of computer algebra applications have been developed in PACLIB on a Sequent Symmetry system with 20 processors [HL94] [LP94].

||MAPLE|| ||MAPLE|| (parallel MAPLE) [Sie92] [Sie93b] [Sie93a] is a concurrent variant of the computer algebra system MAPLE. The system uses the declarative Guarded Horn Clause language STRAND [FT89a] as a parallel *coordination language* to map MAPLE processes to processors. Thus it becomes possible to call from a parallel STRAND program MAPLE functions for algebraic computation. Consequently the programmer may write sequential parts of a computer algebra program in MAPLE and use STRAND for parallelization, communication and synchronization. The system runs on shared and distributed memory multi-processors and on computer networks.

STURM STURM is a C++ library for sequential and parallel symbolic computation [HNSS94] [BHP96]. As a successor to PACLIB, it consists of several packages each of which implements classes and operations in a certain domain. The library provides features for automatic memory management, parallel computation on shared and distributed memory machines and on networks of workstations, operations over numerical domains such as floating point numbers and integers and over domains such as polynomials, matrices, etc. Various high-level application packages have been developed on top of this library.

Computer Networks At RISC-Linz computer networks have been employed for several years as parallel computing platforms. Using workstation networks, characteristic-set-based algorithms [Wan91] and quantifier elimination methods [Hon92] have been parallelized and a distributed variant of the *parallel coordination language* Linda has been developed [Zet92]. Likewise, distributed variants of symbolic computation packages have been developed on top of the multi-processor STURM kernel.

2.2 Computer-Supported Mathematical Education

Creating environments for distributed education sessions is at the natural intersection of various research and teaching projects currently pursued at RISC-Linz and is an urgent necessity for these projects.

2.2.1 Interactive Books

Buchberger's Groebner bases method [Buc85] is now one of the standard methods in computer algebra software systems with an estimated 2 million installations worldwide. Still, there is no interactive textbook available on this method. The author is working on such a book that computer-support both the proof parts (using results from the Theorema project) and the computational parts (using a Mathematica implementation of the method) of the textbook and make it accessible through the web.

2.2.2 The Theorema Project

The Theorema project aims at computer-support of mathematical proving. The distinctive features of this project are:

- the intimate interaction with an existing (symbolic) mathematical software system, in particular Mathematica,
- the development of special proof techniques for each "domain" of mathematics, where domains are defined and automatically generated by "functors",
- the imitation of human proof techniques and automated generation of easy-to-read proofs natural language,
- the development of interactive browsers for reading and studying proofs that allow the reader to determine interactively the desired level of "verbosity".

For more details see [Buc96a] [Buc96c] [Buc96b] [Buc96d] [Buc97a].

2.2.3 Courses

From the variety of courses given by RISC-Linz we take two examples that are prepared to profit from support by computer environment for distributed education.

Thinking, Speaking, Writing This course is regularly given to the graduate (diploma, PhD) students of RISC-Linz and aims at training and improving the six basic working techniques of mathematicians and computer scientists: thinking, speaking, writing, listening, reading, cooperating (i.e. proving, giving talks and lectures, writing publications, analyzing problems, working with the literature, working in groups). For more details see [Buc97b].

Mathematics for Computer Science This course series is taught by RISC-Linz faculty in the first four semesters for undergraduate computer science students at the University of Linz and for the students of the College for Software Engineering in Hagenberg (which was established by RISC-Linz three years ago). As a distinctive feature, it emphasizes those subjects of mathematics that are motivated by computer science and computer applications. In particular, it starts with an intensive training in the general problem solving techniques of mathematics (including formal reasoning) and *emphasizes the interactive use of (symbolic) mathematical software systems.* For more details see [BF81] [Buc93] [Win94] [Wal95].

2.2.4 The College of Media Techniques and Design

Recently, RISC-Linz has established yet another college in Hagenberg, the College for Media Techniques and Design [BJ+95] [FHS96]. This college trains, in a four years curriculum, engineers in the new area of communication and media technology. The distinctive feature of this college is the balance between teaching both the technology as well as the arts and design aspect of the field. The expertise of the faculty of this college and the facilities built up there can be integrated into the development of the CONCERT framework in a natural way.

3 Coordinating Education Sessions

Our goal is to enable computer-supported education sessions where the lecturer can utilize new technologies in order to

1. improve the didactical value of the presentation by integrating computer programs developed for the particular education domain (like e.g. Wolfram Research's Mathematica [Wol96] [Wol97] in mathematical education),
2. allow the participation of non-local users via wide-area computer networks (via generally available Internet infrastructure or via connections especially established for the session).

There may be other motivations as well but in the end most of them can be reduced to one or both of the above (e.g. supporting the participation of physically handicapped users).

3.1 Session Organization

The lecturer shall be enabled to prepare the session by developing a "session program" i.e. preparing the basic materials (texts, images, animations, hypermedia documents, computer programs, ...) and coordinating their use by invoking corresponding applications (browsers, viewers, interpreters, ...), managing their causal and temporal relationships, and potentially handling their outputs (e.g. exercise results). Additionally the session may integrate certain application-independent facilities (whiteboards, audio connections, ...). The session may be developed, tested, and debugged offline i.e. in a simulation environment with the help of various supporting tools.

The execution of the education session is initiated on a certain host computer that serves as the lecturer's interface to the session. This computer may be equipped with audio/video devices if also users in remote locations shall be enabled to participate in the session. Single-classroom presentations generally cope without such devices; here the primary focus is typically the integration of domain-specific computer support.

After the session has been initiated, additional hosts may request access to the session; each session request consist of the address of the host and the role that the host wants to play in the session. In simple education sessions, there may be just one lecturer host and a set of audience hosts; more complex sessions may be composed of a central supervisor, several "floor managers" in the respective locations, several lecturers and various sets of clients (e.g. active participants and quiet listeners). Access may be granted manually or on the basis of rules taking into account address/role information (no restriction, participants restricted to particular network domains, ...). The overall state of the session and its participants may be monitored and controlled from the supervising host.

Each participating host loads and executes that part of the session program that corresponds to its particular role. This session program is in charge of loading the particular media objects, invoking the corresponding applications, synchronizing and coordinating their interaction with each other and with processes on other hosts and handling the generated output.

3.2 Session Components

In the following, we list a couple of applications/components which may cooperate to form a distributed education session:

Audio/Video Real-time audio/video streams may be multicast from the lecturer to the audience (lecture mode), from the audience to the lecturer (question/answer mode), and among the audience (discussion mode). The transmission of video information, while conceptually feasible also on the basis of public Internet infrastructure (MBONE), practically requires the use of dedicated connections (ISDN, ATM).

However, our experience shows [ATM96] that decent audio connections are much more important than video broadcasts because it is the voice which delivers the actual information contents of a lecture (provided that it is supported by the techniques presented below) while video usually just shows the lecturer's appearance. Fortunately audio can be transmitted in reasonable quality on generally available infrastructure (as demonstrated by the audio extensions of various WWW browsers).

Whiteboards The core of highschool and university teaching is still the blackboard/whiteboard. "Electronic whiteboards" may deliver an adequate substitute for such kind of presentations using conventional desk-based pen-board devices[1]. The information content of such whiteboards can be transmitted to the audience locations using low-bandwidth connections (in contrast to video transmissions from physical boards). The same techniques may be also used in the other direction (students presenting at the whiteboard) but not all locations/host will in general be equipped with adequate input devices.

Electronic whiteboards are certainly required for various presentation modes (gradual developments of chains of reasoning) but in the context of virtual classrooms their role is that of a supplement rather than that of a core presentation technology.

[1] Wall-sized pen-boards [MUN96] are nice gimmicks but certainly not a must for most applications.

Slide Shows As he core of most virtual classroom presentations we expect the "slide show" i.e. sets of text documents augmented by images and possibly with animations. Slide shows may be developed manually, with various typesetting/word processing tools (SliTeX, MS Word) or with special presentation packages (MS Powerpoint). The rise of the WWW also suggests the use of HTML-based hypertext documents (potentially Java/JavaScript/ActiveX enhanced) for this purpose with the additional flexibility of crosslinking and hypertext navigation facilities and references to external resources.

The essential characteristics of slide shows is that they are available in electronic format[2] and can be transmitted with comparatively little bandwidth requirements, possibly ahead of the session. However, for their presentation the lecturer needs explicit control/synchronization facilities (even for automatically running slide presentations, "start/stop/rewind" buttons are necessary). This requires interfacing with the corresponding viewers/browsers/interpreters, which may be easy (freeware available in source code), possible (WWW browser plugins, application with "dynamic linking" facilities), or hard to impossible ("closed-world" applications).

General Applications A generalization of the rather straight-forward slide show is the integration of any kind of application programs into distributed education sessions. The selection and use of such systems depends on the particular education domain which makes their particular characteristics rather hard to predict. As an example, we therefore concentrate on the application of the computer algebra system Mathematica in mathematical education [Wol96] [Wol97].

The core of the system is an interpreter for the Mathematica programming language in which a large number of packages of mathematical algorithms have been implemented. Highly elaborated visualization features support the display of results in graphical form. Mathematica "notebooks" allows the typesetting of hypermedia documents that embed executable Mathematica code. "MathLink" allows (network) communication between Mathematica and external programs via serial channels.

A lecturer might use Mathematica simply in an interactive way, entering commands and running programs. The Mathematica window image should then be multicast to the audience. The lecturer might also prepare a Mathematica notebook and use the system as a presentation tool to display mathematical slides, possibly executing embedded Mathematica code. While it is also here possible to simply multicast the window image, a better use of network resources is to load the notebook on each participating host and invoke remote instances of Mathematica executing the local copies of the notebook. However, as described for slide shows, then synchronization with (or control by) the lecturer's presentation is required.

An even more advanced use of the system is possible, if the lecturer can switch control to the remote instances of Mathematica allowing the audience to run own test examples using the downloaded notebook ("exercise

[2] Even manually written "legacy slides" may be scanned as images in the course of the preparation.

mode") and resume control when returning to presentation mode. For discussing questions and problems (see the next item), it would be also necessary to broadcast the particular screen image of a particular Mathematica invocation to lecturer and audience.

In a more general setting, the system might be also used in an "assessment mode" where questions are posed and results have to be computed/programmed/proved. These results have to be collected in stored for later retrieval from the lecturer.

Feedback For use in education, it is also necessary to allow feedback from the audience. The simplest support could be a "raise hand" button that when pressed by some participant results in a message on the lecturer's screen. Having admitted the interruption, the participant may pose a question using a simple text-based interface, whiteboards, or audio/video. The question may be transmitted to the lecturer only or broadcast to all of the audience. More complex communication patterns (multi-user conferencing modes) might be supported as well; managing the complexity of such scenarios however represents a difficult task.

Autonomous Agents Coordinating education sessions with a large audience in multiple locations may become soon to complex for the lecturer. Instead of installing separate floor managers, the use of autonomous agents may be investigated. Such agents may handle routine tasks that only require "semi-intelligent" reasoning capabilities but that put much strain on the lecturer: filtering streams of "raise hand" requests such that only one at a time is forwarded to the lecturer (using some scheduling heuristics and possibly blocking too many requests from single participants), handling session connection requests (using rules to decide which host may participate in which role), monitoring user activities and providing simple aid in "exercise mode", monitoring network connections and adjusting the quality of services or raising warnings, etc.

3.3 Requirements

From the scenarios sketched in the previous subsections, we derive the following requirements that from the users' (lecturer and audience) point of view must be met by an architecture supporting the coordination of distributed education sessions:

Developing Sessions A lecturer must be able to prepare distributed education sessions without being an expert in network-programming. The difficulty of such an endeavor must in the order of magnitude of developing animated slide shows with the help conventional presentation tools or preparing hypermedia documents with modern publishing tools. Obviously the mental complexity of handling concurrent activities in different locations is larger than for a single-threaded presentation, still the abstractions provided by the system must be in the language of the lecturer rather than in the language of the system. Tools must be provided that support the developing, testing, and debugging process.

Performing Sessions A lecturer must be able to execute distributed education sessions without being overloaded by management tasks. While other scenarios are thinkable, we focus on applications where a single lecturer is performing and coordinating a session without the help of a crew of supporting technicians. For single classroom sessions, no additional help at all should be required; for distributed sessions with a couple of classrooms a separate floor manager might be advisable.

Accessing Sessions It must be possible to restrict access to a session but open access should be also supported. Different hosts may play different roles in a session. To contact a session, just the availability of a certain startup software and the knowledge of a simple network address should be required. Access should be possible before the session is started but also during a running session.

Integrating Components It must be possible to integrate external application programs (browsers, viewers, domain-specific applications) into the session. This requirement is subject to the restriction that the external applications may provide various degrees of interoperability (applications with dynamic linking support versus "closed world" applications). However, as soon as the necessary "interface software" for some application has been developed, it should be possible to use this application as a simple building block for a session.

Coordinating Components It must be possible to integrate real-time audio and video streams into a session (subject to bandwidth restrictions). Electronic whiteboards must be supported. Transmission of window images from general applications must be supported. Support of slide presentations should be available where the presentation is executed locally but synchronized centrally. Likewise it should be possible to invoke from the lecturer's host general applications on the audience hosts with possibilities for local use by the audience (and to revoke control again). It must be possible to run several applications simultaneously, control their interaction and communication. Input data for these applications may be objects provided by the lecturer; output data may be collected by the lecturer.

Enabling Feedback Feedback from the audience to the lecturer must be possible ("raise hands"). It must be possible to transmit this feedback to the rest of the audience (text, audio/video, whiteboards, window images, ...).

Integrating Agents It must be possible to integrate autonomous agents that are loaded and executed on the client hosts.

Additionally, general requirements of distributed systems should be met: a certain degree of fault tolerance, reasonable security standards, support of heterogeneous environments with various kinds of machines and communication protocols.

4 The CONCERT Architecture

The CONCERT architecture offers a set of services for building distributed education sessions. It sits between the

- low-level network system and session components on the one side, and
- the developed education sessions on the other side.

The purpose of the architecture is twofold:

1. It represents a mental model providing the organizational framework for designing a distributed education session,
2. It provides an API (application programming interface) for the concrete session program implementing the distributed education session.

The core issue of the architecture is *abstraction* i.e. restricting the various facilities and technical details of the lower levels to a high-level standard interface targeted towards requirements of the application domain. The architecture presented in this section therefore does not provide a framework for building distributed applications in general but for a framework targeted to building *distributed education sessions.*

We will describe these two facets in the following subsections in turn.

4.1 The Mental Model

Our mental model of a distributed education session is inspired by the SPMD (single program, multiple data) model of parallel processing where a single program is executed on multiple processors, asynchronously in principle but synchronized at exactly specified points.

4.1.1 Concerts

The model is based on the following concepts whose relationship is explained below (see Figure 1):

Concert A *concert* is an instance of an education session in action; it may be distributed among various *hosts* and involve multiple *players.*

Host A host is a computer, possibly in a remote location, connected to the concert via a network. Each host accommodates one or multiple participating players.

Player A player is a human that handles (interacts with) one or several *instruments* on the accommodating host.

Score The *score* is the concert program according to which the players operate.

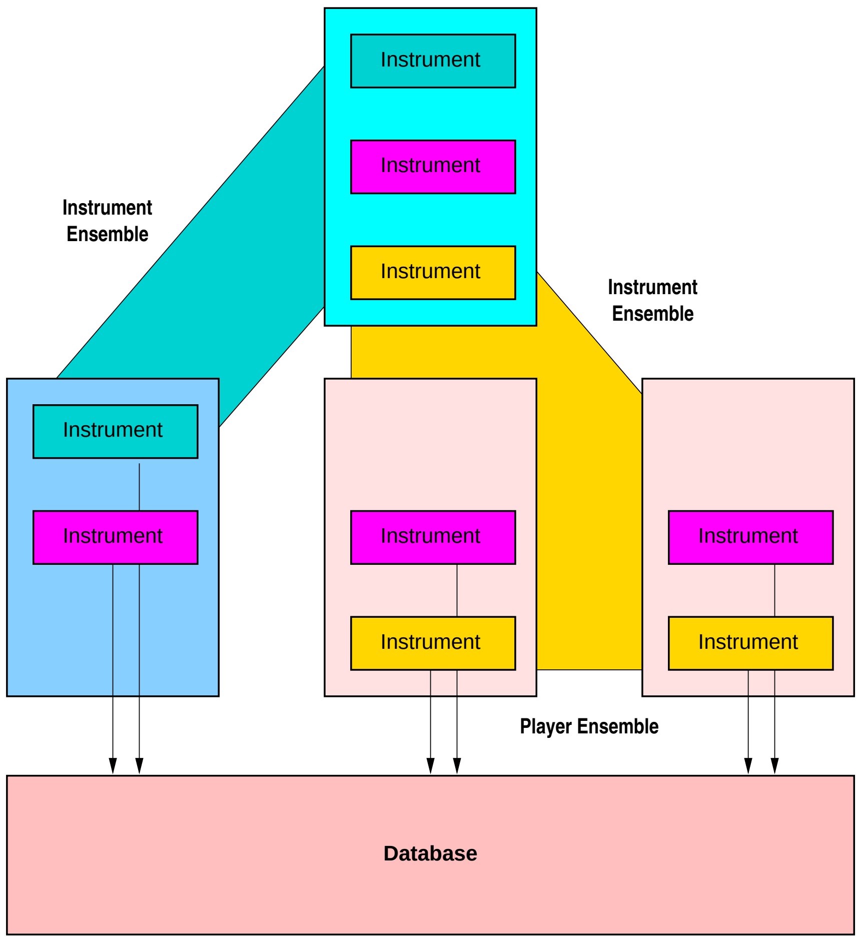

Figure 1: The CONCERT Architecture

Instrument Each instrument is an application component (slide viewer, audio player, etc) that executes independently as an asynchronous process.

Ensembles Players are organized in *player ensembles* which handle the same set of instruments and are generally treated alike. The set of instruments handled by an player ensemble is grouped into the various *instrument ensembles*.

Beat Sequence Each instrument ensemble has an associated *beat sequence* (a stream of signals) by which it synchronizes with other instrument ensembles.

Database A global database contains all the auxiliary objects required for the operation of the instruments (documents, files, programs, ...).

Each player may participate in a concert by providing her credentials for a particular player ensemble, receiving the program score and executing the part of the score corresponding to the ensemble.

In such a model of a distributed educated sessions, the part of the lecturer may be twofold:

- as a *composer* she creates the score according to which the concert operates;
- as a *conductor* she coordinates the interaction of the concert players.

However the second role is *not* implied and not even especially emphasized by our model, because it is the score itself that actually coordinates the interaction of players where the lecturer is just one of them. In fact scores may be composed that do not designate any player with special roles such that also fully democratic "self-conducting" education sessions become possible.

4.1.2 Ensembles

A concert consists of the execution of multiple instruments; for each member of a player ensemble, the same set of instruments is available: a monitor may be only available for a single participant (the lecturer), a slide presentation component may be available for all participants; an audio component may be available only for the lecturer (sender-side) and for the remote participants (receiver-side) because the local participants hear the lecturer anyway.

An instrument ensemble may operate in two modes:

Leader Mode One particular member of an instrument ensemble (the *leader*) controls the activities of all other members (the *followers*). As an example, the lecturer's activities on her instance of the slide presentation tool are replicated on the instances of the audience.

Different instruments of an ensemble may be designated leader in the course of the time, e.g. control of a whiteboard component may be switched from the lecturer to a student and back.

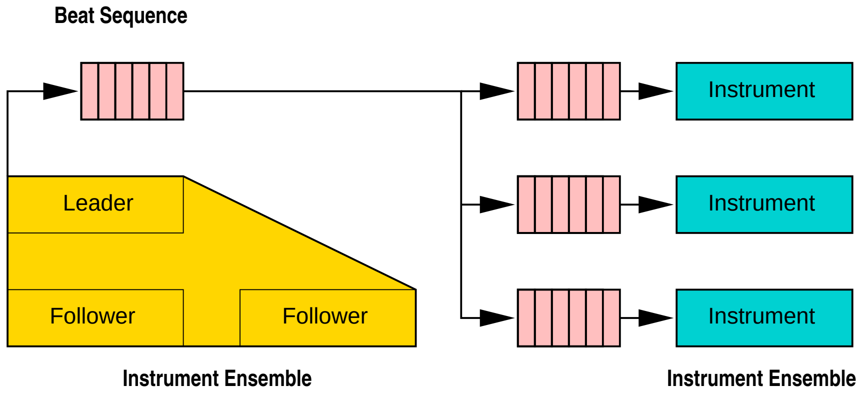

Figure 2: A Beat Sequence

Independent Mode The activities of all members of an instrument ensemble proceed independently ("jam mode"). For instance, in a training session each participant may operate on his instance of an application program.

The execution of concert score if for each player restricted to the part of her ensemble only. Each instrument runs as a parallel process whose start and termination is triggered by the score. Moreover, the score may switch the operation mode of a instrument from leader mode to independent mode and vice versa (possibly designating a new leader). Furthermore, the score may invoke special instrument operations such as hiding a window and showing it again.

4.1.3 Beat Sequences

Synchronization between different instrument ensembles is based on the concept of *beat sequences* (see Figure 2). Each instrument ensemble has associated to it a sequence of (not necessarily uniquely) named beat signals. This sequence is generated by the respective leader of the ensemble (when in independent mode, no beats are generated). An instrument ensemble may wait for the next beat generated by another ensemble or it may wait for a specifically named beat from that ensemble (and skip all other beats).

Synchronization between instrument ensembles takes place on an *individual* basis: each instrument retrieves an independent copy of the beat sequence of the ensemble it synchronizes with. This has the following consequences:

- *Ensemble mates are not punished.* The individual members of a ensemble operate *asynchronously*. When the host accommodating some instrument fails or the connection to its leader is slow, the other followers are *not* blocked.

- *Late-comers are not punished.* When a new players is added in the course of the session, its instruments retrieve all beats generated so far as the other members of the ensemble did.

Both features are important in realistic application contexts.

4.1.4 Communication

Communication between instruments takes place in the following way:

- From the score to an instrument ensemble, by passing (the same) arguments to all members when the ensemble is activated.
- From the leading instrument to its followers, by having the followers replicate the activities of the leader.
- Between different instrument ensembles, via the underlying database from which objects can be retrieved and in which objects can be stored.

Database objects are addressed by qualified identifiers; by incorporating its particular identity as an identifier component, different instruments of an ensemble can retrieve/store different objects. Updates in the database must be synchronized using beat sequences as described above.

4.2 Application Programming Interface

We suggest an implementation of the mental model described in the previous section using the following technological framework (see also Section 7 :

- A concert score is represented by a program in Java VM format that is downloaded by every concert player and executed by a local JVM interpreter. The implementation in JVM secures portability for all kinds of player hosts.
- A concert is originated on a host where the *Concert Manager* is running. This program listens on some port for concert requests and answers them by transmitting the corresponding score registered by the lecturer (the "concert composer").
- Any player may request access to the session by starting a *Concert Client* on her local host with the address of some machine where a concert manager is listening. The concert client contacts the concert master that provides a list of available scores for downloading and the various player ensembles for each concert with the number of players connected.
- When the concert score starts execution on the player's host, it asks for the player ensemble where participation is requested and verifies the corresponding access permissions (password, ...). If successful, the score contacts the concert manager and registers the player as a new member of a particular ensemble. Thus connection to to the other players is established.

The following subsections describe the low-level programming interface on which concert scores are based. Section 6.1 shows how this may serve as the basis abstract coordination models.

4.2.1 Concert Example

The core of the concert score is a class method of the following kind describing the overall structure of a concert:

```
run()
{
  monitor.start();              // start monitor on lecturer's host
  slides.start("slides");       // start slide show controlled by monitor

  monitor.wait("start");        // wait for lecturer's start signal
  audio.start();                // start audio instrument

  slides.wait("example");       // wait until example1 slide

  math.start("notebook");       // start mathematica notebook concert
  slides.wait();                // wait for lecturer prompting next slide
  math.hide();                  // hide mathematica

  slides.wait("exercise");      // wait for exercise slide

  math.show();                  // show mathematica for exercise work
  math.async();                 // let audience work independently
  monitor.wait("interrupt");    // wait for lecturer signal

  math.leader(lecturer);        // lecturer becomes math leader again
  slides.hide();                // hide slide show
  whiteboard.start();           // start whiteboard

  monitor.wait("continue");     // wait for lecturer signal on monitor
  whiteboard.stop();            // terminate whiteboard
  math.stop();                  // terminate mathematica
  slides.show();                // show slide show again

  monitor.wait("stop");         // wait for lecturer signal on monitor
  slides.stop();                // terminate mathematica
  audio.stop();                 // terminate audio connection
  monitor.stop();               // terminate monitor
}
```

This concert consists of the following instruments whose construction is described later:

- monitor, a monitor instrument that allows global concert control,
- audio, an audio instrument transmitting the lecturer's voice to the remote audience,
- slides, a slide presentation tool controlled by the lecturer,
- math, the computer algebra system Mathematica for notebook presentation and interactive exercises,
- whiteboard, a whiteboard utility for drawing and hand-writing.

The concert starts by invoking the monitor on the lecturer's host (the association of instruments to hosts is described later). The slide presentation instrument is started receiving as its argument the name of the presentation stored in the database. All this takes place before the actual concert starts giving the audience hosts time to connect to the concert and watching the cover slide of the presentation.

When the lecturer has given the start signal via the concert monitor (which generates a beat with the name of the key pressed), an audio ensemble is created transmitting the lecturer's voice to the remote audience. The lecturer starts presentation with a set of slides explicitly controlling the leader of the slide viewer ensemble; the audience can watch the corresponding slides via their followers. Furthermore, for each presented slide, the leading slide viewer generates a beat with the name of the slide.

When the presentation has reached slide "example", a Mathematica ensemble is created and brought into independent mode giving the audience the possibility of interactive exercising. When the lecturer presses an "interrupt" button in the monitor, the Mathematica ensemble brought back into follower mode with the lecturer's instrument being the leader. The slide presentation is hidden; instead a whiteboard ensemble is created with the lecturer controlling the leading instrument. The lecturer continues presentation using Mathematica and drawings on the whiteboard.

When the lecturer presses a "continue" button on the monitor, both Mathematica and the whiteboard ensembles are terminated. The lecturer brings the slide ensemble into the foreground and continues the concert with a set of slide presentations. When he finally presses the monitor's "stop" button, all ensembles are dissolved and the concert terminates.

Above demonstration sketches the essential core of the programming interface for developing interactive education sessions. Its essential characteristics is the SPMD style of control with focus on a single thread of control that manages many replicated instances of asynchronously executing processes from a leading process. Explicit synchronization takes place by waiting for specific signals from various instruments (e.g. from a monitor that emits signals with button names for every button pressed, or a a slide viewer that emits signals with slide names for every slide displayed).

Distributed communication takes place "behind the curtains" by the use of predefined components that load the objects they operate on from the network database (such as the slide viewer and Mathematica) and store potential results there.

We summarizing the methods presented up to now in a class

```
class Instrument
{
  ...
  start();
  stop();

  async();
  leader(Instance);
```

```
  String wait();
  wait(String)
  ...
}
```

with methods for starting a instrument, terminating it, putting it into independent execution mode, resetting it into following mode with denoting a particular instrument instance as leader, waiting for the next signal generated by the ensemble and returning its name and waiting for a signal of the specified name, respectively. Subclasses may override and extend the interface e.g. in

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

for iconifying and deiconifying a slide presentation.

More details about the construction of a concert and individual instruments are provided in the following subsections.

4.2.2 Concert Composition

Each concert is represented by a score object that is given a unique name to be displayed in a selection menu when the concert manager is contacted. This object is an instance of a superclass of Concert that provides virtual methods to be overridden by the concrete score:

```
class Concert
{
   ...
   run();
   ...
}

class MyCourse extends Concert
{
   ...
   run()  ...  // shown above
   ...
}

MyCourse mycourse = new MyCourse("Computer Algebra I");
```

Before defining the various player ensembles to be admitted to the concert, we construct various policies that restrict player ensemble access to particular network domains or hosts and optionally introduce password protection. For example,

```
class Control
{
  ...
  adddomain(Domain)     // add domain
  addhost(Host)         // add host
  addany();             // allow access by any host
  ...
  Control(Bool)         // with or without password?
  ...
}

Control lecturerControl = new Control(true); // password protection
myhost.addany();                              // lecturer uses any host

Control localControl = new Control(false);    // the local audience
mydomain.adddomain(); // only my domain

Control all = new Control(true);        // the remote audience
remote.addany();                              // anybody with password
```

The controls are not bound statically to concrete networks/hosts/passwords; the binding takes place when the score is registered at the concert manager; based on a score password, the lecturer can interactively provide/manipulate this information.

Player ensembles are defined via the access controls and linked to the concert being constructed. Ensembles may be also compound to form larger ensembles:

```
Ensemble lecturer = new Ensemble(mycourse, lecturerControl);
Ensemble localAudience = new Ensemble(mycourse, localControl);
Ensemble remoteAudience = new Ensemble(mycourse, remoteControl);

Ensemble audience = new CompoundEnsemble();
audience.add(localAudience);
audience.add(remoteAudience);

Ensemble all = new CompoundEnsemble();
all.add(lecturer);
all.add(audience);
```

Now we define the various instruments utilized by the concert and establish their relation to the various ensembles specifying where the leader is to be located and where the followers are invoked:

```
class Monitor extends Instrument
{
  ...
}
...
Monitor monitor = new Monitor("monitor", lecturer, all);
Audio audio = new Audio("audio", lecturer, remoteAudience);
Slides slides = new Slides("slides", lecturer, audience);
Mathematica math = new Mathematica("math", lecturer, audience);
Whiteboard whiteboard = new Whiteboard("board", lecturer, audience);
```

This list encompasses

- a monitor running on the lecturer host providing control over all of the concert,
- a instrument establishing audio connection between lecturer and remote audience,
- a slide viewer displaying the slides presented on the lecturer host to all of the audience,
- a Mathematica ensemble controlled by the lecturer and linked to the audience,
- an electronic whiteboard ensemble controlled by the lecturer and linked to the audience.

All the corresponding instrument classes have to be defined as subclasses of Instrument and implement the interface to the various applications integrated in the concert. The instruments will be only activated in the specified ensembles (e.g. no audio connection for the local audience); the string passed for the creation of the instrument is a unique name for the ensemble to which the instrument is to be attached.

We still have define code for the various concert instruments, which is described in Section 5. This requires access to several lower-level services that are sketched in the following subsections.

4.2.3 Database Services

For implementing a instrument method

```
class Slides extends Instrument
{
  ...
  start(String) { ... }
  ...
}
```

access to the global database is required in order to load the slide presentation identified by the parameter string. Likewise, a instrument may output objects to be stored in the database. Native code of external applications may be stored in the database and retrieved on demand by the corresponding concert instruments on demand (thus avoiding local installations of the program). Last but not least, the concert score itself may be stored in the database from where it can be retrieved by the concert client.

We assume that all objects in the database can be referenced via qualified names that incorporate the following information:

Concert A unique concert name.

Object A unique object identifier (which may be again a structured identifier corresponding to a directory structure)

Variant A name for the variant of the object, such as the language of a document, the type of native code for a program, or the member of an ensemble that generated an object during a concert.

Version A name for the revision of the object.

An object may then be retrieved by providing this qualified name (where a default may be substituted for the version). Actually, most of the information needs to be specified only once when the concert connection is established (concert name, machine type, language preference, etc.) such that in the actual score typically only the "object" component of the identifier needs to be explicitly provided.

A concrete object-oriented programming interface might then look as follows:

```
class Object
{
  ...
  Int get(String)
  Int put(String)
  ...
}

class MyObject: Object
{
  ...
  Result use(Arguments)
  ...
}

MyObject object = new MyObject();       // create new object handle
code = object.get("identifier");        // bind handle to object in db
result = object.use(arguments);         // access object interface
code = object.put("identifier");        // store object in database
```

Database objects are instances of subclasses of Object that provides methods for loading the object from the database and storing it there while the concrete usage of the object is defined by the method interface of the particular subclass.

4.2.4 Concert Services

For actually implementing classes Concert, Control, and Instrument, ..., a variety of low-level services are required:

- A concert score (running on the player host) needs to contact the concert manager (running on the host where the concert is initiated) to present its credentials and to get linked into the concert.
- An instrument needs to decide whether it is to be activated in the current player ensemble or not,
- An instrument needs to establish who the leader of its ensemble is.
- Facilities for starting and stopping instruments are required.

- Facilities for bringing a instrument into independent mode and back into leading mode have to be provided.
- A leading instrument must broadcast its activities to the follower instruments.
- Beat sequences have to be implemented.

All these facilities are ultimately based on a low-level interface that allows a instrument to get access to several concert services:

```
class Service
{
  ...
  Int bind(String)
  Int call(String, InInterface, InParameter, OutInterface, OutParameter)
  ...
}
...
```

These services are provided in the context of the current concert instrument i.e. a service request is implicitly parameterized by a unique concert identifier, host identifier, player identifier, and process identifier. An instrument may then ask for a service

```
Service service = new Service();
service.bind("instrumentensemble");
MyInstrument leader;
service.call("leader", InVoid, InDummy, OutObject, mymaster);
```

to retrieve the identity of the current leading instrument.

On top of this low-level service request brokering scheme, higher-level abstractions may be provided a few of which are sketched below.

Ensemble Communication Intra-ensemble communication is based on a one-to-many pattern where the leading instrument multicasts its control information to all followers. Communication may be based on several transmission protocols (connection-oriented/connection-less, reliable/unreliable, ...) depending on the demands of the particular instrument. For each of the required protocols, a corresponding low-level service class has to be implemented.

For instance, a slide presentation ensemble might be based on connection-oriented reliable transmission channel for transmitting the signals controlling the presentation (whose content has been retrieved by each instrument independently via the network database). On the other hand, a real-time audio ensemble may be based on a connection-less unreliable multicasting scheme to forward the individual frames.

In any case, it is important that the leading instrument must not be blocked or deferred by communication with any particular follower. In case of connection-oriented reliable transmission schemes this implies the buffering of all generated messages and the use of multiple independent threads for handling the individual communication channels.

The main complication with the communication system arises when the execution mode of an instrument ensemble is changed, in particular by transferring leadership from one instrument to another. To preserve responsiveness, it is not acceptable to wait for all instruments to agree on the leadership change but it must suffice to establish agreement between the old leader and the new one. The protocol must ensure that even without reliable global notification the execution mode can be safely switched (e.g. by letting the old leader forward spurious requests to the new one).

Beat Sequences Beat sequences are an abstraction of a reliable multicasting communication protocol for the transmission of messages whose information contents consist of simple character strings. Their primary role as a special concept is that of a nice interface to the player developing a concert that requires synchronization between different instrument ensembles.

On the lower level they appear as a separate datatype:

```
class BeatSequence
{
  ...
  Int bind(String);      // bind to a global name
  connect(BeatSequence); // connect another sequence to sequence
  signal(String);        // add beat to sequence
  String wait();         // wait for next beat in sequence
  ...
}
```

This interface allows to create a new sequence with a unique global name, to connect a child sequence to a parent sequence such that the child receives all beats from the parent *without* removing the beats, to add a named beat to the tail of the sequence and to remove a beat from the tail of the sequence.

On the higher level, the concepts of beat sequences as separate entities is hidden from the player by letting sequences appear directly attached to instrument ensembles and let each instrument directly access the beats generated by the respective leading instrument:

```
class Instrument
{
  ...
  signal(String);       // add a signal to sequence (leader only)
  String wait();        // wait for next beat in sequence
  ...
}
```

This illusion is created as depicted in Figure 3:

Each instrument is actually connected to its personal local stub of the corresponding ensemble with its own copy of the beat sequence connected to the "master copy". Each wait on a beat of the ensemble therefore removes the beat

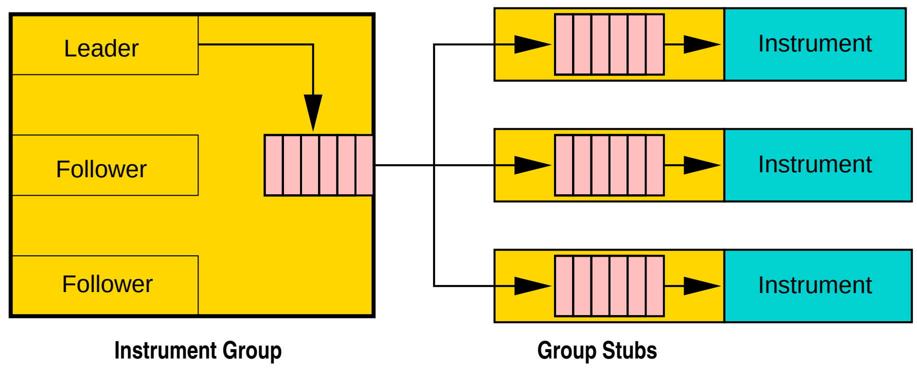

Figure 3: A Beat Sequence

from its local copy only. Actually, in order to reduce network load, each host should probably accommodate a central "proxy copy" of the beat sequence to which the local copies of its various instruments can be connected.

As sketched in the previous subsection, changes in the state of instrument ensembles must be handled appropriately by establishing agreement between old leader and new one only. One possibility for handling the attached beat sequences is to replicate the sequence in each follower instrument (as outlined above, "proxy" copies in each host are advisable anyway) and simply transfer information about all followers to the new leading instrument.

5 Building CONCERT Instruments

The development of components for distributed education sessions has to solve the following 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,
4. Integration of external applications.

The basic problems of the first three items and solution strategies have been described in Subsection 4.2.4. Instruments directly implemented as Java classes of a concert score can rely on the high-level concepts of

- beat sequences for synchronization,
- access to a global database for communication

or utilize low-level protocol classes for special purposes (broadcasting of real-time audio frames from the leading instrument to followers). While the technical details may be demanding (questions of quality of service), the essential concepts are clear. Thus we now concentrate on the last items i.e. the integration of existing application programs as instruments and its relation to the issues listed above.

5.1 Embedding External Applications

The facilities provided by Java, our intended programming and execution environment, for interfacing with external applications are the following:

- Starting a new process running externally to the interpreter (by specifying the command to be executed and its variable environment, communicating with it via the standard input/output/error streams, waiting for process termination and returning its exit value),
- Opening sockets for IP connections to other servers,
- Java IDL (Interface Definition Language) for the use of CORBA-compliant network services,
- Java JNI (Java Native Interface) for calling C/C++ code from Java programs,
- Java RMI (Remote Method Invocation) for interfacing with other Java applications,
- JavaBeans as an architecture to make Java programs distributed components of other applications.
- Java DBC (Database Connectivity) that similar to Microsoft's ODBC interface allows connection to external databases via an SQL interface,

From this side, many possibilities for interfacing with external applications are available. The basic question therefore is by which facilities the external application supports its embedding as a component into a larger framework. Here we can distinguish the following classes of programs:

Closed World This category comprises applications that are essentially designed for interactive use only. Their embedding facilities are restricted to forking a separate process executing the application and communicating with it via

- command line arguments defined at startup time,
- environment variables defined at startup time,
- resource properties provided by the screen server (X11) at startup time,
- input/output files for files for file-based communication,

- standard input/output streams for stream-based communication,
- special hooks for allowing user-denoted applications to perform certain tasks (e.g. by providing the name of some program at the command line),
- the return code provided by the application.

Depending on the kind of "ad hoc" support provided by the application, its embedding into a general framework may range from manageable to hard to impossible.

External Links Into this category we put applications that are mainly designed for interactive use but that also provide facilities such that their use as a component of an external program is facilitated (and vice versa). This includes in particular any provision to define channels for communication with other processes, possibly on other computers.

Such facilities especially designed for external communication and control may not make the embedding into a general framework easy but at least possible (in principle).

Dynamic Linking These are facilities that are explicitly designed to serve as components of other applications by use of dynamic linking techniques that allow to "glue" together components to build larger frameworks. Components adhering to some distributed object standard (CORBA, ...; see Section A.3) typically fall into this category.

Dynamic linking facilities make the use of the individual services provided by an application (almost) as easy as a normal procedure call.

Applications of the "closed world" category are inherently difficult to integrate in distributed sessions; substantial redevelopments may be required. On the other hand, the integration of applications applying distributed object technology like CORBA is (via Java IDL) relatively straight-forward. We thus concentrate our further consideration on applications providing external links for which integration is feasible but for which the development of appropriate interfaces is required. In order to make the presentation more concrete, we sketch in the next section a strategy for embedding Mathematica [Wol96] [Wol97] as a component into our architectural framework.

5.2 Embedding Mathematica

The use of Mathematica in mathematical education sessions is twofold see Section 3:

1. Using the notebook facility to display mathematical hypertext documents on screen ("presenter"),
2. Using the interpreter to run program demonstrations and support interactive exercises ("executor").

However, since Mathematica notebooks may embed executable code, ultimately these issues cannot be separated.

Mathematica's MathLink facility allows to establish communication channels to external processes (also on remote hosts) on the basis of the IP protocol. By two-way communication, a Mathematica program thus may request services from external programs and vice versa. Communication routines on the Mathematica side take care for the proper serialization of Mathematica objects into byte streams that can be transmitted via channels; corresponding facilities are available for C/C++ applications using an external library.

The Mathematica kernel itself uses MathLink to communicate with Mathematica front ends. If the Mathematica kernel is started from within a front end, the kernel will be thus controlled through a MathLink connection to this front end. It is even possible to change in the middle of a Mathematica session the front end to which the kernel is connected by resetting the corresponding communication link.

Normally, the standard notebook frontend is used in Mathematica presentations. *Notebooks* (Mathematica hypertext documents) are composed of individual *cells* that may contain text, images, executable Mathematica code, etc. A session typically consists in the manipulation of the notebook navigating between the various cells and running embedded Mathematica programs. However, cells and notebooks are also Mathematica objects that can be manipulated by Mathematica programs themselves. Available operations include

- modifying cell style and display options,
- reading and modifying the contents of the currently selected cell,
- evaluating the contents of a cell using the Mathematica kernel,
- opening and closing the subcells of a cell,
- moving the current selection from one cell to another within the notebook structure

To use Mathematica as a concert instrument in leader mode, it suffices to pursue the strategy outlined in Figure 4:

We place between the Mathematica notebook frontend and the Mathematica kernel a control interface as a separate process which communicates with both processes via MathLink presenting itself as the kernel to the frontend and as the frontend to the kernel. Indeed when executed in asynchronous mode the task of the interface is just to forward the messages received in both directions.

However, when executed in leader mode the interfaces of the followers block all messages received from their frontends; instead they forward to their kernels the messages from the interface of the leader i.e. ultimately from the leader frontend. The outputs of the Mathematica kernels on all hosts are forwarded to their respective frontends.

The net effect is that all Mathematica commands generated by the leader are executed and the resulting output are displayed by the followers. Since this

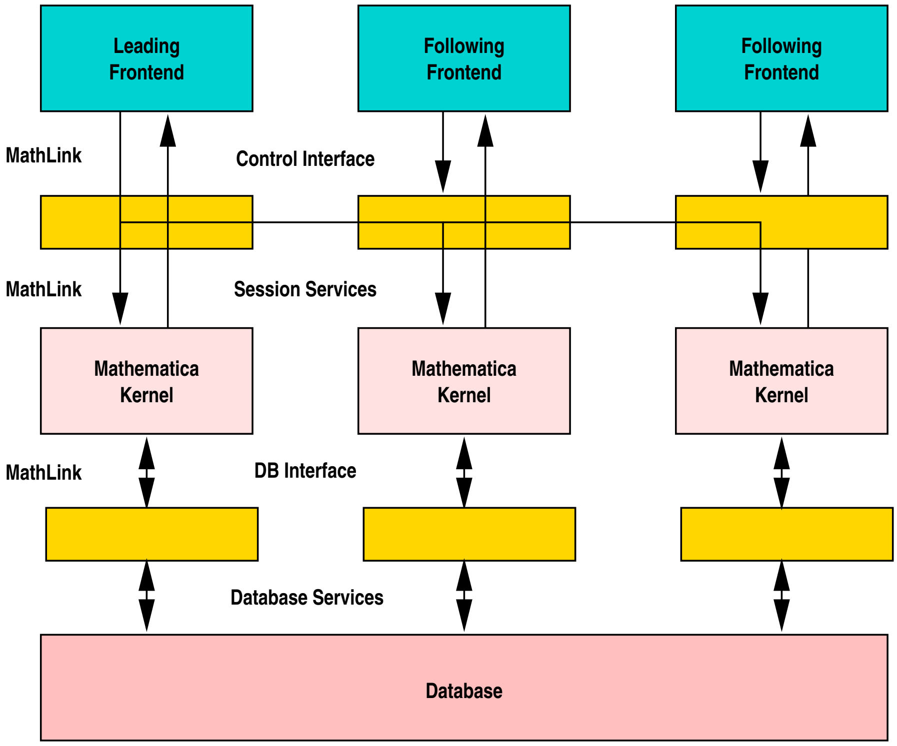

Figure 4: A Mathematica Ensemble

includes all notebook manipulations, distributed Mathematica presentations become possible. Neither frontend nor kernel require any changes, only the interface processes have to be provided and the appropriate connections have to be established.

However, in case the Mathematica notebook session involves input or output from respectively to files appropriate measures have to be taken to redirect those to the network database. Thus an interface has to be established such that Mathematica programs can read and update files stored in the database. Such an interface can be implemented by an external program with the help of MathLink such that opening a "network file" results in a local file copy that can be read and updated in the conventional way and closing a modified network file results stores the result in the database under a name incorporating the identifier of the component instance as a variant (in order to keep the files from different users separate). The lecturer thus can retrieve the results of her students' exercises.

5.3 Mathematica as a CONCERT Instrument

Summarizing the discussion of the previous session, integrating Mathematica as a concert instrument involves the following tasks:

Control Interface A control interface program has to be implemented that handles the execution modes of the instrument ensemble. This program is based on low-level concert services (see Subsection 4.2.4) and can be implemented as a Java application. The low-level communication services of the MathLink libraries have to be integrated via the Java native interface (JNI).

DB Interface A database interface program has to be implemented that handles network database access for Mathematica programs. This program makes use of the architecture's database services (see Subsection 4.2.3) and can be implemented as a Java application integrating via JNI the MathLink facilities for communication with the Mathematica kernel.

Furthermore a corresponding Mathematica library has to be provided that handles the datatype "network file" with the corresponding operations by communication with the database interface program.

Java Class Finally, a Java class has to be provided that allows the use of Mathematica as an instrument within the concert score. This class serves as the interface to the Mathematica frontend, kernel, control interface and database interface that run as independent processes. The Java component class invokes these external programs and establishes the appropriate communication links (by denoting the machine ports through which the three MathLink layers shall communicate).

Ultimately the central part of the control interface (the only part interacting with the instrument running on other machines) may be taken by the interface class itself.

The consistent organization of the locations where the native code involved in such a Mathematica session (Mathematica frontend, Mathematica kernel, MathLink libraries) may become a difficult problem in distributed heterogeneous environments. One possibility is to let the administrator of the corresponding host take care of this task (by placing the code in standard locations or denoting the locations by environment variables).

Another possibility is to let the Java interface class load the (appropriate variant of the code) from the global database. While technically feasible and organizationally advisable, this in general imposes problems with license contracts of commercial software packages (such as Mathematica). Consequently both solutions should be supported.

6 Developing CONCERT Tools

The actual development and performance of concerts requires various supporting software tools sketched in the following subsections.

6.1 Composing Concert Scores

Composing a concert score by constructing a Java session program as depicted in Subsections 4.2.1 and 4.2.2 is conceptually simple but still a task too low-level for typical users. Various coordination languages, based on high-level declarative or visual abstraction models, can be used to construct sessions in a more comfortable way and generate the corresponding Java classes.

As an example, Figure 5 sketches the essentials of a possible visual programming interface; this picture contains the same same information as the concert score described in Subsection 4.2.1):

- Each vertical bar represents an instrument ensemble with arrows describing their interaction with the concert score (leftmost bar):
- Green (thick) arrows from the score denote the activation of the corresponding ensemble; annotations describe the argument strings passed to each activated ensemble.
- Blue (dashed) arrows describe beat synchronization between score and ensemble causing the score to be (potentially) blocked.
- Black (thin) arrows describe the activation of general instrument methods such as hiding an instrument, showing it again, bringing it into independent execution mode and back into follower mode again.
- Those times when ensembles are hidden are denoted by dashed vertical bars; those times when ensembles are operating in independent mode are denoted by thick vertical bars.
- A red arrow to the score finally signals the termination of each ensemble.

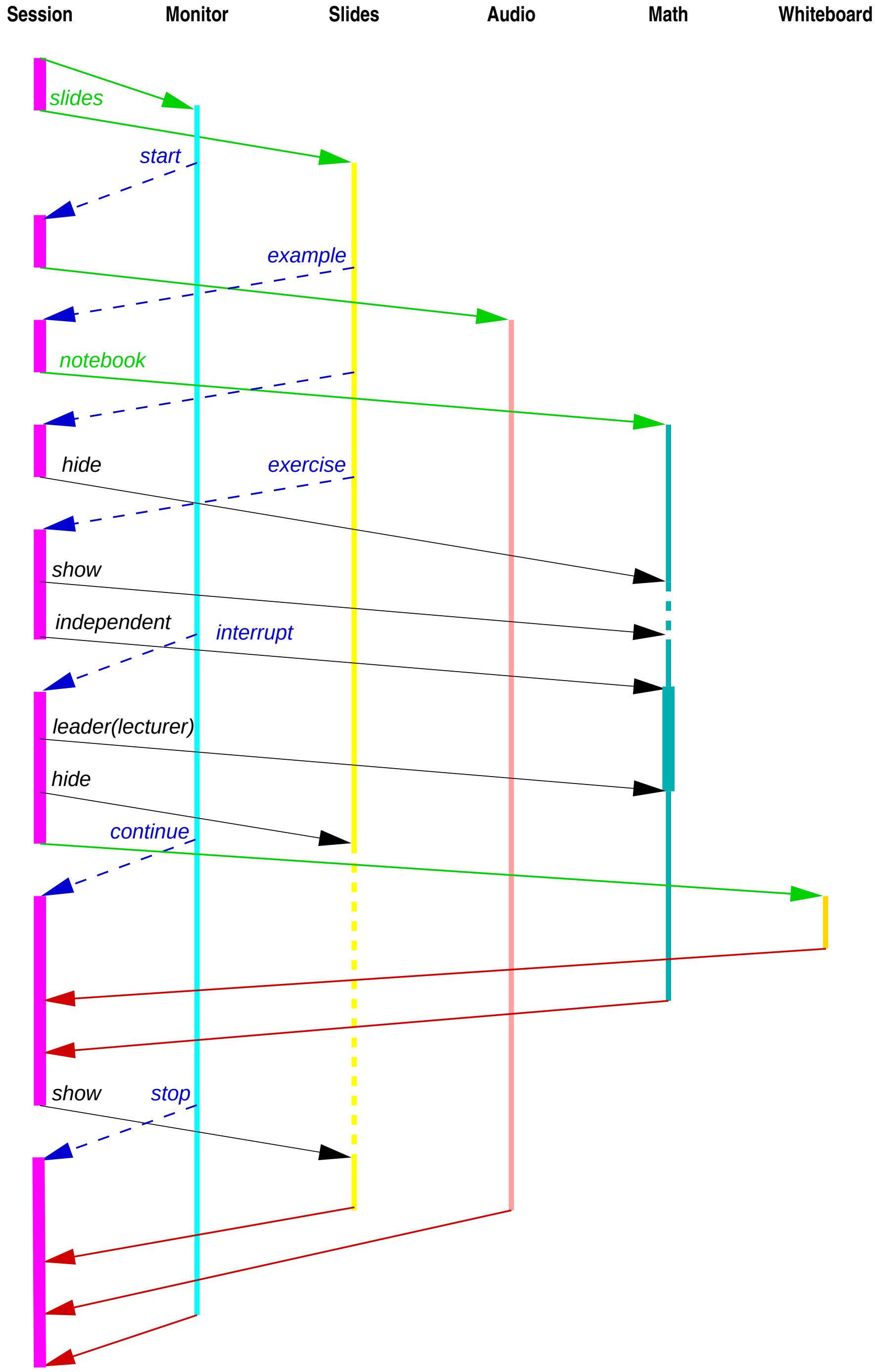

Figure 5: Visual Concert Composition

While the details of graphical representations may vary, it should be clear that the concert model described in this paper can be visually described in a unique way such that the corresponding Java classes are generated by a compiler. The only remaining task is the association of instrument ensembles to player ensembles and the definition of access policies, which can be also supported by graphical standard forms.

6.2 Concert Simulation and Debugging

An important part of the development environment is "offline simulation" i.e. testing and debugging a concert on a single computer without actually involving a cluster of networked computers. This involves the verification of the correctness of

- individual instrument ensembles,
- interaction between ensembles,
- mapping of instrument ensembles to player ensembles,
- access permissions to player ensembles.

Fortunately most of these issues can be tested by simply running a concert on a single host computer. The lecturer connects herself to the concert multiple times for each individual player group displaying all the components on a single screen. By running the intended lecturer actions, the effects on the various users can be observed; by switching instrument ensembles to independent mode allows to run the interactions of the audience with the concert.

However for detecting subtle problems in the flow of the concert, the availability of a "concert debugger" is advisable that allows step-tracing the concert and visualizing otherwise hidden activities (such as the state of beat sequences). Based on corresponding facilities in Java programming environments such a debugging/monitoring/simulation utility can be developed and integrated with the visual programming interface.

6.3 Concert Manager

The central point for concert organization is a *concert manager* where the lecturer can register scores for subsequent access from other hosts providing

- the score itself (a set of Java byte code files),
- access restrictions for each player group (based on network domains, host names, passwords).

The subsequent task of the concert manager is comparatively simple: it listens on some machine port for concert requests, lists the available scores (perhaps plus ensemble-specific information such as the instruments involved and the number of players connected), transmits the requested score, receives and checks the credential for access to the requested player group, and connects the player to the concert.

In addition, the concert manager must provide facilities for maintaining (updating and deleting) scores under protection of a score password that the lecturer defines when registering the score.

6.4 Concert Client

For contacting the concert manager from a client host, the *concert client* is required. This is a small Java startup program that can be published via the WWW and that implements the client side of the manager protocol sketched in the previous subsection. Its essential task is to display the score information provided by some concert manager, transmit the credentials (password) to the server, receive the corresponding score and start its execution by invoking an JVM interpreter as an external process. The task of the concert client is then fulfilled and it may terminate.

7 Implementing CONCERT

In the previous sections we've used Java and related technologies as the basis of several design decisions and implementation sketches. We are now going to elaborate the technological basis of the proposed architecture in more detail.

7.1 Technological Assessments

Appendix A contains an analysis of the main technologies, systems and languages that are relevant to building geographically distributed multi-user applications and that are projected to play a major role in the mid-term future. In a nutshell, this analysis leads us to the following assessments:

Coordination Languages While a large number of coordination languages have been proposed and developed, there is no empirical evidence for the a priori superiority of any particular model. We consequently believe that the coordination model should be based on the demands of the application domain.

WWW/Java *Java* is de facto *the* Internet programming language with a large number of vendors providing technologies and compatible systems. It is extremely portable and available on practically every existing system; JNI provides an interface to external C/C++ code. Despite of its known problems (efficiency, security versus flexibility), it would be foolish not to utilize the existing technology base wherever possible. *ActiveX*, the main competing technology, will also play an important role but its future outside the Microsoft world is insecure.

Information Services The WWW conceptually provides the worlds largest network database but without the structuring and management facilities of a real databases. *HyperWave* is based on a much better organization concept for providing distributed information services; its multi-protocol support allows to access most of its features also from conventional WWW browsers. Its commercial support and its fairly large user base ensure future reliability and availability.

Distributed Objects *CORBA/OpenDoc* are currently the industry standards for building distributed object-based services. *Java RMI* and *JavaBeans* provide similar services for Java-based environments; *Java IDL* represents an interface to the OMG standards. The Microsoft technologies *DCOM/ActiveX* play a similar role for Windows/Intel systems but their future support as an "open standard" by other vendors is uncertain.

Network Services The Argonne *Nexus* system is a stable basis for building distributed services. However, its primary purpose is to serve as a runtime system for parallel languages and communication libraries; correspondingly its interface provides rather low-level concepts like threads, global pointers, and remote service requests. On the other hand, the Caltech *Infospheres Infrastructure* has been primarily designed to build distributed applications on the basis of the Internet infrastructure leveraging open standards in the WWW/Java world. The system is still in its beta stage but the prospects seem promising. The European *PageSpace* effort pursues similar goals but its reliance on tuplespaces as the basic communication mechanism may result in efficiency problems. Technologies for mobile agent transfer like the IBM *Aglets Workbench* might be integrated into II or PageSpace.

Collaborative Environments The commercial market is still mainly focused on groupware supporting the needs of centrally located medium-sized enterprises. There exist already various research prototypes for distributed virtual reality, concurrent engineering, and virtual classrooms (the later mostly limited to video conferencing), but few systems are usable for real-world applications and most of them are based on proprietary technologies.

Based on this analysis, we sketch in the next subsection a possible implementation strategy for our architectural framework.

7.2 Implementation Roadmap

The proposed design relies on the fact that every concert player downloads and executes a program that starts the instruments relevant for the particular player and coordinates them among each other and with the instruments of other players (in particular of the lecturer). This coordination is based on a set of *concert services* that must be provided by an underlying runtime system and of which *beat sequences* are a high-level user abstraction. Furthermore, the components retrieve information from a global database filled during the preparation of the concert and may also put created objects there.

We intend to implement the proposed architecture i.e.

- an application framework for concerts,
- various concert instruments, in particular a Mathematica instrument,
- supporting tools, in particular a visual programming environment, concert manager, and concert client,

in a Java-based environment i.e. using the Java language and systems and related/compatible systems and technologies:

Concert Services and Ensemble Communication We plan to utilize the CalTech Information Infrastructure (II) for implementing the application framework and providing the distributed services on top of which the higher-level abstractions can be provided. The II can be utilized for

- distributed communication based on the "mailbox" facility (II package info.net): signal streams and general message passing abstractions can be based on this abstraction.
- distributed services based on the "Djinn" abstraction (II package info.djinn): any service can be encapsulated as a Djinn (an encapsulation of a set of threads manipulating an internal state and using mailboxes for external communication).

When a downloaded concert score starts execution it invokes the so called "Djinn Master" that is in charge of forwarding service requests to Djinns, invoking the corresponding Djinn if necessary.

Since the II system still is in a beta-stage, it might however turn out that the basis is not yet stable enough for own developments (although we do not expect this, because the system is continuously developed and improved). In this case we can utilize the proved Nexus runtime system and to build the required distributed services and to use the JNI interface for integration into Java. Experiences of the Globus project (building a large-scale distributed infrastructure) on the basis of Nexus may aid such an approach.

As an supplement respectively alternative to the II mailbox abstraction for inter-process investigation we will also investigate the Java RMI (remote method interface) for direct invocation of Djinn methods. We expect this to provide a for many kinds of services a more suitable interface than the II message passing abstraction. Furthermore, for real-time multicasting services (i.e. audio and video) communication between Djinns has to be based on special protocols like the Internet-based MBONE; corresponding Java interfaces have to be developed.

Concert Instruments and Application Interfaces As sketched in Section 5.2 we will use the JNI (Java Native Interface) for integrating Djinns with services that are only available in other languages (such as the MathLink communication library). For other application components, we expect the JNI interface to play an important role as well.

For case that CORBA-compliant services/components are to be utilized, the Java IDL interface is available. However, CORBA is basically more suitable for client/server applications than for general peer-to-peer frameworks; thus the necessity respectively suitability of its application in the proposed framework is questionable.

We do not yet see any purpose in utilizing distributed component technology (JavaBeans) for building the proposed components; their interfaces are closely bound to the architectural framework proposed in this paper, which makes their use in other application contexts unlikely.

Network Database We intend to use the HyperWave hypermedia information system as the basis of the proposed global database services. HyperWaves structuring and indexing facilities, its multi-protocol facilities and its availability on many platforms with commercial support make it an ideal platform for out framework.

We have to develop appropriate Djinn servers mapping the abstract network database services into concrete HyperWave requests, a task that can be performed with Java's networking facilities.

Supporting Tools The visual programming interface sketched in this paper can be implemented as a Java application using the AWT (Abstract Windowing Toolkit) either as a stand-alone application or as a WWW applet. The technical facilities are sufficient; the main challenge lies in finding the right visual abstractions and in generating the appropriate Java classes. On the same software basis, the concert manager and client can be implemented. The development of a visual debugging environment is technically more challenging since it requires close interaction with the Java VM and a Java breakpoint debugger.

While various technical details remain to be settled during the concrete development, the overall roadmap to an implementation of the proposed architecture based on well-supported technologies is clear.

8 Conclusion

The CONCERT architecture is based on an SPMD style of coordination where essentially a single thread of control directs the invocation and interaction of multiple processes in a distributed environment. Processes are organized in the form of "ensembles" that may either operate in leader/follower or in independent mode; different ensembles interact with each other using "beat sequences" as their basic synchronization mechanism and using a global database for communication. This form of abstraction allows also non-computer science experts to compose education sessions in a high-level language.

For the coordination of processes within an ensemble, various lower-level communication protocols and system services may be applied. This gives the necessary flexibility to hide different kinds of services with different interaction demands behind an interface that allows easy integration into the overall framework. Only the implementor of a new CONCERT instrument needs to be aware of these lower layers, the lecturer coordinating their use can just rely on the standard interface. As an example, we have described how Mathematica can be integrated for use in distributed mathematical education using the system for the presentation of hypermedia documents and for the demonstration respectively training of algorithms.

The presented implementation sketch relies on Java and compatible systems such as the CalTech Infosphere Infrastructure for providing network services and the HyperWave hypermedia system for offering distributed database services. Based on these technologies as building blocks, an implementation of the architecture with realistic efforts seems feasible in the mid-term future,

A Technologies for Distributed Systems

In the following, we summarize the main technologies, systems and languages relevant to building geographically distributed multi-user applications. The presented list is by no means complete; we intentionally focus on those activities that we consider to represent the main branches in this area. Our goal is to make the picture *clear*, not complete.

Our assessment is structured as follows:

Coordination Languages for coordinating concurrent activities,

WWW-Related Technologies as the basis of a future "Global Information Infrastructure".

Distributed Objects and Components i.e. technologies for providing network-transparent object interfaces.

Network Computing Systems that let computer clusters appear as "metacomputers" to applications,

Collaborative Environments i.e. examples of distributed systems that support cooperation towards a common goal.

A.1 Coordination Languages

Coordination languages express coordination among entities which have to cooperate or to synchronize to perform some set of tasks or achieve some common goal. This notion was elaborated in the context of the programming model Linda [GC89] [GC92] which divided a parallel programming language into two (ideally) orthogonal components: the *computation model* and the *coordination model* where the later is in charge for the creation and interaction of concurrent activities.

Linda coordinates parallel processes via a common tuple store and has been implemented for a large set of language bindings. Consequently various coordination languages based on logic [FT89b],[FT94] functional [Col89] [D+93] [Sch96b] and object-oriented models [CCR96] have been developed. Also the Gamma model based on the simulation of chemical reactions has been extensively studied for this purpose [AHLM96]. [Cia96a] provides an overview over coordination models and languages; [Cia96b] contains bibliography on this topic. [Adl95] investigates the more technical oriented site of coordination models for client/server applications.

Unfortunately there is no empirical evidence that any of the coordination languages proposed in the past has a clear advantage over the other ones. Most actual distributed systems therefore concentrate on the underlying architectural, interfacing, portability and compatibility issues (see the next sections) rather than on high-level coordination models.

A.2 WWW-Related Technologies

The World Wide Web (WWW) was designed to provide a minimal technology for the delivery of multimedia hypertext documents across the Internet. Its architecture is very simple consisting of

- a communication protocol (HTTP — HyperText Transfer Protocol),
- a hypertext document description language (HTML — HyperText Markup Language), and
- a global addressing scheme for document resources (URL — Uniform Resource Locators).

Starting from this simple basis, the WWW is starting to evolve into an environment supporting distributed multiuser applications which involve communication, cooperation, and coordination [CTV96a].

A.2.1 Java

The SunSoft development Java [Jav96a] is a portable execution environment based on a C++ like programming language [Fla96] which compiles to code for an abstract Java Virtual Machine (JVM). JVM programs ("applets") are served via WWW servers and may be referenced from WWW pages. On the client side, a WWW browser viewing a Java enhanced page loads the referenced applets from the server and executes them by invoking a JVM interpreter.

Java applets may be used to implement "live" documents but also user interfaces and interactive applications. Security restrictions however prevent the direct interaction between applets executed by different clients (network connections are only allowed to the server); standalone Java applications are not subject to such constraints. Java technology is nowadays generally utilized as the basis of many distributed computing projects because of its portability and its integrated support by various WWW browsers.

Java is currently *the* Internet programming language supported by many vendors and products. During the next years it will certainly be the basis of many further developments.

A.2.2 ActiveX

ActiveX is Microsoft's approach to provide executable contents in WWW pages and to let WWW browsers interact with Windows applications. Similar to Java applets, ActiveX "controls" are loaded from a server and executed on the client machine using COM for communication between objects and applications. Unlike to Java applets, ActiveX controls are proprietary; they consist of compiled machine code and run on Windows/Intel platforms only.

ActiveX has been promoted by Microsoft as a response to the challenge presented by the Java world and its envisioned role in the Internet/WWW. Recently Microsoft has presented ActiveX as an open standard by handing control over to the ActiveX group (a suborganization of the Open Group); still the technology is closely related to the Windows/Intel world.

A.2.3 VRML

Another branch of activities is centered around the recently standardized VRML (Virtual Reality Modeling Language) [VRM96] which allows to build animated 3D scenarios on top of the WWW infrastructure. While the current VRML 2.0 standard provide each client its own copy of a virtual world, extensions to shared worlds with interactions between multiple users (e.g. based on Java technology) have been proposed [BE95] [And96] [Roe95b].

A.2.4 HyperWave

The World-Wide-Web essentially represents a set of hypertext pages that are stored on various servers and reference each other by embedded links that contain absolute addresses of the machines where the information is stored. This lack of coordination makes the flexible management of large-scale information bases difficult. Various facilities for managing WWW servers are available but suffer from the deficiencies of the basic concept.

A system better suitable for such purposes is HyperWave (former Hyper-G), a large-scale, multi-protocol, distributed hypermedia information system which uses an object-oriented document and hyperlink database to provide information structuring, link maintenance, and content search facilities [Mau96] [AM95] [Hyp96]. The system consequently allows the network-transparent relocation of hypertext graphs, a task which is very difficult in the WWW. Still HyperWave leverages WWW technologies; by its multi-protocol support it can serve information to existing WWW clients as well as to special HyperWave clients. The system was developed at the Technical University Graz and is now commercially distributed.

A.3 Distributed Objects and Components

The term *middleware* denotes a set of general-purpose services that sits between platforms (various types of hardware and operating systems) and applications [Ber96]. A standardized interface allows applications to request services over the network without knowing how or even where the service is implemented. Middleware therefore facilitates the design of distributed systems whose configuration may dynamically change and where multiple applications implemented in different languages and running on different systems communicate with each other.

Modern middleware is based on the model of *distributed objects* [OaE96]. Services are encapsulated as objects with well-defined interfaces; middleware allows the invocation of methods from objects located on other nodes (remote requests). Typically the method caller (the client) is blocked until the remote object method has terminated and delivered its answer (remote procedure call, synchronous message passing). Thus client/server applications can be easily implemented in a network-transparent fashion.

Distributed object technology can be utilized to break complex applications into small *components*. Each component is in charge of a specific task and may run on a different machines; still all components may be seamlessly integrated in a common application. E.g. a document may be compound of multi-media objects of different types (potentially stored in different locations); when one of the components is processed, the corresponding methods are activated (*document-centered processing*).

A.3.1 CORBA and OpenDoc

CORBA (Common Object Request Broker Architecture) [Gro95a] is an emerging open distributed object computing infrastructure being standardized by the OMG (Object Management Group). CORBA automates many common network programming tasks such as object registration, location, and activation, request demultiplexing, framing and error-handling, parameter marshalling and demarshalling, and operation dispatching.

CORBA thus provides the mechanisms by which objects transparently make requests and receive responses, as defined by OMG's ORB (Object Request Broker). The CORBA ORB is an application framework that provides interoperability between objects, built in (possibly) different languages, running on (possibly) different machines in heterogeneous distributed environments. It is the cornerstone of OMG's Object Management Architecture.

OpenDoc [Ope96] is the distributed component builder standard chosen by the OMG on the basis of a multi-vendor specification based on IBM's SOM (System Object Model). This standard provides the specifications for creating distributed component software that can be shared and customized across heterogeneous environments. While having its origin in a compound document architecture, OpenDoc has evolved into a general distributed software builder.

CORBA/OpenDoc are currently *the* industry standards in distributed object technology; however they are based on a client/server model of computation which makes it more difficult to build peer-to-peer applications [Gro96].

A.3.2 COM, DCOM, and OLE

In contrast to the OMG standards, Microsoft pursues its proprietary COM (Component Object Model) [COM96] as the fundamental object technology for Windows 95 and Windows NT. COM provides the low-level object-binding mechanism that enables objects to communicate with each other; it is the basis of OLE (Object Linking and Embedding) which allows to build compound documents whose components (COM objects) stem from different applications. COM is platform-specific (Intel-based Windows systems) and only works within a machine. For communication with objects on other machines, a distributed variant DCOM [DCO96] (former "Network OLE") is required. COM is also the basis of ActiveX (see the next subsection); the further development of COM/DCOM has been transferred to the ActiveX group, a subdivision of the Open Group.

The Microsoft approach to middleware is characterized by the promotion of proprietary technologies that are tightly bound to Windows/Intel systems and thus little attractive for vendors outside the "Microsoft world". Still by its market dominance, the Microsoft technologies will certainly play an important role in the future.

A.3.3 Java JNI, RMI, IDL, and JavaBeans

The JNI (Java Native Interface) allows to call external C/C++ code from Java programs and vice versa; corresponding tools are available that generate interface definitions for the respective languages.

SunSoft has developed Java RMI (Remote Methods Invocation), a distributed object architecture to provide interfaces between remote Java applications. Java IDL (Interface Definition language) is supported as an interface between Java and CORBA.

A corresponding distributed component architecture, JavaBeans [Jav96b] was developed that allows to write reusable and platform-independent components for all systems supporting the Java runtime system. JavaBeans provides both interfaces to OpenDoc and to COM.

The Java approach to distributed object middleware puts much focus on portability and platform-independence as well as interfacing to existing standards. This ensures its continued support by many vendors and its future relevance.

A.4 Network Computing Systems

Network computing systems (network runtime systems, network operating systems, metacomputing systems) attempt to provide the basic technology for utilizing computer networks for a common application. Among their basic functionality, they provide facilities for creating, locating, and communicating to processes in a network environment. In the following, we only consider systems intended for use in wide-area scales.

A.4.1 Nexus

The Argonne National Laboratory Nexus is a portable library providing the multithreading, communication, and resource management facilities required to implement advanced languages, libraries, and applications in heterogeneous parallel and distributed computing environments [FKT94] [FKT97] [FKPT96] [Nex97]. Its interface provides multiple threads of control, dynamic processor acquisition, dynamic address space creation, a global memory model via interprocessor references, and asynchronous events. Its implementation supports multiple communication protocols and resource characterization mechanisms that allow automatic selection of optimal protocols. Fault tolerance or security issues are not yet addressed.

Nexus is intended for use by compiler writers and library developers, rather than application programmers. Several parallel languages and communication libraries have been implemented on top of Nexus. The 1995 Information Wide Area Year (I-WAY) experiment for high-performance distributed computing was based on the Nexus system [FGN+95] [IWA95]. The Argonne and CalTech joint Globus Project builds on Nexus mechanisms to develop a basic software infrastructure for computations that integrate geographically distributed computational and information resources [FK96] [Glo96].

Nexus has been developed over a period of few years and provides a stable implementation with a number of users, applications, and projects relying on it. Its focus however is more on providing "networked virtual supercomputers" than on distributed multi-user systems and thus lacks a certain integration with WWW-technologies.

A.4.2 Infospheres Infrastructure

The Caltech Infospheres Infrastructure (II) [Inf97] is a distributed system framework implemented in Java. It provides a generic object model and a variety of messaging models: asynchronous, synchronous, and remote method calls. The system (currently a beta release [Gro96]) is provided free-of-charge so that researchers and developers can develop lightweight distributed systems that can leverage open standards and the World Wide Web [CRS+96] [CS96] [CR97]. The II is being extended to support interoperability with other distributed system models and to utilize emerging standards in the Java domain.

The system is centered around the concept of "Djinns": each Djinn essentially consists of a set of mailboxes divided into inboxes and outboxes; the outboxes of a Djinn may be connected to an arbitrary number of inboxes of other Djinns and vice versa. Within a Djinn, one or more threads may execute by removing messages from inboxes, changing the internal state, and adding messages to outboxes. Djinns can be persistent and migrate and also encapsulate a variety of services (e.g. CORBA objects or databases); their main purpose is to provide well-defined interfaces for the composition into distributed systems.

A Djinn Master on every node is responsible for the instantiation of persistent Djinn; if a message is invoked on a non-activated Djinn, it finds the appropriate Djinn and brings it into live. A Djinn Generator allows the visual construction of Djinns by tables representing the Djinn states and Java code that describes the Djinn's action in each state. A Djinn Initiator allows the visual construction of sessions i.e. graphs whose nodes represent Djinns and whose edges represent the communication channels between Djinns. On the initiation of such a session, communication with the Djinn Masters takes place to bring the various Djinns into existence. Also mechanisms for providing meta-information about Djinns and querying for Djinns with certain properties are provided.

The Infospheres Infrastructure has been under development for a year or so with no complete documentation yet and hardly more than a few test applications available. However, its convincing design, its well-supported technological basis (Java) and its resulting portability make it certainly an attractive vehicle for future developments.

A.4.3 PageSpace

The ESPRIT funded project PageSpace aims at supporting networked applications which require interaction between distributed software components and active processing [CTV+96b] [Pag97]. It is based on Java-implemented agents embedded into Web pages that concurrently execute and interact with each other in a world-wide distributed environment.

For coordination between agents, various coordination styles are used: One is based on concept introduced by the programming model Linda [GC89] [Lin96] with agents communicating via a set of shared tuple spaces. The implementation of this concept is based on the system Jada, an implementation of the Linda as a Java class library [Jad96].

On top of this basic coordination model, it is intended to introduce two additional flavors of coordination styles: The first one is a rule-oriented style that allows declarative rules on coordination based on ideas of the system ShaDe [CCR96], a coordination language with spaces of objects whose state is transformed in form of rewriting rules. The second one is a service-oriented coordination style based on ideas of the coordination language Laura [Tol96] [Lau93] that manages a service space containg forms describing service offers, requests, and results. For this purpose, both ShaDe and Laura are going to be reimplemented in Java.

PageSpace is in some aspects similar to the Infospheres project, in particular in its reliance on Java and WWW-based technologies. Its characteristic distinction is however the use of Linda-like tuplespaces as the core communication mechanism and the vision of putting higher-level coordination models on top of it. The Linda approach however is today outside the mainstream of since efficient distributed implementation of associative matching mechanisms are difficult. Since the project is still in a rather preliminary state, the success of this approach is uncertain.

A.4.4 Aglets Workbench

The Internet is also envisioned as a medium for mobile network agents i.e. programs that are programs that can be dispatched from one computer and transported to a remote computer for execution. Arriving at the remote computer, they present their credentials and obtain access to local services and data. The remote computer may also serve as a broker by bringing together agents with similar interests and compatible goals, thus providing a meeting place at which agents can interact.

IBM has developed the Aglets Workbench [CL96] [Agl97], a visual environment for building mobile agent applications in Java. The workbench consists of a framework for developing agile agents (aglets), an Agent Transfer Protocol (ATP) for their transfer, libraries for the support of typical communication/transportation patterns, a server for aglet management, and a launcher for dispatching Aglets from a WWW page.

Based on the Aglets Workbench and ATP, IBM has submitted a Mobile Agent Facility specification to the OMG as an attempt to create a standard for transferring agents between computers on the Internet. In addition to the mobile-agent specific components, it also contains data access components (JoDax and JDBC) and a visual builder component (Tazza) supporting the user in building mobile agent applications complete with data access back ends and visual front ends. The alpha version of the Aglets Workbench is freely available for Windows 95/NT systems (only).

A.4.5 Legion

The Legion Project [GWtLT97] [GW96] [Leg] at the University of Virginia aims at the development of a metasystem software providing a single coherent virtual machine that addresses issues as scalability, programming ease, fault tolerance, security, and site-autonomy. Users will have the illusion of a single desktop computer through which they can manipulate objects; Legion is responsible for supporting the abstraction presented to the user, transparently scheduling application components on processors and managing data migration, caching, and transfer. As a first prototype, in 1995 the Campus Wide Virtual Computer (CWVC) was released. Writing parallel applications is supported by a CWVC-aware variant of PVM, parallel C++, and Fortran wrappers. A release of the system for public use is expected in mid-1997.

Legion is an ambitious middleware project trying to address many different complex issues that arise in the design of distributed systems. The goals is to provide a integrated conceptual foundation on which to build applications. The system is not yet a reality; the current prototype software is not designed to evolve directly into a complete Legion implementation.

A.5 Collaborative Environments

Commercially available *groupware* products (such as Lotus Notes or DEC LinkWorks) provide a standard suite of services for group interaction in corporate, medium-sized, enterprise domains (conferencing, electronic mail, multimedia document management, scheduling, workflow). They do not deal with the problems that arise when *CSCW (Computer Supported Cooperative Work)* is extended to widely distributed and dynamically changing environments. The following classification lists approaches for various application domains that try to address exactly such issues.

A.5.1 Distributed Virtual Reality

Distributed Virtual Reality (DVR) systems aim at real-time simulations of 3D worlds in which multiple users can interact with each other [Roe95a] [Loe95]. The central problem of such systems is how to minimize communication between sites while presenting each participant a consistent view of the virtual environment. Because of the bandwidth constraints of wide-area-network connections [BMZ95], most systems rely on multicasting schemes where each host sends the state changes it caused to the affected group of other hosts. Locking schemes are used to guarantee a minimum amount of consistency.

NPSNET of the American Naval Postgraduate School is a test-bed for research and development on distributed VR systems with focus on military battlefield simulation [MZP+94] [NPS95]. It is based on the DIS (Distributed Interactive Simulation) protocol which extrapolates the position ofs objects from its last known position and state. The current version of the software is freely available but utilizes proprietary SGI hardware and software features and is therefore not portable to other systems.

DIVE (Distributed Interactive Virtual Environment) is a toolkit developed by the Swedish Institute for Computer Science (SICS) for building distributed VR applications [CH95] [DIV95a]. Every host of the simulation network has a complete copy of the world database; updates are broadcasted to all other hosts. Binaries for various platforms (but not the source code) are freely available.

AVIARY is a generic world model for distributed VR systems [WHH+93] [Sno95]. AVIARY uses an object-oriented framework where a virtual world is represented as a collection of communicating and concurrently executing autonomous objects. A distributed Environment Database (EDB) manages the spatial extents of objects and notifies them when collisions are detected; it is up to the objects to resolve the collision. A prototype implementation of the AVIARY model has been produced on a local-area network; issues of scalability and non-local distribution are not addressed.

MASSIVE (Model, Architecture and System for Spatial Interaction in Virtual Environments) is a VR tele-conferencing system [GB95] [Gre95]. It integrates real-time video and audio into a virtual 3D environment consisting of multiple worlds with portals to move between worlds. The system works with up to about 10 users and is not a general-purpose VR application development environment.

CVE (CRG Virtual Environment) is a joint effort by the developers of AVIARY and MASSIVE on the development of a new VR platform to support Collaborative Virtual Environments (CVEs) [BGL77] [CVE97]. Multiple users can communicate via a combination of 3D graphics, real-time packet audio and text; the system provides an extensible object-oriented developers API. A first developers version is expected for spring 1997.

PARADISE (Performance Architecture for Advanced Distributed Interactive Simulation Environments) of the Stanford Distributed Systems Group aims to build a large-scale networked simulation environment over a wide-area network [Gro95b]. The current Version 0.1 of the PARADISE software incorporates an Area of Interest directory service for distributing information about entities to the participating hosts. Ongoing work pursues real-time distributed agreement protocols (for distributed collision detection and resolution).

A.5.2 Concurrent Engineering

There exist various approaches to supporting *distributed design and manufacturing* tasks:

MADE (Manufacturing Automation and Design Engineering) [MAD96] is an ARPA program supporting research in the development and demonstration of next-generation design environments, specifically for electromechanical systems.

PACT (Palo Alto Collaborative Testbed) [CEF+93] was designed as a testbed for cooperative research and knowledge sharing in concurrent engineering [CEF+93]. PACT was based on a set of agents sharing and exchanging knowledge using a Knowledge Query and Manipulation Language and a standard Knowledge Interchange Format.

SHADE (Shared Dependency Engineering) [KMW+94] is an extension of PACT providing a medium in the context of the Internet that allows designers to accumulate and share engineering knowledge in a distributed environment.

dVISE is a virtual reality environment developed by the British company Division for manufacturing simulation [Ghe95] [Div95b]. It comes with a collaborative design feature that allows multiple users in remote locations to connect to the simulation and to review and interact with complex assemblies. Each user in the collaborative session is represented by a user-definable object. Individuals can pick up parts, move them around, and hand them to one another in real time.

A.5.3 Virtual Classrooms

Most work on *CSCL (Computer Supported Cooperative Learning)* concentrates more on the pedagogical and interaction issues rather than on the basic technical environments required for building wide-area distributed virtual classrooms [CSC95] [OM95]. The essential constituents of such classrooms are typically video-conferencing systems and electronic whiteboards as employed in the MUNIN project [MUN96] that applys MBONE multimedia conferencing tools piloted in the MICE project [MIC97].

The ARPA program CAETI (Computer Assisted Education and Training Initiative) [CAE96] is concerned with the provision of intelligent tutoring capabilities in an Internet infrastructure. It supports individualized learning regardless of the availability of local resources, as well as enhancing group training and education through multiuser environments and simulation. One of its initiatives, Collaborative Applications for Project-based Education Resources (CAPER) includes multimedia environments supporting involvement, experimentation, exploration, and collaboration in cross-disciplinary projects.

The ALN Web (Web of Asynchronous Learning Networks) [ALN97] documents various activities in this area.

References

[Adl95] R. Adler. Distributed Coordination Models for Client/Server Computing. *IEEE Computer*, 28(4):14–22, April 1995.

[Agl97] IBM Aglets Workbench – Programming Mobile Agents in Java, 1997. http://www.trl.ibm.co.jp/aglets/.

[AHLM96] J.-M. Andreoli, C. Hankin, and D. Le Metayer, editors. *Coordination Programming: Mechanisms, Models and Semantics.* Imperial College Press, London, UK, August 1996.

[ALN97] ALN Web — The Web of Asynchronous Learning Networks, 1997. http://www.aln.org/.

[AM95] Frank Andrews, Keith aand Kappe and Hermann Maurer. Serving Information to the Web with Hyper-G. *Computer Networks and ISDN Systems*, 27(6), April 1995. Also: Proceedings of the Third International World-Wide Web Conference, WWW'95, Darmstadt, Germany, April 1995.
http://www.tu-graz.ac.at/0x811b9908_0x00167d20.

[And96] Hakan Andersson. VRML Behaviour — A Proposal, December 1996.
http://www.lysator.liu.se/~zap/vr_prop1.html.

[ATM96] *1996 ATM Summer School*, Johannes Kepler University, Linz, Austria, July 1996.

[BCE+92] Bruno Buchberger, George Collins, Mark Encarnación, Hoon Hong, Jeremy Johnson, Werner Krandick, Rüdiger Loos, Ana Mandache, Andreas Neubacher, and Herbert Vielhaber. A SACLIB Primer. Technical Report 92-34, RISC-Linz, Johannes Kepler University, Linz, Austria, 1992.

[BE95] W. Broll and D. England. Adding Multi-User Support to VRML, 1995.

[Ber96] Philip A. Bernstein. Middleware — A Model for Distributed System Services. *Communications of the ACM*, 39(2):86–98, February 1996.

[BF81] Bruno Buchberger and Lichtenberger Franz. *Mathematics for Computer Science I — The Method of Mathematics.* Springer, Berlin, 2nd edition, 1981.

[BGL77] S. Benford, C. Greenhalgh, and D. Lloyd. Crowded Collaborative Virtual Environments, 1977.
ftp://turing.cs.nott.ac.uk/pub/papers/CHI97a.ps.gz,
ftp://turing.cs.nott.ac.uk/pub/papers/CHI97b.ps.gz.

[BHP96] Piergiorgio Bertoli, Hoon Hong, and Ovidiu Podisor. The Design of the Sturm Parallel Kernel. Technical Report 96-11, RISC-Linz, Johannes Kepler University, Linz, Austria, May 1996.

[BJ+95] Bruno Buchberger, Withold Jacak, et al. he College of Media Techniques and Design (in German), 1995. Description of the College of MTD in Hagenberg, Austria.

[BMZ95] Donald P. Brutzman, Michael R. Macedonia, and Michael J. Zyda. Internetwork Infrastructure Requirements for Virtual Environments. In *NII 2000 Forum of the Computer Science and Telecommunications Board*, National Research Council, Washington, D.C., May 1995.
ftp://taurus.cs.nps.navy.mil/pub/auv/brutzman/nii_2000.txt.

[Buc78] Bruno Buchberger. Computer-Trees and their Programming. In Dauchet, editor, *4th Colloquium "Trees in algebra and programming"*, pages 1–18, University of Lille, February 16–18, 1978. Also: Technical Report 78-11, RISC-Linz, Johannes Kepler University, Linz, Austria, 1978.

[Buc85] Bruno Buchberger. *Groebner-Bases: An Algorithmic Method in Polynomial Ideal Theory*, chapter 6, pages 184–232. Reidel Publishing Company, Dodrecht, Germany, 1985.

[Buc93] Bruno Buchberger. Mathematics: An Introduction to Mathematics Integrating the Pure and Algorithmic Aspect. Volume I: The Language of Mathematics. Lecture notes in the form of a mathematica notebook, Research Institute for Symbolic Computation, Johannes Kepler University, Linz, Austria, 1993. In preparation as a book with Springer, Vienna.

[Buc96a] Bruno Buchberger. Computer Algebra and Logic. In F. Baader and K.U. Schulz, editors, *Frontiers of Combining Systems*, Applied Logic Series, pages 193–220, Munich, Germanyu, March 26–29, 1996. Kluwer Academic Publishers. Invited paper.

[Buc96b] Bruno Buchberger. Mathematica as a Rewrite Language. In Tetsuo Ida, editor, *Fuji Conference on Functional Logic Programming*, Shonan Village, Japan, November 1–4, 1996. Telos publishing conference. Invited paper.

[Buc96c] Bruno Buchberger. Proving, Solving, Computing. In Tetsuo Ida and Yike Guo, editors, *MLP '96 — Multiparadigm Logic Programming*, Bonn, Germany, September 5, 1996. Springer, Vienna. Invited paper.

[Buc96d] Bruno Buchberger. Using Mathematica for Doing Simple Mathematical Proofs. In *4th Mathematica Users' Conference*, pages 80–96, Tokyo, Japan, November 2, 1996. Wolfram Media Publishing. Invited paper.

[Buc97a] Bruno Buchberger. *Mathematica: A System for Doing Mathematics by Computer?* Springer, New York, 1997. To appear.

[Buc97b] Bruno Buchberger. *Thinking, Speaking, Writing: A Practical Course in the Working Techniques of Mathematicians and Computer Scientists.* Springer, Vienna, 1997. In preparation.

[CAE96] Computer Assisted Education and Training Initiative (CAETI), 1996.
http://www.cs.unc.edu/caeti.

[CCR96] S. Castellani, P. Ciancarini, and D. Rossi. The ShaPE of ShaDe: A Coordination System. Technical Report UBLCS-96-5, Department of Computer Science, University of Bologna, Italy, March 1996.
ftp://ftp.cs.unibo.it:/pub/TR/UBLCS/96-05.ps.gz.

[CEF+93] M. Cutkosky, R. Engelmore, R. Fikes, M. Genesereth, T. Gruber, W. Mark, J. Tenenbaum, and J. Weber. PACT: An Experiment in Integrating Concurrent Engineering Sessions. *IEEE Computer*, 26(1):8–37, January 1993.
http://www.eit.com/creations/papers/pact/.

[CH95] Christer Carlsson and Olof Hagsand. DIVE — A Multi-user Virtual Reality System. In *1995 IEEE Virtual Reality Annual Symposium (VRAIS '93)*, Seattle, OR, September 1995.

[Cia96a] P. Ciancarini. Coordination Models and Languages as Software Integrators. *ACM Computer Surveys*, 28(2):300–302, 1996.

[Cia96b] Paolo Ciancarini. Coordination Models and Languages. Technical report, Department of Computer Science, University of Bologna, Italy, January 1996.
http://www.cs.unibo.it/~cianca/coord_ToC.html.

[CL96] Daniel T. Chang and Danny B. Lange. Mobile Agents: A New Paradigm for Distributed Object Computing on the WWW. In *OOPSLA'96 Workshop Toward the Integration of WWW and Distributed Object Technology*, Atlanta, Georgia, October 5–9, 1996. ACM Press. http://www.trl.ibm.co.jp/aglets/ma.html.

[Col89] Murray I. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation.* Pitman/MIT Press, 1989.

[COM96] The Component Object Model Specification — Draft, 1996.
ftp://ftp.microsoft.com/developr/drg/OLE-info/COMSpecification/.

[CR97] K. Mani Chandy and Adam Rifkin. Systematic Composition of Objects in Distributed Internet Applications: Processes and Sessions. In *HICCS 97 Mini-track on Software Engineering for Distributed Systems*, January 7–10, 1997.
ftp://ftp.cs.caltech.edu/tr/cs-tr-96-15.ps.Z.

[CRS+96] K. Mani Chandy, Adam Rifkin, A. G. Sivilotti, Jacob Mandelson, Matthew Richardson, Wesley Tanaka, and Luke Weisman. A World-Wide Distributed System Using Java and the Internet. In *High Performance Distributed Computing (HPDC-5) Focus Workshop on Multimedia and Collaborative Environments*, Syracuse, NY, August 6–9, 1996.
http://www.infospheres.caltech.edu/papers/hpdc96.ps.

[CS96] K. Mani Chandy and Eve M. Schooler. Designing Directories in Distributed Systems: A Systematic Framework. In *High Performance Distributed Computing (HPDC-5) Focus Workshop on Multimedia and Collaborative Environments*, Syracuse, NY, August 6–9, 1996.
http://www.infospheres.caltech.edu/papers/hpdc96b.ps.

[CSC95] *CSCL '95 — Computer-Supported Collaborative Learning*, Indiana University, Bloomington, Indiana, October 17–20, 1995. Lawrence Erlbaum Associates.
http://www-cscl95.indiana.edu/cscl95/toc.html.

[CTV96a] Paolo Ciancarini, Robert Tolksdorf, and Fabio Vitali. Towards an Interactive Web, 1996. Submitted to IEEE Internet Computing,
http://www.cs.tu-berlin.de/˜pagespc/ieeeip/ciancarini.html.

[CTV+96b] Paolo Cinciarini, Robert Tolksdorf, Fabio Vitali, Davide Rossi, and Andreas Knowche. Distributed Applications on the WWW: Coordinating Java Agents with PageSpace, 1996. Submitted to the IEEE Computer Special Issue on Executable Content: Java and Related Technologies,
http://www.cs.tu-berlin.de/˜pagespc/ieeecsp/ieeec.html.

[CVE97] The CRG Virtual Environment (CVE), 1997.
http://www.crg.cs.nott.ac.uk/crg/Research/crgshare/.

[D+93] John Darlington et al. Parallel Programming Using Skeleton Functions. In A. Bode, M. Reeve, and G. Wolf, editors, *PARLE '93 Parallel Architectures and Languages Europe*, volume 694 of *Lecture Notes in Computer Science*, pages 146–160, Munich, Germany, 1993. Springer, Berlin.

[DCO96] Distributed Component Object Model (DCOM) Binary Protocol, May 1996.
http://www.microsoft.com/intdev/prog-gen/dcom.htm.

[DIV95a] DIVE - The SICS Distributed Interactive Virtual Environment, 1995.
http://www.sics.se/dce/dive.html.

[Div95b] Division Home Page, 1995.
http://www.division.co.uk/.

[FGN+95] I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke. Software Infrastructure for the I-WAY High-Performance Distributed Computing Experiment. In *Proceedings of the 5th IEEE Symposium on High Performance Distributed Computing*, 1995.
ftp://ftp.mcs.anl.gov/pub/nexus/reports/isoft.ps.Z.

[FHS96] Fachhochschule Hagenberg, Obersterreich, 1996. http://www.fhs-hagenberg.ac.at/.

[FK96] Ian Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. In *Proceedings of the Workshop on Environments and Tools*. SIAM, 1996.
ftp://ftp.mcs.anl.gov/pub/nexus/reports/globus.ps.Z.

[FKPT96] Ian Foster, Carl Kesselman, James Patton, and Steven Tuecke. Nexus User's Guide (Version 3.0). Technial report, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.
http://www.mcs.anl.gov/Projects/nexus/uguide_3.0/index.html,
ftp://ftp.mcs.anl.gov/pub/nexus/reports/nexus_uguide_v3.0.ps.Z,.

[FKT94] Ian Foster, Carl Kesselman, and Steven Tuecke. Nexus: Runtime Support for Task-Parallel Programming Languages. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory, 1994.
http://www.mcs.anl.gov/Projects/nexus/paper/index.html,
ftp://ftp.mcs.anl.gov/pub/nexus/reports/nexus_paper.ps.Z.

[FKT97] Ian Foster, Carl Kesselman, and Steven Tuecke. The Nexus Approach to Integrating Multithreading and Communication. *Journal of Parallel and Distributed Computing*, 1997. To appear.
ftp://ftp.mcs.anl.gov/pub/nexus/reports/jpdc.ps.Z.

[Fla96] David Flanagan. *Java in a Nutshell — A Desktop Quick Reference for Java Programmers*. O'Reilly, Bonn, 1996.

[FT89a] Ian Foster and Stephen Taylor. *Strand — New Concepts in Parallel Programming*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[FT89b] Ian Foster and Stephen Taylor. *Strand: New Concepts in Parallel Programming*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[FT94] Ian Foster and Stephen Taylor. A Compiler Approach to Scalable Concurrent-Program Design. *ACM Transactions on Programming Languages and Systems*, 16(3):577–604, May 1994.

[GB95] C. Greenhalgh and S. Benford. MASSIVE: A Distributed Virtual Reality System Incorporating Spatial Trading. In *IEEE 15th International Conference on Distributed Computing Systems (DCS '95)*, Vancouver, Canada, May 30 – June 2, 1995. IEEE Computer Society.

[GC89] David Gelernter and Nicholas Carriero. Linda in Context. *Communications of the ACM*, 32(4):444–458, April 1989.

[GC92] David Gelernter and Nicholas Carriero. Coordination Languages and their Significance. *Communications of the ACM*, 35(2):97–107, February 1992.

[Ghe95] Steve Ghee. dVS — a Distributed VR System Infrastructure. In *SIGGRAPH 95*, Los Angeles, California, August 6–11, 1995. ACM Press.

[Glo96] Globus: Infrastructure for Internet Computations, 1996. http://www.globus.org/.

[Gre95] Chris Greenhalgh. MASSIVE, 1995. http://www.crg.cs.nott.ac.uk/˜cmg/massive.html.

[Gro95a] Object Management Group. The Common Object Request Broker Architecture and Specification (Revision 2.0), 1995. http://www.omg.org/corbask.htm, http://www.cs.wustl.edu/ schmidt/CORBA-docs/corba2.0.ps.gz.

[Gro95b] Stanford Distributed Systems Group. The PARADISE Project, 1995. http://www-dsg.stanford.edu/paradise.

[Gro96] The Infospheres Group. II: The Infosphere Infrastructure User's Guide (Version 1.0b1). Technical report, California Institute for Technology, Pasadena, CA, November 1996. http://www.infospheres.caltech.edu/releases/current/documentation /user_guide/user_guide.ps, http://www.infospheres.caltech.edu/releases/current/documentation /user_guide/html/user_guide.html.

[GW96] A. Grimshaw and W. Wolf. Legion — a View from 50,000 Feet. In *5the IEEE Symposium on High Performance Distributed Computing*, pages 89–99. IEEE Computer Society Press, 1996.

[GWtLT97] Andrew S. Grimshaw, Wm.A. Wulf, and the Legion Team. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1):39–45, January 1997.

[HL94] Hoon Hong and Hans-Wolfgang Loidl. Parallel Computation of Modular Multivariate Polynomial Resultants on Shared Memory Machine. In Bruno Buchberger and Jens Volkert, editors, *Parallel Processing: CONPAR 94 — VAPP VI International Conference on Parallel and Vector Processing*, volume 854 of *Lecture Notes in Computer Science*, pages 325–336, Linz, Austria, September 6–8, 1994. Springer, Berlin.

[HNS95] Hoon Hong, Andreas Neubacher, and Wolfgang Schreiner. The Design of the SACLIB/PACLIB Kernels. *Journal of Symbolic Computation*, 19:111–132, 1995.

[HNSS94] Hoon Hong, Andreas Neubacher, Volker Stahl, and Wolfgang Schreiner. The C++ Interface to the STURM Multi-Processor Kernel. Technical Report 94-32, RISC-Linz, Johannes Kepler University, Linz, Austria, April 1994.

[Hon92] Hoon Hong. Parallelization of Quantifier Elimination on Workstation Network. Technical Report 91–51, Research Institute for Symbolic Computation (RISC-Linz), Johannes Kepler University, Linz, Austria, May 1992.

[HSN+92] Hoon Hong, Wolfgang Schreiner, Andreas Neubacher, Kurt Siegl, Hans-Wolfgang Loidl, Tudor Jebelean, and Peter Zettler. PACLIB User Manual. Technical Report 92-32, RISC-Linz, Johannes Kepler University, Linz, Austria, May 1992. Also: Technical Report ACPC/TR 92-9, Austrian Center for Parallel Computation, July 1992.

[Hyp96] HyperWave, 1996.
http://www.tu-graz.ac.at/hyperg-0x811b9908.

[Inf97] Caltech Infospheres Project, 1997.
http://www.infospheres.caltech.edu/.

[IWA95] I-WAY, 1995.
http://www.iway.org/.

[Jad96] The Jada Home Page, 1996.
http://www.cs.unibo.it/~rossi/jada/.

[Jav96a] Java Computing, 1996.
http://java.sun.com.

[Jav96b] JavaBeans — Component APIs for Java, 1996.
http://java.sun.com/beans.

[KMW+94] D. Kuokka, J. McGuire, J. Weber, J. Tenenbaum, T. Gruber, and G. Olsen. SHADE: Knowledge-based Technology for the Re-Engineering Problem, 1994. http://www-ksl.stanford.edu/knowledge-sharing/papers/shade-overview.html.

[Lau93] Documents on the coordination languages Laura and ALICE, 1993.
http://flp.cs.tu-berlin.de/homepage/subsection3.1.3.html.

[Leg] The Legion Project.
http://www.cs.virginia.edu/~legion.

[Lin96] The Yale Linda Group, 1996.
http://www.cs.yale.edu/HTML/YALE/CS/Linda/linda.html.

[Loe95] Carl E. Loeffler. Distributed Virtual Reality: Applications for Entertainment and Industry, 1995.
http://www.nta.no/telektronikk/4.93.dir/Loeffler_C_E.html.

[LP94] Carla Limongelli and R. Pirastu. Exact Solution of Linear Systems over Rational Numbers by Parallel p-adic Arithmetic. In Bruno Buchberger and Jens Volkert, editors, *Parallel Processing: CONPAR 94 — VAPP VI International Conference on Parallel and Vector Processing*, volume 854 of *Lecture Notes in Computer Science*, pages 313–324, Linz, Austria, September 6–8, 1994. Springer, Berlin.

[MAD96] Manufacturing Automation and Design Engineering (MADE), 1996.
http://elib.cme.nist.gov/made/made.html.

[Mau96] Hermann Maurer. *HyperWave — The Next Generation Web Solution*. Addison-Wesley, 1996. http://www.tu-graz.ac.at/0x811b9908_0x001bfd8b.

[MIC97] MICE National Support Centre England, 1997.
http://www-mice-nsc.cs.ucl.ac.uk/mice-nsc/.

[MUN96] Distributed Electronic Classrooms — The MUNIN Project, 1996.
http://munin.uio.no/English/intro.html.

[MZP+94] Michael R. Macedonia, R. Zyda, David R. Pratt, et al. NPSNET: A Network Software Architecture for Large Scale Virtual Environments. *Presence*, 3(4):265–287, Fall 1994.

[Nex97] The Nexus Multithreaded Runtime System, 1997.
http://www.mcs.anl.gov/nexus.

[NPS95] The NPSNET Research Group Home Page, 1995.
http://www.cs.nps.navy.mil/research/npsnet/.

[OaE96] Orfali and Harkey aand Edwards. *The Essential Distributed Objects Guide*. Wiley, 1996.

[OM95] C. O' Malley. *Computer Supported Collaborative Learning*. Springer, Berlin, 1995.

[Ope96] The Club OpenDoc Home Page, 1996.
http://www.software.ibm.com/clubopendoc/.

[Pag97] The PageSpace Effort, 1997.
http://www.cs.tu-berlin.de/~pagespc/.

[Roe95a] Bernie Roehl. Distributed Virtual Reality — An Overview (First Draft), June 1995.
http://sunee.uwaterloo.ca/~broehl/distrib.html.

[Roe95b] Bernie Roehl. Some Thoughts on Behavior in VR Systems, August 1995.
http://sunee.uwaterloo.ca/ broehl/behav.html.

[Sch94a] Wolfgang Schreiner. *Parallel Functional Programming for Computer Algebra*. PhD thesis, Johannes Kepler University, Linz, Austria, September 1994. RISC-Linz Report 94-61.

[Sch94b] Wolfgang Schreiner. Virtual Tasks for the PACLIB Kernel. In Bruno Buchberger and Jens Volkert, editors, *Parallel Processing: CONPAR 94 — VAPP VI International Conference on Parallel and Vector Processing*, volume 854 of *Lecture Notes in Computer Science*, pages 533–544, Linz, Austria, September 6–8, 1994. Springer, Berlin.

[Sch95] Wolfgang Schreiner. Application of a Para-Functional Language to Problems in Computer Algebra. In Wim Bhm and T. Feo, John, editors, *High Performance Functional Computing*, pages 10–24, Denver, Colorado, April 9–11, 1995. Lawrence Livermore National Laboratory Report CONF-9504126.

[Sch96a] Wolfgang Schreiner. A Para-Functional Programming Interface for a Parallel Computer Algebra Package. *Journal of Symbolic Computation*, 21:523–614, 1996. Special Issue on Parallel Symbolic Computation.

[Sch96b] Wolfgang Schreiner. A Para-Functional Programming Interface for a Parallel Computer Algebra Package. *Journal of Symbolic Computation, Special Issue on Parallel Symbolic Computation*, 21:523–614, 1996.
http://www.risc.uni-linz.ac.at/people/schreine/papers/jsc96.ps.gz.

[SH93a] Wolfgang Schreiner and Hoon Hong. A New Library for Parallel Algebraic Computation. In Richard F. Sincovec et al., editors, *Sixth SIAM Conference on Parallel Processing for Scientific Computing*, volume II, pages 776–783, Norfolk, Virginia, March 22–24, 1993. SIAM.

[SH93b] Wolfgang Schreiner and Hoon Hong. PACLIB — A System for Parallel Algebraic Computation on Shared Memory Computers. In Hussein M. Alnuweiri, editor, *Parallel Systems Fair at the Seventh International Parallel Processing Symposium*, pages 56–61, Newport Beach, CA, April 14, 1993. IPPS '93. Also: Technical Report 93-03, RISC-Linz, Johannes Kepler University, Linz, Austria, February 1993. Also: Technical Report ACPC/TR 93-12, Austrian Center for Parallel Computation, February 1993.

[SH93c] Wolfgang Schreiner and Hoon Hong. The Design of the PACLIB Kernel for Parallel Algebraic Computation. In Jens Volkert, editor, *Parallel Computation — Second International Conference of the Austrian Center for Parallel Computation*, volume 734 of *Lecture Notes in Computer Science*, pages 204–218, Gmunden, Austria, October 4–6, 1993. Springer, Berlin.

[Sie92] Kurt Siegl. ||MAPLE||: A System for Parallel Computer Algebra for Transputers. In Szabolcs Ferenczi, editor, *1st Austrian-Hungarian Workshop on Transputer Applications*, number KFKI-1992-34/M,N in Technical Report Series of the Hungarian Academy of Sciences Central Research Institute for Physics, pages 21–30, Sopron, Hungary, October 8–10, 1992. Hungarian Transputers User

Group and Austrian Center for Parallel Computation. Also: Technical Report 93-06, RISC-Linz, Johannes Kepler University, Linz, Austria, February 1993.

[Sie93a] Kurt Siegl. ||MAPLE|| — A System for Parallel Symbolic Computation. In Hussein M. Alnuweiri, editor, *Parallel Systems Fair at the Seventh International Parallel Processing Symposium*, pages 62–67, Newport Beach, California, April 14, 1993. IPPS '93. Also: Technical Report 93-07, RISC-Linz, Johannes Kepler University, Linz, Austria, February 1993.

[Sie93b] Kurt Siegl. Parallelizing Algorithms for Symbolic Computation Using ||MAPLE||. In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 179–186, San Diego, California, May 19-22, 1993. ACM Press.

[Sno95] David N. Snowdon. AVIARY, 1995. http://www.crg.cs.nott.ac.uk/%7Edns/vr/aviary/.

[Tol96] Robert Tolksdorf. Coordinating Services in Open Distributed Systems with Laura. In Paolo Ciancarini and Chris Hankin, editors, *Coordination Languages and Models, Proceedings of Coordination '96*, volume 1061 of *Lecture Notes in Computer Science*, pages 386–402. Springer, Berlin, 1996.

[VRM96] Virtual Reality Modeling Language, 1996. http://vag.vrml.org.

[Wal95] Bernhard Wall. Mathematics: An Introduction to Mathematics Integrating the Pure and Algorithmic Aspect. Volume III: Algebra. Lecture notes in the form of a mathematica notebook, Research Institute for Symbolic Computation, Johannes Kepler University, Linz, Austria, 1995.

[Wan91] Dongming Wang. On the Parallelization of Characteristic-Set-Based Algorithms. In *Parallel Computation — First International ACPC Conference*, volume 591 of *Lecture Notes in Computer Science*, pages 316–337, Salzburg, Austria, September 30–October 2, 1991. Springer, Berlin.

[WHH+93] A. J. West, T. L. J. Howard, R. J. Hubbold, et al. AVIARY — A Generic Virtual Reality Interface for Real Applications. In R. A. Earnshaw, M. A. Gigante, and H. Jones, editors, *Virtual Reality Systems*, chapter 15, pages 213–236. Academic Press, March 1993.

[Win94] Wolfgang Windsteiger. Mathematics: An Introduction to Mathematics Integrating the Pure and Algorithmic Aspect. Volume II: Analysis. Lecture notes in the form of a mathematica notebook, Research Institute for Symbolic Computation, Johannes Kepler University, Linz, Austria, 1994.

[Wol96] Stephen Wolfram. *The Mathematica Book*. Cambridge University Press, 3rd edition, 1996.

[Wol97] Wolfram Research: Makers of Mathematica, 1997. http://www.wolfram.com/.

[Zet92] Peter Zettler. RISC-Linda — A Dialect of Linda. Master's thesis, Johannes Kepler University, Linz, Austria, 1992.

RISC-Linz
Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria, Europe

CONCERT: A Software Architecture for Coordinating Education Sessions in Distributed Environments

Bruno BUCHBERGER, Wolfgang SCHREINER

(February 1997)

RISC-Linz Report Series No. 97-04

Supported by: IPA