Summary of Papers

Part I: Language Issues

Abstraction and Modularity Mechanisms for Concurrent Computing
This paper describes a paradigm to support abstraction and modularity in concurrent systems. The constructs proposed allow generic and reusable specification of coordination patterns, temporal ordering, resource management, and dependability protocols. The development is in the framework of the Actor model.

Tradeoffs between Reasoning and Modeling
Object-oriented and logic-programming paradigms are shown to be incompatible as component-based models of computation. This “impossibility result,” based on a new notion of observability of interactions among components, suggests that combining object-oriented with logic programming is not merely hard but impossible. It implies the more general incompatibility of reasoning and modeling and the impossibility of reducing modeling to reasoning as in automatic program verification. The paper proposes practical research that will contribute to the development of component-based software technology.

A Survey of Logic Programming-Based Object-Oriented Languages
Languages which augment logic programming with object-oriented programming features are surveyed by considering how they deal with the representation of objects and classes, message passing, state, inheritance, and their relationship with other paradigms. The ideas behind concurrent logic programming are also described in some detail. The paper concludes with speculation on future directions for the field.

Analysis of Inheritance Anomaly in Object-Oriented Concurrent Programming Languages
Inheritance is a key feature in many object-oriented languages. It has been pointed out that inheritance and synchronization constraints in concurrent object systems often conflict with each other, resulting in an inheritance anomaly which requires re-definitions of inherited methods in order to maintain the integrity of concurrent objects. The paper proposes a scheme whereby high degree of superclass encapsulation can be achieved for code re-use while retaining utmost efficiency on conventional massively parallel architectures.

Composing Active Objects
Many of the shortcomings of present-day object-oriented programming languages can be traced to two phenomena: the lack of general support for software composition, and the semantic interference between language features addressing operational and compositional aspects of object-oriented programming. To remedy this situation, the paper proposes the development of a “pattern language” for active objects in which objects are constructed by composing software patterns. The relevant semantic issues are also discussed.
Part II: Programming Constructs

Supporting Modularity in Highly-Parallel Programs

The Actor model is extended to allow the construction of multi-access data abstractions. This extended actor model, called the Aggregate Model, introduces the notion of collections of actors (aggregates) – accessible by a single group name. Messages sent to the group are directed to one of the members of the collection. Because each of the actors can receive messages concurrently, the abstraction implemented by the group need not be serializable. The paper describes four different usage paradigms for aggregates. In addition, two detailed program examples are given; these examples show how the introduction of aggregates decreases serialization and increases program concurrency.

Multiple Concurrency Control Policies in an Object-Oriented Programming System

Different parallel and distributed applications have different consistency models, so multiple concurrency control policies are needed. When objects are shared among applications with different policies, multiple policies must operate simultaneously and compatibly. The paper shows how the distributed object-oriented programming language MELD is used to support concurrency models. The relevant language constructs, programming problems, and implementation issues are described.

Ports for Objects in Concurrent Logic Programs

Ports are introduced to provide communication support for object-oriented programming in concurrent constraint logic programming languages. From a pragmatic point of view ports provide efficient many-to-one communication, object identity, mechanisms for garbage collection of objects, and opportunities for optimised compilation techniques for concurrent objects. From a semantic point of view, ports preserve the monotonicity of the constraint store which is a crucial property of all concurrent constraint languages. Ports are available in AKL, the Andorra Kernel language, a concurrent logic programming language that provides general combinations of don’t know and don’t care nondeterministic computations.

Part III: Language Design

Specifying Concurrent Languages and Systems with Δ-Grammars

The use of graph grammars for specifying concurrent systems and languages is described. The model used in this paper, Δ-GRAMMARS, is rooted in existing graph grammar theory and provides a convenient framework in which to specify both static and dynamic concurrent systems.

Interaction Abstract Machines

Linear Objects is an abstract linguistic model for concurrent computation whose theoretical background is given by Linear Logic, a logic recently introduced by Jean-Yves Girard to provide a theoretical account for the notion of action. This paper characterizes
Summary of Papers

Linear Object computations in terms of Interaction Abstract Machines (IAMs), in the same vein of such metaphors as the Chemical Abstract Machine. IAMs allow interactions among independent, locally defined subsystems — a crucial requirement for capturing the global behavior of open systems.

CC++: A Declarative Concurrent Object-Oriented Programming Notation

Compositional C++ is a parallel object-oriented notation which extends C++. The goals of CC++ include developing a theory, notation, and tools for reliable scalable concurrent program libraries. CC++ also aims to unify declarative programs and object-oriented imperative programming. This paper is a brief description of CC++, and its extensions. Included are some example programs together with reasoning about their correctness.

A Logical Theory of Concurrent Objects and Its Realization in the Maude Language

A new theory of concurrent objects is presented. The theory has the important advantage of being based directly on a simple logic called rewriting logic in which concurrent object-oriented computation exactly corresponds to logical deduction. An axiomatization of objects, classes, and concurrent object-oriented computations in terms of rewriting logic is proposed as a general semantic framework for object-oriented programming. The theory is used to develop a new language, called Maude. Maude provides a simple and semantically rigorous unification of functional programming and concurrent object-oriented programming. The relationship with Actors and with other models of concurrent computation is discussed. The model theory of rewriting logic and an initial model semantics for Maude modules are also presented.

Part IV: Operating Systems

CHOICES: A Parallel Object-Oriented Operating System

The Choices parallel object-oriented operating system design is a collection of interconnected frameworks. Each subframework’s design specifies the possible concurrent messages, control flow, and synchronization between a dynamic number of component objects. In addition, objects within a subframework have other dependency relationships that vary dynamically. Choices has a number of implementations on different hardware platforms and with different resource allocation and management algorithms. The paper describes the problem-oriented models of concurrency and communication implemented in Choices as attributes that are defined for an abstract subframework and inherited by more concrete subframeworks.

COSMOS: An Operating System for a Fine-Grain Concurrent Computer

COSMOS is an operating system for the J-Machine, a fine-grain message-passing concurrent computer. COSMOS provides a global virtual namespace, object-based memory management, support for distributed objects, and low-overhead context switching. Its memory management system provides fast, transparent access to storage distributed
across the machine. COSMOS is designed to efficiently support fine-grain concurrent computation and is tailored for an environment where local computation is inexpensive. COSMOS provides a shared global name space across the nodes of a message-passing concurrent computer. In COSMOS, all data and code are stored in objects. Objects are free to migrate between nodes to balance load or to exploit locality. The system also supports distributed objects.

Part V: Performance Monitoring

Monitoring Concurrent Object-Based Programs

The paper develops a simulation of multiprocessor architectures executing concurrent applications. The simulation has focussed on the communication and scheduling support facilities of architectures. The monitoring facilities of the simulator have then aided in identifying bottlenecks—both in the application and in the underlying multiprocessor system—leading to refined implementations with better performance. The applications are programmed in a concurrent, object-oriented extension of Lisp called Lamina. The instrumentation facility directly exploits the programming model. The facility provides a comprehensive, customizable library of modules for aggregating this information, for organizing it along various dimensions, and for presenting it visually.