MIT9904-12
Cooperative Computing in Dynamic Environments

Nancy Lynch and Idit Keidar

Overview

Both the Cooperative Computing research group at NTT and the Theory of Distributed Systems research group at MIT are carrying out research on theoretical foundations of distributed computing, with a focus on cooperative group activities in networks. Such group activities range from human social activities in cyber communities to powerful distributed applications involving data sharing and cooperative work. Such activities are often supported by *agent communication* services, which provide distributed intelligence, or by *group communication* services, which manage group membership and guarantee coherent communication.

An important characteristic of the environments in which such activities take place is their *dynamicity*: participants come and go (and change their location), network topology changes, and components fail and recover. Coping with such difficult environments leads to complex implementations, which are difficult to build, understand, and analyze. Formal, mathematical techniques can help to make such implementations more tractable.

Both the NTT and MIT research groups are addressing these problems using formal modeling and verification techniques. The techniques used by the two groups have, in the past, been different: The NTT group has been emphasizing certain *language-based methods*, including process algebras such as the pi-calculus, modal logics, and temporal logics, and *automatic verification tools* such as model-checkers. In contrast, the MIT group has been emphasizing *mathematical methods* based on (asynchronous and timed) input/output automaton models, and *interactive verification tools* such as interactive theorem-provers.

This project, begun in September, 1999, aims at combining these methods into a strong, coherent foundation for understanding cooperative group activities in networks.

Problems

Our work involves developing formal models and validation methods for agent communication services and group communication services in dynamically-changing networks, and for cooperative computing applications built on top of these services.

Agents are computational entities that communicate using a special ``Agent Communication Language'' (ACL). In a basic communication step, an agent receives an ACL message over an incoming communication channel, changes its internal state and sends some new messages over an outgoing channel. We are mainly interested in flexible agent systems, in which agents and communication channels are dynamically created and destroyed.

Although agent communication systems are popular, they so far do not have a mathematical foundation adequate for supporting formal language definitions and formal verification methods. Milner's pi-calculus appears to be a useful starting point for such a mathematical foundation. For example, Kogure's group at NTT has developed a network programming language, Nepi^2, based on the pi-calculus, and is currently developing an agent programming language on top of Nepi^2. However, this agent language depends on extensions to Nepi^2 and to the pi-calculus, to accommodate issues of timing, failures, and human interface behavior. These extensions require a formal foundation.

Group communication services are powerful distributed communication services that allow application processes located at different nodes of a distributed network to operate collectively as a group, using the service to multicast messages to all members of the group. Such a service is based on a group membership service, which provides each group member with a view of the group, including a list of the processes that are members of the group. The service manages both consistent delivery of messages within each view and the reconfiguration involved in changing views. Such systems are used for applications in many areas, including banking, stock market, air-traffic control computer-supported cooperative work, and interactive games.

Building on a large body of prior work, members of Lynch's group have developed formal I/O automaton-based models for a collection of group communication services and algorithms, and have used these models in proofs of safety, performance and fault-tolerance properties. In developing the service and algorithm models, they have worked closely with network and distributed system developers. This work has resulted not only in good models and proofs, but also in the discovery of significant logical errors in three real systems (Orca, Ensemble, and Horus).

Approach

The modeling/verification framework used by Lynch's group at MIT is based on:

- The untimed and timed I/O automaton models of Lynch, Tuttle, and Vaandrager.

- Methods of structural decomposition using parallel composition and levels of abstraction.

- Methods of proof involving invariant assertions, formal abstraction relationships, compositional reasoning, and conditional performance/fault-tolerance analysis. The basic framework is defined mathematically, which allows it to support descriptions in different languages.

We believe that I/O automaton models can be used as the mathematical basis for semantic definitions and proof methods for agent programming languages, group communication services, and other primitives for
cooperative group activities in dynamic networks. The basic communication behavior of agent communication languages is easily expressed in this style. However, some features that are fundamental to the agent application domain, such as dynamic process creation, open system behavior, and failures,
are not represented explicitly in the basic automaton models. It will be important to identify such features and develop ``standard'' patterns for modeling those on top of the basic automaton models. For instance, dynamic process creation can be modeled by treating the components that are created at runtime as if they were there (but inactive) from the beginning. Special "activate" actions could be used to model the dynamic creation steps.

We plan to work on using untimed and timed I/O automata to express the semantics of an extended pi-calculus sufficient to support agent communication. We are currently working on determining what must be added to our existing IOA modeling/programming language in order to express the notions needed for this domain. We are also using untimed and timed automata, and the IOA language, to provide models for group communication and services.

As such semantic definitions are developed, we are using them as the basis for modeling and analyzing a collection of typical examples from cooperative computing applications (e.g., computer-supported cooperative work, distributed knowledge bases and data bases). The models for these examples will emphasize structural decomposition, and the analyses will be based on the standard techniques for distributed algorithms (invariants, abstraction, conditional timing analyisis, composition). The examples will allow us to determine which techniques are the most useful and what they can accomplish (that is, what types of properties they can be used to formulate and prove, and for what types of systems). Since agent communication and group communication systems cope with similar environmental anomalies, we can compare how similar applications can be built on top of both.

Our initial work is being done by hand, in terms of the basic automaton models, because the main issues seem to involve devising good abstractions. However, it is also important to develop computer support for the most useful types of reasoning. This computer support will involve a combination of automatic tools (as currently used at NTT) and interactive tools (as used at MIT). Since the automaton framework is very general, it is capable of supporting such a combination of tools.

In developing this combination of tools, we expect to find that different tools will require different restrictions on the systems and properties being handled. For example, model-checkers require finite-state automata; theorem-provers require expressiveness in particular logics (LP is first-order). We will try to identify useful restrictions on systems and properties that will allow them to be handled with particular types of computer tools.

The implementation of agent communication services and group communication services involve interesting, complex distributed algorithms, running over rapidly-changing, fault-prone networks. For example, the communication manager of Nepi[2] manages rendezvous-style communication among processes, using a protocol that coordinates among local and global communication managers, using
LINDA-style tuple spaces. For another example, Keidar, Marzullo and Sussman are developing a novel group membership algorithm for WANs (also based on a combination of local and global communication managers). As challenge problems for the use of our methods, we will attempt to model and verify several of these algorithms.

Many of the results we would like to prove about distributed algorithms involve performance and fault-tolerance properties. We also plan work on developing general

techniques for compositional performance and fault-tolerance analysis. Some ideas for this appear in work by Fekete, Lynch, and Shvartsman on
view-synchronous group communication services.

Both Kogure's group and Lynch's group are interested in connecting models for distributed systems with actual runnable code. In order to establish such connections, it is important to model the data types used in distributed algorithms at multiple levels of abstraction. Our two groups appear to be taking similar approaches to this problem, using a combination of programming language code (Nepi^2 uses LISP, while IOA uses JAVA) and logical equations. We would like to combine our knowledge and ideas on this topic.

Specific Tasks

Specifically, we will work on:

1. Defining semantics for agents in distributed networks, using I/O automata and timed I/O automata. Identifying concepts that are fundamental to agent systems,
such as dynamic process creation and open system behavior, and that are not explicitly present in the basic automaton models. Developing standard patterns for modeling these concepts.

2. In the same style, defining semantics and identifying and modeling key concepts for group communication systems.

3. Modeling and analyzing typical examples from the cooperative computing application domain, using the automaton models for the underlying agent communication and group communication services.

4. Developing computer support for the types of reasoning that prove to be most useful in these examples, using a combination of automatic and interactive tools. Identifying useful restrictions on systems and properties that will allow them to be handled with the various types of computer tools.

5. Using automata and their standard proof techniques to model and verify particular communication protocols. Example protocols may include protocols from the Nepi^2 system developed at NTT and group communication protocols under study at MIT.

6. Developing techniques for analyzing timing and failures for distributed systems, in particular, techniques that can be used compositionally, and at multiple levels of abstraction.