Extending Operator Nets to Reason About Knowledge

Janice I. Glasgow
Department of Computing and Information Science,
Queen's University,
Kingston, Ontario, K7L 3N6
Phone: (613) 545-6050
Network: janice@qucis.bitnet

Randy Goebel
Department of Computer Science,
University of Alberta,
Edmonton, Alberta, T6G 2H1
Phone: (403) 432-2683
Network: goebel@alberta.uucp

Abstract: We present a model for reasoning about knowledge in a distributed system. This model is based on a behavioral semantics for operator nets that allows us to express abstract properties related to time and causality. Operator nets is a graphical language that has previously been used for describing and reasoning about interprocess communication and parallelism in distributed systems. Here we extend the operator net model to express temporal, knowledge-based specifications in an asynchronous distributed environment.
1. Introduction

Operator nets [Ashcroft 1985] is a graphical language that can be used to describe interprocess communication and parallelism in a distributed computing environment. An operator net consists of a set of nodes and a set of directed arcs. The nodes of a net denote process's and the arcs denote communication channels of a distributed system. A program in the language of operator nets is a set of equations that relate the output arc of a node to a function applied to the input arcs of the node. These equations can themselves be considered a language: the functional programming language Lucid [Wadge 1985]. A behavioral semantics for operator nets that models distributed systems has been defined [Glasgow 1987a], and used successfully to develop, implement and verify the multi-level secure system SNet [Glasgow 1987b; MacEwen 1987]. This paper describes an extension to the operator net model that allows for reasoning about knowledge, where knowledge is defined as a function of a process's initial knowledge, input history and reasoning capability.

The concept of knowledge in a distributed system is fundamental when trying to reason about properties that depend on the global state of knowledge of a set of reasoning processes. The operator net model provides a method for both describing and implementing knowledge-based specifications. In particular properties that relate knowledge to time and causality can be expressed and enforced using operator nets.

Our paper is organized as follows. Section 2 summarizes the motivation and use of knowledge in distributed computing environments and reviews some of the important properties of existing models. Section 3 presents an overview of the operator net model for distributed computing. Section 4 demonstrates how this model can easily be extended for reasoning about knowledge. We conclude by summarizing the work.
2. Knowledge and Distributed Computing

2.1 What is the problem?

Before considering what "knowledge" is, it is useful to consider the motivation for combining the concepts of distributed computing and knowledge. In other words, what problem gives rise to the need for this combination?

The motivation comes in two flavours. The first is a general concern for the specification and debugging of distributed protocols, which loosely names a specification of how control information is communicated in distributed computing environments. In contrast to algorithms for centralized computing systems where the flow of control information is an integral part of the specification, a distributed environment requires an extra level of specification that says how control information is communicated to co-ordinate computing at the distributed nodes.

Intuitively, the need for distributed protocols can be seen by comparing the following two situations. The first consists of a single room in which a supervisor and several employees are working on a common task. The supervisor voices directions, which the employees follow to complete the task. In the second situation no suitably large room is available, so the employees are distributed into several rooms. In addition to providing the original instructions, the supervisor must cope with the additional complexity of ensuring that the appropriate instructions are provided in the appropriate rooms at the appropriate time.

It has been argued (e.g., [Halpern 1984]) that the "right way" to reason about such protocols is to consider the information or "knowledge" state of the system. In terms of second situation, it is suggested that we attribute knowledge states to the rooms (or their inhabitants), and make decisions accordingly.

The second flavour of motivation is merely a formal abstraction of the first: it is based on the similarity between the second situation above, and some classical logical puzzles about rea-
soning about knowledge. A typical example is that of several children, a subset of which have muddy faces, who are asked to provide arguments as to whether their particular face is muddy. This particular problem is cast in a “phased” protocol [Halpern 1984], where each phase consists of each child reporting the current state of his argument to the others. There are various formalizations within which solutions can be derived (e.g., [Halpern 1984; McCarthy 1978; Nguyen 1986]).

An important point concerning such puzzles is the expressive complexity of the language in which the puzzles are formalized. As there is, as yet, no clear consensus on what the actual problems of distributed protocols are, one is currently left pondering the applicability of existing puzzle solutions to the more general problems of the first flavour. Further analysis of the relationship between the puzzle solutions and the more general problem first requires that we discuss how knowledge is formalized.

2.2 What is knowledge and why is it important?

The study of knowledge and its use has a long and colourful history, but here it suffices to adopt the view that to have knowledge is to record some attitude towards a possible or actual world state. More specifically, having knowledge simply means to record one’s position on some property or relation. Even this vague definition is enough to argue that reasoning about the status of a process in a distributed system is reasoning about what a process knows, e.g., does a process believe that it is awaiting a message, is a process obliged to send a reply, does a process know that another is awaiting its sending of a message, etc.

Most elaborations of this concept rely on the use of mathematical logic, where the semantics associated with a logic provides a specification of the correspondence between a written specification and the concrete or abstract domain to which it applies. Specifically, we can take a logical language, say, the language of first order logic, and regard various axioms as the initial knowledge of a process. We naturally expect that, over time, communication amongst processes
will affect the state of a processes' knowledge. We further expect that our logic should at least allow us to draw conclusions about how the knowledge states of various processes differ, for example, to allow us to determine whether two rooms of employees both know about the same instructions.

Having made the leap from knowledge to logic now allows us to discuss the relationship between a logic, the formalization of a puzzle in logic, and the relationship of that formalization to the more general problems of specifying and debugging distributed systems. For example, the typical specification of the muddy children problem requires a language in which there is an explicit "knows" operator, so that after each phase, a child can record his position on a proposition like "knows(child-1, not-muddy(child-2))." As Halpern and Fagin explain [Halpern 1984], their formalization is closely related to a modal logic called S5. In this case, the question becomes what problems of distributed computing protocols can be solved within a framework that associates an S5 logical theory with each process?

Similar questions can be asked for other combinations of a formal model of knowledge with distributed systems, e.g., [Nguyen and Perry, 1986; Halpern and Moses, 1984; Chandy and Misra, 1985]. Other natural questions include whether or not such a logic should have an explicit representation of time, or even an explicit representation of other processes' reasoning abilities?

Having isolated the question about a knowledge formalism's expressiveness, there remains the original question about how processes communicate information. This question is now further convolved with knowledge-related questions like "Do processes pass on statements about what other processes know?" and "Do processes always report truthfully?" and "Does the synchronization method employed by the distributed protocol affect what a process can know?"

It should be clear that what constitutes an appropriate amalgamation of knowledge and distributed computing is still unclear. In fact the formalization of process knowledge states in terms of logical theories is not wholly undisputed (e.g., see [Rosenschein 1985]). What is clear, how-
ever, is that more is to be learned about those properties of distributed systems that can be analyzed in any framework that attributes some notion of knowledge to each process. So, rather than analyze the applicability of puzzles, we can take a more simple stance to combining the concepts of knowledge and distributed systems. Here we can adopt a very simple formalization of knowledge (e.g., a first order language with monotonic rules of inference) and discuss what in general can be concluded when we assume that processes are imbued with a simple deductive mechanism together with some initial axioms, and communicate only simple propositions in an asynchronous, deterministic manner. The concept of time is captured in the model by defining a partial ordering on events in a computation. Since processes learn as a result of events, knowledge acquisition can also be ordered in this way.

3. Operator Net Model for Distributed Systems

An operator net is a graph consisting of nodes and directed arcs. Each arc of a net denotes a sequence of values taken from some underlying set and each node a function or operator that accepts input sequences from its incoming arcs and produces an output sequence along its outgoing arc. Nodes in an operator net correspond to Lucid operators or functions over infinite sequences. Modal operators (first, next, fby, whenever, etc.) are used to express both recursive and iterative computations in a purely functional way.

Operator net programs intuitively reflect the concept of data flowing through a network. While operator nets are syntactically similar to dataflow nets, they completely separate the operational semantics from the mathematical semantics. With this approach there are several computational methods for achieving the same mathematical meaning.

A general framework for the language of operator nets has been previously described [Ashcroft 1985]. In this section of the paper we provide a summary description of a behavioral semantics for Lucid and operator nets that corresponds to concurrent processing in a distributed environment [Glasgow 1987a].
The set of equations and function definitions for an operator net specify the meaning for
the net as the unique minimal solution for the set of equations. Since these equations are over
cpo's, where operators are continuous, such a solution can be determined using conventional fix-
point theory. This technique is similar to that used in Kahn's "simple language for parallel pro-
cessing" [Kahn 1974, 1977].

The behavioral semantics for operator nets involves a parallel computational method for
determining the solution for a net in a stepwise incremental fashion, where each step of the com-
putation corresponds to a finite approximation to the fixpoint solution for the set of equations.
The solution can be defined as the limit of a chain of such approximations. This approximation,
or history, consists of a set of history sequences corresponding to the arcs of a net. The history
can be considered a record, or snapshot, of all the values of messages transmitted on all channels
of a system up to some particular time.

We define the behavior of an operator net in terms of the computations that can occur in
the net. A computation is a possibly infinite sequence of events, where each event denotes the
results of concurrent processing at one or more of the nodes of the net. An event occurs in some
history and results in a new history that can be defined by applying a continuous function to the
current history and a set of processes.

In order to exploit our model for reasoning in distributed systems, we need to express rela-
tionships among the knowledge states of a system. The concepts of temporal ordering, causality
and dependence between events and messages can also be expressed in the operator net model.
Three fundamental relations have been defined in the theory to describe these concepts.

**Definition.** An event is said to **cause** a message if the value of the message was undefined in the
history in which the event occurs and defined in the history resulting from the event.

**Definition.** An event e1 **precedes** another event e2 in a computation if the history resulting from
e1 approximates the history in which e2 occurs.

Definition. A message m1 depends-on a message m2 if the value of m1 is computed (directly or indirectly) as a function of the value of m2.

The depends-on relation is transitive and allows us to trace the dependency of messages through several nodes of a network.

4. Knowledge-based Specifications

In this section of the paper we describe how the previous model can be extended to handle knowledge-based specifications. This is done by first giving axioms and definitions to describe the behavior of knowledge in the model. We then present a simple example of how knowledge-based specifications can be expressed in the model.

When considering the communication of knowledge, the set of processes for an operator net are partitioned into two categories: reasoning processes and non-reasoning processes. The reasoning processes have deductive capabilities and include the notion of a knowledge set. This set is a record of the formulae which the reasoning process knows. The knowledge set contains the initial axioms for the process as well as knowledge gained from communication or deduction. An assumption made about reasoning processes is that they have at most one input arc. This corresponds to the notion that knowledge is gained sequentially.

Given a history h for an operator net, we denote the knowledge set for a reasoning process p in h as \( K^h_p \). The idea that process p knows formula f in history h is written as \( h \models K^h_p f \), which means that \( f \in K^h_p \).

Associated with each process is a set of axioms that are contained in the knowledge set for the initial history (before communication begins). Each reasoning process p also has an associated closure operator, \( C_p \), that defines the set of formulae that are deducible for p in a given history. The initial knowledge set for a process p is the deductive closure of the initial axioms for the
process. For any process \( p \) with closure operator \( C_p \) and set of formulae \( F \), we denote the deductive closure of \( F \) as \( C_p(F) \). The closure operator may vary for processes as we assume that the processes may have different reasoning abilities.

An event in the extended knowledge model for operator nets can result in a change of knowledge for a reasoning process. Such an event corresponds to communication or transference of knowledge from one process to another. It is assumed that the knowledge sets for a process are monotonic. Thus formulae may be added to the set, but existing formulae cannot be changed or deleted. Given any history \( h \) and set of reasoning processes \( RP \) for an operator net, the knowledge set for a process in \( RP \) can be defined in terms of the set of initial axioms and messages communicated to the process. Assuming that:

- \( A_p \) denotes the initial axiom set for a process \( p \),
- \( C_p \) denotes the closure operator for process \( p \), and
- \( X \) is the input arc for \( p \)

we can define the current knowledge of a reasoning process in a history \( h \) using the following axiom:

**Knowledge Axiom:**

\[
K_p^h = C_p(A_p \cup \{x_0, \ldots, x_k\})
\]

given that \( X^h = x_0, \ldots, x_k \). This axiom states that for all reasoning processes \( p \), the knowledge set of \( p \) is the deductive closure of the initial axioms for \( p \) and the messages communicated to \( p \) on its input channel.

Relations such as *cause*, *precedes* and *depends-on* for knowledge can be defined in the extended model by relating the knowledge set for a process to events that occur in an operator net. Given a reasoning process \( p \) and an event \( e \), we say that the event \( e \) *causes* a formula for \( p \) provided that formula is in the knowledge set for \( p \) in the history resulting from \( e \), but not in the knowledge set for the history in which \( e \) occurs.
Definition. (Kcauses)

\[ \text{If } p \in \text{ RP and event}(h,p,h') \text{ and } f \in (K_p^h - K_p^{h'}) \text{ then } K\text{causes}((h,p,h'),f,p) \]

The relation \( K\text{causes}(e,f,p) \) states that event \( e \) causes formula \( f \) to be added to the knowledge set for process \( p \).

Since events are ordered by time, knowledge acquisition can also be ordered using the events that cause formulae to be added to a knowledge set.

Definition (Kprecedes)

\[ \text{If } p_1,p_2 \in \text{ RP and } K\text{causes}(e_1,f_1,p_1) \text{ and } K\text{causes}(e_2,f_2,p_2) \text{ and precedes}(e_1,e_2) \]
\[ \text{then } K\text{precedes}((f_1,p_1),(f_2,p_2)) \]

The relation \( K\text{precedes}((f_1,p_1),(f_2,p_2)) \) states that process \( p_1 \) knew formula \( f_1 \) before process \( p_2 \) knew formula \( f_2 \). These axioms allow us to reason about the relative order of knowledge acquisition of related processes but do not provide any information about processes that do not communicate (directly or indirectly) with one another.

As well as knowing when knowledge is acquired, it may also useful to know how. For any formula in a knowledge set for a process, it is often possible to determine whether that formula depends on knowledge received from another process. This dependence can be traced through the network so that all processes that contribute to the knowledge of a particular formula may be determined. Whereas in the original model the depends-on relation was defined in terms of messages in the history sequence, when considering knowledge we wish to relate formulas to processes. For a given process \( p_1 \), we say that \( p_1 \) knowing \( f \) depends on a process \( p_2 \) (denoted as \( k\text{depends-on}((f,p_1),p_2) \)) if:

- \( f \) is the result of an event and \( p_2 \) is the process that communicates directly to process \( p_1 \), or
- \( f \) is the value of an element of the input sequence to \( p_1 \) and that
element depends-on an element of the output sequence of p2, or
- f is deduced from a formula that kdepends-on p2.

A more formal definition of Kdepends-on can be found in [Glasgow 1988].

The relations described above allow us to do high level reasoning about knowledge in an asynchronous distributed system. Such systems have no access to global clocks, thus only relative ordering on related events can be determined. This ordering allows us to reason about the order of knowledge acquisition. Thus specifications such as "if process p1 knows f then p2 knew f before p2" can be stated. The concept of a process learning new knowledge is captured by the kcause relation.

The ability to determine "how" knowledge is acquired in the sense of what other processes contribute to a process knowing some formula is useful in specifying properties of secure systems. One definition of security for multilevel secure distributed systems, known as knowledge independence security [Glasgow 1988], states that a process cannot know anything that depends on knowledge of a process at a higher security level. This is stated formally as:

For all histories h,

If level(p1) ≤ level(p2) and h |= Kp1 f

then ~ Kdepends-on((f,p1),p2)

This constraint on a system can be proved or disproved for a particular operator net by tracing the dependency path for formula f.

5. Discussion

We have proposed a model that allows us to reason about the communication of knowledge in a distributed system. Unlike much of the earlier work in this area, we are not concerned with the concept of common knowledge. Instead we are interested in reasoning about the relation between time, causality, dependency and knowledge. The operator net model provides a
means of expressing and an operational interpretation for such knowledge-based specifications.

A reasoning process can be used to represent such objects as deductive databases, inference engines or human users. Examples where we may wish to reason about such processes are in distributed expert systems, robotics or secure distributed systems. In recent work we have shown that the extended operator net model is appropriate for expressing security properties in a simple and intuitive way.

Some of the advantages of the operator net approach that we point out are:

- Operator nets is a language that can be used to express and prove properties of asynchronous distributed systems.

- The equations of operator nets provide a programming and proof technique that share a single coherent structure.

- The extended operator net model can be used to specify knowledge-based specifications, as well as safety and liveness properties, in a distributed system.

- Operator nets provide a graphical illustration that can be used in the design and development stages of distributed systems.

- High-level properties relating knowledge to time and causality can be expressed in the extended model.

A current disadvantage of the operator net model is the inability to specify nondeterministic operators. This is a result of the functionality of the model. A behavioral semantics for nondeterministic operator nets is currently under development.

References

[Ashcroft 1985]

[Chandy 1985]

[Fagin 1986]

[Fisher 1986]

[Glasgow 1987a]


[Glasgow 1987b]


[Goguen 1984]


[Halpern 1985]


[Halpern 1984]


[Kahn 1974]


[Kahn 1977]


[MacEwen 1987]


[McCarthy 1978]


[Nguyen 1986]


[Rosenschein 1985]


[Wadge 1985]