

Formal Modeling of Multi-Agent Systems: An Application of the π -Calculus

Toinette Rorie, Jamika Burge, and Albert C. Esterline

Department of Computer Science/NASA ACE

North Carolina A&T State University

Greensboro, NC 27411

Email: {rorie, jb988330, esterlin}@ncat.edu

Abstract

We show how the π -calculus, a calculus for communicating systems where one can express systems of processes that have changing communication structure, can be used to model aspects of multi-agent systems. The π -calculus is a process algebra. Process algebras are term algebras that can be used to describe communicating concurrent processes, to specify concurrent systems, and to refine specifications down to the level of designs. We present a π -calculus model of certain aspects of the LOGOS multi-agent system being developed at NASA/GSFC as a prototype for an unattended ground operations center in the area of control center automation.

Keywords : π -calculus, multi-agent systems, concurrency, process algebras

INTRODUCTION

Multi-agent systems have become important recently in computer science, especially in artificial intelligence (AI). NASA Goddard Space Flight Center (NASA/GSFC) is currently developing a design for a multi-agent system called LOGOS (Lights Out Ground Operations System) as a prototype for an unattended ground operations center in the area of control center automation [1]. The system will integrate present technologies that exist at NASA/GSFC with emerging technology via the use of autonomous agents.

We show how the π -calculus [2,3], a calculus for communicating systems where one can express systems of processes that have changing communication structure, can be used to model aspects of multi-agent systems. The π -calculus is a process algebra; other examples include CCS [4] and CSP [5]. Process algebras are term algebras that can be used to describe communicating concurrent processes, to specify concurrent systems, and to refine specifications down to the level of designs.

We use the π -calculus to model aspects of the LOGOS agent community. Specifically, we give a π -calculus specification that supports a scenario presented by NASA/GSFC that involves several of the key LOGOS agents. This scenario requires the linkage among the agents to be modified by one agent passing a link to another agent; this link

allows communication with a third agent. The next section of this paper presents a fault scenario for the LOGOS system. The third section summarizes the π -calculus, and the fourth section gives a π -calculus model of aspects of LOGOS involved in the scenario given in the second section. The last section concludes and discusses future work.

A LOGOS FAULT RESOLUTION SCENARIO

There are several LOGOS expert systems that exist [1], such as the GENSAA/GENIE, which operates a satellite on a routine basis and performs routine spacecraft health and safety monitoring. The FIRE agent provides procedures for handling reported mission faults. The DBIFA is an internal agent responsible for storing and retrieving information into and from databases, such as mnemonic values, metadata, anomaly information, and anomaly resolution procedures. The GIFA (GenSAA/IF agent) agent is a reactive agent that responds in a reasonable amount of time to any changes that occur. It is responsible for broadcasting anomaly alerts. The UIFA (User Interface Agent) agent is responsible for responding to an agent request or any other type of data that it receives from the user interface, informing the user of anomalies and schedule changes. It is responsible for having the user contacted (by paging, email, etc.) if he/she is not logged on when an anomaly occurs. The Pager agent is responsible for contacting system personnel or a mission engineer when a situation arises within the LOGOS community that requires human intervention.

We give a π -calculus specification that models behaviors of a part of the LOGOS agent community. The specific behavior we take as an example is a fault resolution scenario. Here the FIRE agent sends messages to the GIFA agent along channel *mvreq*, and the GIFA agent sends messages to the FIRE agent along channel *amsg*. The FIRE agent sends messages to the DBIFA agent along channel *areq* and the DBIFA agent sends messages to the FIRE agent along channel *matproc*. There are two branches to the scenario. In the first branch of the scenario, the FIRE agent is able to resolve the fault. In the second branch of the scenario, the FIRE agent is not able to resolve the fault and requests the direct intervention of a human. In this example, we look only at the second branch. The mnemonic channel names (which will be used in the π -calculus model) and their meanings are as follows: *amsg* (advisory message), *mvreq* (mnemonic value request), *areq* (request for resolution procedures), *matproc* (matching resolution procedures), *humreq* (request for a human to resolve the fault), *humres* (response from the human), *reqhum* (request for a human expert to resolve the fault), *sighum* (signal to contact the human user interface), and *respsig* (response to signal for request for human intervention).

The scenario proceeds as follows.

GIFA to FIRE : Alerts the FIRE agent that a fault has occurred.

FIRE to GIFA : Requests the GIFA agent for the mnemonic values associated with the fault.

GIFA to FIRE : Informs the FIRE agent of the mnemonic values associated with the fault.

FIRE to DBIFA : Requests the DBIFA agent for fault resolution procedures for resolving the fault.

DBIFA to FIRE : Informs the FIRE agent that it was successful at finding matching fault resolution procedures.

In this branch of the scenario, the FIRE agent is unable to resolve the fault and requests the UIFA agent for human assistance in resolving the fault.

FIRE to UIFA : Requests the UIFA agent for human intervention in resolving the fault.

UIFA to FIRE: Responds to the FIRE agent that it will resolve the fault.

FIRE to Pager: Requests that the Pager agent locate the UIFA.

Pager to UIFA: Signals the UIFA agent to resolve a fault.

UIFA to Pager: Commands the Pager agent to stop paging.

UIFA to FIRE: Responds to the FIRE agent that it will resolve the fault.

THE π -CALCULUS

The π -calculus [2] can be used to model a system that consists of processes that interact with each other, and whose environment is constantly changing. The π -calculus is available in two basic styles: the *monadic* calculus [2,3], where exactly one name is communicated at each synchronization, and the *polyadic* calculus [6], where zero or more names are communicated. This presentation mainly follows the monadic π -calculus. The basic concept behind the π -calculus is naming or reference. The names are the primary entities that refer to links or channels, and the processes, sometimes referred to as agents, are the only other kind of entities. We let the letters u, v, w, x, y range over the names. Also, P, Q, R, \dots range over agents or process expressions, of which there are six kinds (corresponding to the six kinds of combinators or operators).

1. A *summation* has the form $\sum_{i \in I} P_i$, where the set I is a finite index set. The agent behaves like one or another of the P_i . The empty summation or inaction, represented by 0 , is the agent that can do nothing. The binary summation is written as $P_1 + P_2$.
2. A *prefix* is of the form $\overline{y}x.P, y(x).P$ or $\tau.P$. “ $\overline{y}x.P$ ” is a *negative prefix*; \overline{y} can be thought of as an output port of an agent which contains it. “ $y(x)$ ” is a *positive prefix*, where y is the input port of an agent; it binds the variable x . At port y the arbitrary name z is input by $y(x).P$, which behaves like $P\{z/x\}$, where $P\{z/x\}$ is the result of substituting z for all free (unbound) occurrences of x in P . We think of the two complementary ports y and \overline{y} as connected by a channel (link), also called y . τ is the *silent action*; $\tau.P$ first performs the silent action and then acts like P .
3. A *composition*, $P_1 | P_2$, is an agent consisting of P_1 and P_2 acting in parallel.
4. A *restriction* $(x)P$ acts like P and prohibits actions at ports x and \overline{x} , with the exception of communication between components of P along the link x . Restriction, like positive prefix, binds variables.
5. $[x = y]P$ is a *match*, where an agent behaves like P if the names x and y are identical.
6. A *defined agent* (with arity n), $A(y_1, \dots, y_n)$, has a unique defining equation, $A(x_1, \dots, x_n) \stackrel{\text{def}}{=} P$, where x_1, \dots, x_n are distinct names and the only names that may occur free in P . $A(y_1, \dots, y_n)$ behaves like $P\{y_1/x_1, \dots, y_n/x_n\}$ for the simultaneous substitution of y_i for all free occurrences of x_i ($1 \leq i \leq n$) in P .

An agent E may perform a communication action (or “transition”) α , corresponding to a prefix, and evolve into another agent E' . This is indicated by the notation $E \xrightarrow{\alpha} E'$. The meanings of the combinators are formally defined by transition rules. Each rule has a conclusion (stated below a line) and premises (above the line). The following rule, where l and \bar{l} are complementary prefixes, is of particular interest.

$$\frac{E \xrightarrow{l} E', F \xrightarrow{\bar{l}} F'}{E | F \xrightarrow{\tau} E' | F'}$$

This indicates communication between E and F , resulting in a silent action, τ , whereby generally a value is communicated between E (now E') and F (now F'). For example,

$$\bar{y}x.P | y(z).Q \xrightarrow{\tau} P | Q\{x/z\}$$

so that after this communication all free occurrences of z in Q are replaced by whatever value x had in $\bar{y}x.P$. Of particular interest are those cases where we use restriction to internalize ports so that they cannot be observed from the outside, forcing communication actions to match their complements. For example, $(y)(\bar{y}(x).P | y(z).Q)$ can perform only a τ action where x is communicated along channel y to become the value of z in Q .

Consider next a simple case of restructuring a system of agents. We begin with

$$(y)(m)(y(x).\bar{x}z.0 | \bar{y}m.0 | m(u).P)$$

with three parallel components. Because of the restrictions on channels y and m , the only action that can occur in this system is a τ action by which the second agent sends m on channel y to the first agent. We then have $(m)(\bar{m}z.0 | m(u).P)$. The second agent has disappeared, being reduced to the do-nothing process 0. Also, the restriction (y) has been dropped since y no longer occurs in the expression. The second agent communicated to the first a channel, m , and the only action the resulting system can perform is a τ action by which z is communicated along this channel, resulting in $P\{z/u\}$.

There are two notions of equivalence in CCS, which are strong equivalence and observational equivalence. The former is defined in terms of strong bisimulation, which holds between two processes if they can perform the same actions yielding processes that are equivalent. The latter, however, is defined in terms of weak bisimulation, which allows two processes to differ due to the presence of τ actions. Strong equivalence is a congruence relation: one term can be replaced by a strongly equivalent term within a larger term while preserving the denotation of the larger term. Statements of congruence are equational laws that support equational reasoning. The π -calculus supports similar notions of bisimulation, equivalence, and congruence, but with the subtlety that the substitution of free names does not preserve strong bisimulation.

We can often prove that a system satisfies its specification through the use of the notions of equivalence and the defining notions of bisimulation as follows. We express concisely the desired behavior (specification) as a process expression. A process expression can be used to describe the designed system, one with components corresponding to the components in the design. One can show that the states of the specification and the states of the design are equivalent if one can find a bisimulation between the two. Often, however, we begin, not with a complete specification of a system's behavior, but with only certain conditions on its behavior. Such conditions can be expressed in the process (or Hennessey-Milner) logic L , a modal logic based on the notion of possible action. The Mobility Workbench [7,8] is an automated tool used for manipulating and analyzing concurrent systems. It allows one to define behaviors presented in the syntax of the polyadic π -calculus, simulate the behavior of an agent, and check whether a given process satisfies a specification.

THE π -CALCULUS SPECIFICATION FOR THE SCENARIO

We give the π -calculus specification that supports the branch of our scenario where the FIRE agent is not able to resolve a fault and requests that a human expert resolve the fault. This specification is presented in Figure 1. Consider, for example, definition (1) of Figure 1.

$$GIFA \stackrel{def}{=} \overline{ams}g(124000).GIFA1 \\ + \overline{ams}g(246000).GIFA1$$

Here the GIFA agent sends the FIRE agent a message alerting it that a fault has occurred. The definition also involves the summation operator, allowing the GIFA agent the alternative to output one of two different types of faults. In definition (2), the FIRE agent receives a message from the GIFA agent indicating that a fault has occurred.

Next, in definitions (3)-(4) the FIRE agent sends a message to the GIFA agent requesting *tmon* values, based on what it receives in definition (2). For example,

$$FIRE1 = \overline{mvreq}(tm12st). \overline{mvreq}(tm13st).FIRE4 \\ FIRE2 = \overline{mvreq}(tm14st). \overline{mvreq}(tm15st).FIRE3$$

In definition (5) we make use of the match operator:

$$GIFA1 = mvreq(x).mvreq(y).([x=tm12st][y=tm13st])GIFA2 \\ + [x=tm14st][y=tm15st])GIFA3$$

Here, if x and y (the *tmon* values requested by FIRE) are $tm12st$ and $tm13st$, then the GIFA agent will respond with the values 45 and 52; if x and y are $tm14st$ and $tm15st$, GIFA will respond with the values 50 and 52. In definitions (6)-(8) we show the GIFA

(1) $GIFA \stackrel{def}{=}$	$\overline{ams\!g}(124000).GIFA1 + \overline{ams\!g}(246000).GIFA1$
(2) $FIRE \stackrel{def}{=}$	$\overline{ams\!g}(x).([x=124000].FIRE1 + [x=246000].FIRE2$
(3) $FIRE1 \stackrel{def}{=}$	$\overline{mvreq}(tm12st). \overline{mvreq}(tm13st).FIRE4$
(4) $FIRE2 \stackrel{def}{=}$	$\overline{mvreq}(tm14st). \overline{mvreq}(tm15st).FIRE3$
(5) $GIFA1 \stackrel{def}{=}$	$\overline{mvreq}(x).\overline{mvreq}(y).([x=tm12st][y=tm13st])GIFA2$ $+ [x=tm14st][y=tm15st])GIFA3$
(6) $GIFA2 \stackrel{def}{=}$	$\overline{ams\!g}(45). \overline{ams\!g}(52).GIFA$
(7) $GIFA3 \stackrel{def}{=}$	$\overline{ams\!g}(50). \overline{ams\!g}(52).GIFA$
(8) $FIRE3 \stackrel{def}{=}$	$\overline{ams\!g}(x).\overline{ams\!g}(y).FIRE4$
(9) $FIRE4 \stackrel{def}{=}$	$\overline{areq}(\text{anomaly resolution procedures}).FIRE5$
(10) $DBIFA \stackrel{def}{=}$	$\overline{areq}(x).DBIFA1$
(11) $DBIFA1 \stackrel{def}{=}$	$\overline{matproc}(\text{fault resolution procedures}).DBIFA + \overline{matproc}(\text{no fault resolution procedures}). DBIFA2$
(12) $FIRE5 \stackrel{def}{=}$	$\overline{matproc}(x).FIRE7 + \overline{matproc}(y).FIRE6$
(13) $FIRE6 \stackrel{def}{=}$	$\overline{humreq}.FIRE7$
(14) $UIFA \stackrel{def}{=}$	$\overline{humreq}.UIFA1$
(15) $UIFA1 \stackrel{def}{=}$	$\overline{humres}(\text{no user}).UIFA2 + \overline{humres}(\text{user}). \overline{humres}(\text{resolve fault}).UIFA$
(16) $FIRE7 \stackrel{def}{=}$	$\overline{humres}(resp).([resp = nouser]FIRE + [resp = user]FIRE8$
(17) $FIRE8 \stackrel{def}{=}$	$\overline{humreq}(\text{sighum}). \overline{humreq}(\text{respsig}). \overline{reqhum}(\text{sighum}).$ $\overline{reqhum}(\text{respsig}).FIRE9$
(18) $UIFA2 \stackrel{def}{=}$	$\overline{humres}(x). \overline{humres}(y).UIFA3(x, y)$
(19) $Pager(x, y) \stackrel{def}{=}$	$\overline{reqhum}(x).\overline{reqhum}(y).Pager1(x, y)$
(20) $Pager1(x, y) \stackrel{def}{=}$	$\overline{x.y.Pager}$
(21) $UIFA3(x, y) \stackrel{def}{=}$	$x. \overline{y}.UIFA4(x, y)$
(22) $UIFA4(x, y) \stackrel{def}{=}$	$\overline{humres}(\text{user}). \overline{humres}(\text{resolve fault}).UIFA$
(23) $FIRE9 \stackrel{def}{=}$	$\overline{humres}(resp1). \overline{Humres}(resp2).FIRE$

Figure 1. The π -calculus specification supporting the scenario

agent sending the FIRE agent the requested *tmon* values, and the FIRE agent actually receiving these value.

$$\begin{aligned}
GIFA2 &= \overline{ams\!g}(45). \overline{ams\!g}(52).GIFA \\
GIFA3 &= \overline{ams\!g}(50). \overline{ams\!g}(52).GIFA \\
FIRE3 &= \overline{ams\!g}(x).\overline{ams\!g}(y).FIRE4
\end{aligned}$$

Our next example shows the FIRE agent requesting the DBIFA agent for matching resolution procedures for resolving a fault, and the DBIFA agent's response with successfully finding these procedures. This is shown in definitions (9)-(12).

$$\begin{aligned}
FIRE4 &= \overline{areq}(\text{anomaly resolution procedures}).FIRE5 \\
DBIFA &= \overline{areq}(x).DBIFA1 \\
DBIFA1 &= \overline{matproc}(\text{fault resolution procedures}).DBIFA \\
&\quad + \overline{matproc}(\text{no fault resolution procedures}).DBIFA2 \\
FIRE5 &= \overline{matproc}(x).FIRE7 + \overline{matproc}(y).FIRE6
\end{aligned}$$

In definitions (13)-(16) the FIRE agent requests assistance in resolving a fault from the UIFA agent, and the UIFA agent responds with either a message indicating that there is no user available or a message indicating that there is an available user and that the fault will be resolved. If there is no user available, it behaves like *FIRE8*.

In definitions (17)-(21) we give an example using perhaps the most powerful feature in the π -calculus. This feature allows us to restructure our system, by passing links between agents within our system. Consider the following:

$$\begin{aligned}
FIRE8 &= \overline{humreq}(\text{sighum}). \overline{humreq}(\text{respsig}). \overline{reqhum}(\text{sighum}). \\
&\quad \overline{reqhum}(\text{respsig}).FIRE9 \\
UIFA2 &= \overline{humres}(x). \overline{humres}(y).UIFA3(x, y) \\
Pager &= \overline{reqhum}(x). \overline{reqhum}(y).Pager1(x, y) \\
Pager1(x, y) &= \overline{x}.y.Pager \\
UIFA3(x, y) &= x. \overline{y}.UIFA4(x, y)
\end{aligned}$$

Here the FIRE agent sends the links *sighum* and *respsig* to the UIFA agent on link *humreq* and to the Pager agent on link *reqhum* to provide a means for communication between the UIFA agent and the Pager agent. The Pager agent can then send a message to the UIFA agent on link *sighum* requesting that it resolve a fault upon the FIRE agent's request. Once the UIFA agent receives the Pager agent's request, it can send a message to it requesting that it stop paging, indicating that the message was received. Finally, in definitions (22)-(23) the UIFA agent will inform the FIRE agent that the fault will be resolved.

Conclusion

We have shown how a multi-agent system – specifically, part of the NASA/GSFC LOGOS agent community – can be modeled in the π -calculus. In other work [9], we have translated this specification into the language required by the Mobility Workbench and have used the Mobility Workbench to establish that certain properties expressed in the logic *L* hold of this system. We have also refined the specification somewhat and

used the Mobility Workbench to verify certain bisimulations. Our work shows promise for a general method of formally modeling multi-agent systems in general.

The general advantages of a formal model are twofold. First of all, one can formally check the correctness of the specification of a multi-agent system, that is, one can establish that it is internally consistent and that it reflects the intended system behavior. Secondly, a formal model supports formal validation of the refinement of a specification into a design. A process algebraic specification of a multi-agent system has the advantage over a logical specification that it can be refined into a design presenting the structure of the intended system. A π -calculus model in particular can specify changing structure. Finally, a model presented in the π -calculus (or any process algebra) has the advantage over a model using Petri nets, Statecharts, or other automata-like formalisms that it is compositional, that is, two or more π -calculus model can be directly combined using the given combinators to produce a more extensive model.

In the future, we intend to formulate general aspects of multi-agent architectures in the π -calculus. We have already begun to formulate principles of BDI (Belief, Desire and Intention) architectures [10] in the π -calculus. A challenge here is to represent persistent features (such as belief and intention), but this is analogous to models of computer memory already done in the π -calculus. We have also begun to develop semantics of agent communication languages [11] in the π -calculus. This is analogous to the formal semantics of concurrent programming languages given in the π -calculus. The central notion for us is that of a speech act, which involves discourse patterns that structure sequences of communication actions.

REFERENCES CITED

1. LOGOS Development Team, Automation Technology Section, GSFC, "LOGOS Requirements & Design Document", NASA/Goddard Space Flight Center, Greenbelt, Md. (1997), <http://groucho.gsfc.nasa.gov/agents/products.html>.
2. R. Milner, J. Parrow, and D. Walker, "A Calculus of Mobile Processes, Parts I and II". *Journal of Information and Computation*, Vol. 100 (1992), 1-77.
3. R. Milner, *Communication and Concurrency*, Hertfordshire, UK: Prentice Hall International (1989).
4. Hoare, C.A.R., *Communicating Sequential Processes*, Prentice Hall (1985).
5. R. Milner, "The Polyadic π -Calculus: a Tutorial, in F. L. Bauer, W. Brauer, and H. Schwichtenberg (eds.), *Logic and Algebra for Specification*, Berlin: Springer-Verlag, (1993), 203-246.
6. B. Victor and F. Moller, "*The Mobility Workbench - A Tool for the π -Calculus*". Technical Report DoCS 94/45, Department of Computer Science, Uppsala University, Uppsala, Sweden (1994).
7. T. Rorie, "Formal Modeling of Multi-Agent Systems Using the π -Calculus", Master's Thesis, Department of Computer Science, North Carolina A&T State University, Greensboro, NC (1998).
8. A. Haddadi and K. Sundermeyer, "Belief-Desire-Intention Agent Architectures", in G. O'Hare and N. R. Jennings (eds.), *Foundations of Distributed Artificial Intelligence*, NY: Wiley (1996), 169-185.
9. Y. Labrou and T. Finin, "Semantics and Conservations for an Agent Communication Language" in M. Huhns & M. Singh (eds.), *Readings in Software Agents*, San Francisco: M. Kaufmann (1998), 235-42.