CHAPTER 1

REQUIREMENTS OF AN INTEGRATED FORMAL METHOD FOR INTELLIGENT SWARMS

M.G. Hinchey[1], J.L. Rash[2], C.A. Rouff[3], W.F. Truszkowski[2], and A.K.C.S. Vanderbilt[4]

[1]Lero-the Irish Software Engineering Research Centre
[2]NASA Goddard Space Flight Center (*Emeritus*)
[3]Lockheed Martin, Advanced Technology Laboratories
[4]Vanderbilt Consulting

1.1 INTRODUCTION

NASA is investigating new paradigms for future space exploration, heavily focused on the (still) emerging technologies of autonomous and autonomic systems [47, 48, 49]. Missions that rely on multiple, smaller, collaborating spacecraft, analogous to swarms in nature, are being investigated to supplement and complement traditional missions that rely on one large spacecraft [16]. The small spacecraft in such missions would each be able to operate on their own to accomplish a part of a mission, but would need to interact and exchange information with the other spacecraft to successfully execute the mission.

This new systems paradigm offers several advantages:

- the ability to explore environments and regions in space where traditional craft would be impractical,

- greater mission redundancy (and, consequently, greater protection of assets), and
- reduced costs and risk,

to name but a few. Examples of concept swarm missions include

- the use of autonomous unmanned air vehicles (UAVs) flying approximately one meter above the surface of Mars, which will cover as much of the surface of Mars in seconds as the now-famous Mars rovers did in their entire time on the planet;
- the use of armies of tetrahedral walkers to explore the surface of Mars and the Moon [15];
- constellations of satellites flying in formation; and
- the use of miniaturized pico-class spacecraft to explore the asteroid belt, where heretofore it has been impossible to send exploration craft without the unacceptably high likelihood of loss [16].

However, these new approaches to exploration simultaneously pose many challenges. These missions will be unmanned and highly autonomous. Many of them will be sent to parts of the solar system where manned missions are simply not possible within the foreseeable limits of technology, and to destinations where the round-trip delay for communications to spacecraft exceeds 40 minutes, meaning that the decisions on responses to exploration opportunities as well as problems and undesirable situations must be made *in situ* rather than from ground control on Earth.

The degree of autonomy and intelligence necessary for such missions would require an unprecedented amount of testing of any developed (software and hardware) systems. Furthermore, with learning and autonomic properties—such as self-optimizing and self-healing—emergent behavior patterns simply cannot be fully predicted. Consequently, these missions will be orders of magnitudes more complex than traditional single-spacecraft missions, and verifying these new types of missions will be infeasible using traditional techniques. The authors believe that formal specification techniques and formal verification will play important roles in the future development of NASA space exploration missions. Formal methods would enable software assurance and proof of correctness of the behavior of swarms, even when (within certain bounds) this behavior is emergent (as a result of composing a large number of interacting entities, producing behavior that, absent extraordinary design and verification measures, was not foreseen). Formal models derived may also be used as the basis for automating the generation of much of the code for the mission [25].

To address the challenge in verifying these types of missions, a NASA project, Formal Approaches to Swarm Technology (FAST), investigated formal methods for use in such missions [34, 35, 36, 37, 38, 41, 42]. A NASA concept mission, Autonomous Nano-Technology Swarm (ANTS), was used as an example mission to be specified and verified [15, 16, 17]. An effective formal method for use on the ANTS mission would have to be able to predict the emergent behavior of 1000 agents operating as a swarm, as well as the behavior of the individual agents. Crucial to the mission would be autonomic properties and the ability to modify operations autonomously to reflect the changing conditions and goals of the mission.

This chapter gives an overview of swarm technologies and the ANTS swarm-based mission concept, presents the results of an evaluation of a number of formal methods for verifying swarm-based missions, and proposes an integrated formal method for verifying swarm-based systems.

1.2 SWARM TECHNOLOGIES

Swarms [3, 4] consist of a large number of simple agents that have local interactions with each other and the environment. There is no central controller directing the swarm and no one agent has a global view; the simple interactions give rise to "emergent behaviors" and dynamic self-organization of the swarm. Emergent behavior is observed in insects and flocks of birds. Bonabeau et al. [8], who studied self-organization in social insects, state that "complex collective behaviors may emerge from interactions among individuals that exhibit simple behaviors" and describe emergent behavior as "a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components." The emergent behavior is sometimes referred to as the macroscopic behavior and the individual behavior and local interactions as the microscopic behavior.

Agent swarms are often used as a modeling technique and as a tool to study complex systems [22]. In swarm simulations, a group of interacting agents [50] (often homogeneous or near-homogeneous) is studied relative to their emergent behavior. Swarm simulations have supported the study of flocks of birds [11, 33], business and economics [31], and ecological systems [43]. In swarm simulations, each of the agents is given certain parameters that it tries to maximize. In terms of bird swarms, each bird tries to find another bird to fly with, and then will fly off to one side and slightly higher to reduce its drag, and eventually the birds form flocks. Swarms are also being investigated for use in applications such as telephone switching, network routing, data categorizing, and shortest path optimizations [7].

In intelligent swarms, the individual members of a swarm exhibit intelligence [6, 7]. With intelligent swarms, members may be heterogeneous or homogeneous. Due to their differing environments, swarm members, even if initially they are homogeneous, may learn different things and develop different goals, and thereby the swarm may become heterogeneous. A swarm that is homogeneous from the start (such as the NASA concept mission described below) will possess different capabilities as well as a possible social structure. This makes verifying such systems even more difficult, since the swarms are no longer made up of homogeneous members with limited intelligence and communications.

The emergent behavior of swarms can be unpredictable. Though swarm behaviors are the combination of often simple individual behaviors, they can, when aggregated,

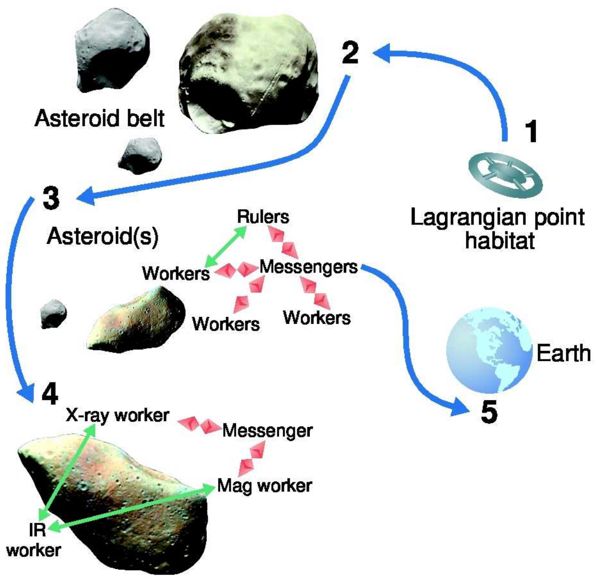

Figure 1.1 ANTS Mission Concept

form complex and often unexpected behaviors. Verifying *intelligent* swarms will be even more difficult, not only due to the greater complexity of each member, but also due to the complex interaction of a large number of intelligent elements. Intelligent swarms possess a huge state space, and since the elements may be in "learning mode", the behavior of the individual elements and the emergent behavior of the swarm may be constantly changing and difficult to predict. Accurately predicting these behaviors, however, will be very important to mission developers in assuring that these missions operate safely and as planned.

The remainder of this section gives an overview of the NASA ANTS concept swarm-based mission and the difficulty in specifying such a mission. We are using the ANTS mission as an example test-bed and case study, for the purpose of evaluating multiple formal methods in the specification, validation, and verification of swarm-based missions.

1.2.1 ANTS Mission Overview

The Autonomous Nano-Technology Swarm (ANTS) concept mission [15, 16, 17] would involve the launch of a swarm of autonomous pico-class (approximately 1kg) spacecraft that would explore the asteroid belt for asteroids with certain scientific characteristics. Figure 1.1 gives an overview of the ANTS mission [48]. In this mission, a transport ship, launched from Earth, will travel to a point in space where net gravitational forces on small objects (such as pico-class spacecraft) are negligible (a solar-system Lagrangian point). From this point, 1000 spacecraft, that have been manufactured en route from Earth, will be launched into the asteroid belt. Because of their small size, each spacecraft will carry just one specialized instrument for collecting a specific type of data from asteroids in the belt.

To implement this mission, a heuristic approach is being considered that provides a social structure to the swarm, with hierarchical behavior analogous to colonies or swarms of insects, and with some spacecraft directing others. Artificial-intelligence technologies such as genetic algorithms, neural nets, fuzzy logic, and on-board planners are being investigated to assist the mission to maintain a high level of autonomy. Crucial to the mission will be the ability to modify its operations autonomously to reflect the changing nature of the mission and the distance and low bandwidth of communications back to Earth. Approximately 80 percent of the spacecraft will be workers that will carry the specialized instruments (e.g., a magnetometer, a sensor in the x-ray, gamma-ray, or visible/IR band, or a neutral mass spectrometer) and that will obtain specific types of data. Some will be coordinators (called rulers or leaders) that have rules that will decide the types of asteroids and data the mission is interested in and that will coordinate the efforts of the workers. The third type of spacecraft are messengers that will coordinate communication between the rulers and workers, and communications with the mission control center on Earth (also known as "ground control").

Many things can happen when an ANTS team encounters an asteroid. A spacecraft can do a flyby and do opportunistic observations. The flyby can be used to first determine whether the asteroid is of interest before sending an entire team to the asteroid, and whether, due to the nature of the instrument on the spacecraft, only a flyby is necessary. If the asteroid is of interest, an imaging spacecraft will be sent to the asteroid to ascertain its exact dimensions and features and to create a rough model to be used by other spacecraft for maneuvering around the asteroid. Other teams of spacecraft will then coordinate to finish studying and mapping the asteroid to form a complete model.

1.2.2 Specifying and Verifying ANTS

The above is a very simplified description of the ANTS mission. For a more detailed description, the interested reader is directed to [40, 48], or to the ANTS web-site. As can be seen from the brief exposition above, ANTS is a highly complex system that poses many significant challenges. Not least amongst these are the complex interactions between heterogeneous components, the need for continuous re-planning, re-configuration and re-optimization, the need for autonomous operation without intervention from Earth, and the need for assurance of the correct operation of the mission.

In missions such as ANTS that will be highly autonomous and out of contact with ground control for extended periods of time, errors in the software may not be observable or correctable after launch. Consequently, a high level of assurance is necessary for these missions before they are launched. Testing of space exploration systems is done through simulations, since it would be impractical or impossible to test them in their end environment. Although these simulations are of very high quality, often very small errors get through and can result in the loss of an entire mission, as is thought to have happened with the Mars Polar Lander Mission [5], where the absence of one line of code in the flight software may have resulted in the loss of the entire mission. In the report on the loss of the Mars Polar Lander [12], it is stated that

> it is important to recognize that space missions are a "one strike and you are out" activity. Thousands of functions can be correctly performed and one mistake can be mission catastrophic.

and

> A thorough test and verification program is essential for mission success.

Complex missions such as ANTS exacerbate the difficulty of finding errors, and will require new mission verification methods to provide the level of software assurance that NASA requires to reduce risks to an acceptable level. Many of the ANTS behaviors, including those that produce race conditions, for example, are time-based and only occur when processes send or receive data at particular times or in a particular sequence, or after learning occurs. Errors under such conditions can rarely be found by inputting sample data and checking for correct results. To find these errors through testing, the software processes involved would have to be executed in all possible combinations of states (state space) that the processes could collectively be in. Because the state space is exponential (and sometimes factorial) to the number of states, it becomes essentially untestable with a relatively small number of processes. Traditionally, to get around the state explosion problem, testers have artificially reduced the number of states of the system and approximated the underlying software using models. This reduces the fidelity of the model and can mask potential errors.

A significant issue for specifying (and verifying) swarm behavior is support for analysis of and identification of emergent behavior. The idea of emergence is well known from biology, economics, and other scientific areas. It is also prominent in computer science and engineering, but the concept is not so well understood by computer scientists and engineers, although they encounter it regularly. Emergent behavior has been described as "system behavior that is more complex than the behavior of the individual components ... often in ways not intended by the original designers [32]." This means that when interacting components of a system whose individual behavior is well understood are combined within a single environment, they can demonstrate behavior that can be unforeseen or not explained from the behavior of the individual components. The corollary of this is that making changes to components of a system of systems, or replacing a sub-system within a system of systems, may often have unforeseen, unexpected, and completely unexplained ramifications for both overall system behavior and the behavior of other subsystems.

A formal specification for swarm-based missions will need to be able to track the goals of the mission as they change and be able to modify the model of the universe as new data comes in. The formal specification will also need to allow for specification of the decision-making process to aid in the decision as to which instruments will be needed, at what location, with what goals, etc. Once written, the formal specification must be usable to prove properties of the system and check for particular types of errors, and be usable as input to a model checker. The formal method must also be able to track the models of the leaders, and it must allow for decisions to be made as to when the data collected have met the goals. To accomplish all of this, an integration of four identified formal methods [34, 37, 38, 41, 42] seems to be the best approach for verifying cooperating swarm-based systems.

1.3 NASA *FORMAL APPROACHES TO SWARM TECHNOLOGY* PROJECT

The FAST project identified several important attributes needed in a formal approach for verifying swarm-based systems [35, 38] and surveyed a wide range of formal methods and formal techniques [34, 41, 42] to determine whether existing formal methods, or a combination of existing methods, could be suitable for specifying and verifying swarm-based missions and their emergent behavior. Various methods were surveyed based on a small number of criteria that were determined to be important in their application to intelligent swarms. These included:

- support for concurrency and real-time constraints;
- formal basis;
- (existing) tool support;
- past experience in application to agent-based or swarm-based systems; and
- algorithm support.

A large number of formal methods that support the specification of one of, but not both, concurrent behavior and algorithmic behavior were identified. In addition, there were a large number of integrated formal methods that have been developed over recent years with the goal of supporting the specification of both concurrency and algorithms. Based on the results of the survey, four formal methods were selected to be used for a sample specification of part of the ANTS mission. These methods were:

- Communicating Sequential Processes (CSP) [24, 26, 27],
- Weighted Synchronous Calculus of Communicating Systems (WSCCS) [46],
- X-Machines [28, 29, 30], and
- Unity Logic [13].

CSP was chosen as a baseline specification method because the team had significant experience and success [39, 23] in specifying agent-based systems with CSP. WSCCS and X-Machines were chosen because they had already been used for specifying emergent behavior by others, with some success. Unity Logic was also chosen because it had been successfully used for specifying concurrent systems and was logic-based, in contrast with the other methods. Integrating the memory and transition function aspects of X-Machines with the priority and probability aspects of WSCCS and other methods may produce a specification method that will allow all the necessary aspects for specifying emergent behavior in the ANTS mission and other swarm-based systems. In addition, available tools supported these formal methods [38].

The approach being taken to integrate the formal methods was the viewpoint-integration [18, 45] method. In this type of formal-methods integration, the base formalisms of the methods are maintained, and relationships between the formalisms are developed to reflect the new formal method. This approach preserves the strength of the underlying methods, allows a seamless specification of the ANTS mission, and allows the development of support tools using existing semantics of the methods.

Specification by viewpoints is also widely advocated as a useful method for dealing with the complexity of specifying large systems. Each view specification describes an aspect (rather than a component or module) of the system, using a language most suited to that view. A consequence of using this approach is that specifications of the same or related elements often appear in different views and must adequately cross reference each other.

1.4 INTEGRATED SWARM FORMAL METHOD

The majority of formal notations currently available were developed in the 1970s and 1980s and reflect the types of distributed systems being developed at that time. Current distributed systems are evolving and may not be specifiable the same way past systems have been specified [14]. Consequently, many researchers and practitioners are combining formal methods into integrated (hybrid) approaches to address the new features of distributed systems (e.g., mobile agents, swarms, and emergent behavior).

Integrated approaches have been very popular in specifying concurrent and agent-based systems. Integrated approaches often combine a process algebra or logic-based approach with a model-based approach. The process algebra or logic-based approach allows for easy specification of concurrent systems, while the model-based approach provides strength in specifying the algorithmic part of a system. The following is a partial list of integrated approaches that have been used for specifying concurrent and agent-based systems:

- Communicating X-Machines [2],
- CSP-OZ (a combination of CSP and Object-Z [19]),
- Object-Z and Statecharts [10],
- Timed Communicating Object Z [20],
- Temporal B [9],
- Timed CSP [44],

- Temporal Petri Nets (Temporal Logic and Petri Nets) [1], and
- ZCCS (a combination of Z and CCS [21]).

To illustrate the integration of CSP, WSCCS, X-Machines, and Unity Logic, examples of different views of the ANTS mission are given below. These views show how they can cross reference each other and how each can provide a different view of the ANTS mission. CSP provides the inter-process communication view, X-Machines provides the state machine and memory views, WSCCS provides the probability and priority views, and Unity Logic the logic views. Variables that each of them references, such as goals and models, will have additional notation or highlighting to indicate cross references to specifications by other views. For a longer specification, color coding could be used. Since this is only a sample specification for illustration purposes, only bold highlighting will be used. The following sections give examples of these views based on the specifications presented in [37, 38, 42].

1.4.1 Communicating Sequential Processes View

The following is the top-level specification of the ANTS mission in CSP:

$$ANTS_{goals} = Leader_{i,\mathbf{l\_goals}} || Messenger_{j,\mathbf{m\_goals}} || Worker_{k,\mathbf{w\_goals}},$$

$$1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq p.$$

where m is the number of leader spacecraft, n the number of messenger spacecraft, and p the number of worker spacecraft. This specification shows that the ANTS mission starts, or is initialized, with a set of goals given to it by the principal investigator and some of these goals are given to the leader (while some of these goals may not be given to the leader because the goals are ground-based or not applicable to the leader). In addition to goals, each of the spacecraft is given a name (in this case in the form of a number) so that it can identify itself when communicating with other ANTS spacecraft and the Earth.

For the viewpoint integration, the goals of ANTS will also be specified by the X-Machine view of ANTS. The goals are highlighted here to indicate they are also referenced in other views. The following gives the specifications for the communications of a leader with indications of where the specification and the variables are used in other views.

1.4.1.1 Leader Specification The leader-spacecraft specification consists of the communications process and the intelligence process:

$$Leader_i = LEADER\_COM_i, \{\} || LEADER\_INTELLIGENCE_{i,\mathbf{goals},\mathbf{model}}$$

The communications process, LEADER_COM, specifies the behavior of the spacecraft as it relates to communicating with the other spacecraft and Earth, and specifies a protocol between the spacecraft. The second process, LEADER_INTELLIGENCE, constitutes the intelligence of the leader; it maintains the goals and the models of the spacecraft and its environment, and specifies how they are modified during operations. For each of the above processes, one of its parameters is a spacecraft-group identifier, and other parameters represent sets that store conversations (empty at startup), goals, and models. The models and goals are specified in the X-Machine specifications, and the reasoning part of the LEADER_INTELLIGENCE process is specified in the WSCCS specifications.

The following gives the specifications for the communication process of the leader.

1.4.1.2 Leader Communication Specification A leader spacecraft may receive messages from another leader, a worker, a messenger, or Earth. A message from any other sender constitutes an error condition and the message is returned (it is assumed that there is a mechanism that keeps an error message from being returned as an error message, thus causing an endless loop). It is assumed that messages that are relayed through a messenger from another spacecraft are marked as being sent from the original sender and not the messenger. Leaders also maintain a set ("*conv*") of persistent messages to keep track of the current state of the conversations. Requests may be made to other spacecraft for status, to move to a new location, change goals, or return specific data. Requests may also be sent to other spacecraft for similar actions.

Leader communication is specified via WSCCS, X-Machines, and Unity Logic, and when a communication occurs, a change of state is executed from either the reasoning state or the processing state. Also, if, while communicating, any of the actions listed in the WSCCS, X-Machine, or Unity Logic specification occurs, then a change is executed from the communicating state to either the reasoning state or the processing state. Again, highlighting is used to indicate specifications in other views. The rendezvous for each of the below CSP specifications also acts as a transition to the WSCCS, X-Machine, or Unity Logic finite state machine.

The following is the top level specification of the leader communication.

$$
\begin{aligned}
&LEADER\_COM_{i,conv} = leader.in?msg \rightarrow \\
&\quad \text{case } \mathbf{LEADER\_MESSAGE}_{i,conv,msg} \\
&\qquad\quad \text{if } sender(msg) = LEADER \\
&\qquad \mathbf{MESSENGER\_MESSAGE}_{i,conv,msg} \\
&\qquad\quad \text{if } sender(msg) = MESSENGER \\
&\qquad \mathbf{WORKER\_MESSAGE}_{i,conv,msg} \\
&\qquad\quad \text{if } sender(msg) = WORKER \\
&\qquad \mathbf{EARTH\_MESSAGE}_{i,conv,msg} \\
&\qquad\quad \text{if } sender(msg) = EARTH \\
&\qquad \mathbf{ERROR\_MESSAGE}_{i,conv,msg} \\
&\qquad\quad \text{otherwise}
\end{aligned}
$$

The above shows the messages from other spacecraft types that a leader may receive. The WSCCS, X-Machine, and Unity Logic views define the above "LEADER-_MESSAGE", etc. as actions that change the state of the leader from either Reasoning or Processing to Communicating. The specification in Unity Logic also has predicates that express the conditions for passing messages. The above statements are highlighted to indicate a link to another view.

The following processes further describe the messages that may be received from other leaders. Messages sent from another leader may be one of two types: request or informational. Requests may be issued for, among other things, information on the leader's model or goals, for resources (e.g., more workers), or for status. Messages may also be informational and may contain data concerning new goals or new information for the agent's model (pertaining, e.g., to a new discovery or a message from Earth). This information needs to be examined by the intelligence process and the model process to determine whether any updates to the goals or model need to be made. Since X-machines also specify the goals and models, references to each of them have links to the X-machine specification.

```
$\mathbf{LEADER\_MESSAGE}_{i,conv} =$
  case $LEADER\_INFORMATION_{i,conv,msg}$
      if $content(msg) = information$
    $LEADER\_REQUESTS_{i,conv,msg}$
      if $content(msg) = request$
    $LEADER\_RECEIVE_{i,conv,msg}$
      if $content(msg) = reply\_to\_request$
    $ERROR\_MESSAGE_{i,conv,msg}$
      otherwise
```

Further specification of each of the above sub-processes follows.

```
$LEADER\_INFORMATION_{i,conv} =$
  $\mathbf{leader\_model_i}\,!(NEW\_INFO, msg)$
  $\rightarrow \mathbf{goals\_channel_i}\,!(NEW\_INFO, msg)$
  $\rightarrow LEADER\_COM_{i,conv}$
```

If the message is new information, then that information has to be sent to the deliberative part of the agent to check whether the goals should be updated, as well as to the model part to check whether any of the information requires updates to the model. Again, since the model and goals are defined in other views (X-Machine) and the communications causes state changes in the WSCCS, X-Machine, and Unity Logic specifications, the channels above are highlighted.

```
$LEADER\_REQUESTS_{i,conv,msg} =$
  case $LEADER\_STATUS\_REQ$
      if $content(msg) = status\_request$
    $LEADER\_INFO\_REQ_{i,conv,msg}$
      if $content(msg) = info\_request$
    $LEADER\_RESOURCE\_REQ_{i,conv,msg}$
      if $content(msg) = resource\_request$
    $ERROR\_MESSAGE_{i,conv,msg}$
      otherwise
```

If the message is a request, then, depending on the type of request, different processes are executed. An agent (e.g., a worker) may issue requests for status of the spacecraft, requests for information on the leader's goals or model, or requests for resources (e.g., a request that some workers under the leader's direction should form a sub-team to investigate a particular asteroid, or that a messenger should be relocated to perform communication functions). Since this specification is an abstraction of lower level specifications, it does not affect specifications in other views.

$$\begin{array}{l} LEADER\_STATUS\_REQ_{i,conv,msg} = \\ \quad \mathbf{leader_i!reply}(msg, current\_status()) \\ \quad \rightarrow LEADER\_COM_{i,conv} \end{array}$$

As shown above, if the request is for status, then the current spacecraft status is sent back to the sender using a standard function that retrieves the status. The $leader!reply$ part of the specification is a communication that causes a change of state in the WSCCS, X-Machine, and Unity Logic specifications.

$$\begin{array}{l} LEADER\_INFO\_REQ_{i,conv,msg} = \\ \quad \text{case } \mathbf{goals\_channel_i}!(REQUEST, msg) \rightarrow LEADER\_COM_{i,conv} \\ \qquad\quad \text{if } content(msg) = goals\_request \\ \qquad \mathbf{leader\_model_i}!(REQUEST, msg) \rightarrow LEADER\_COM_{i,conv} \\ \qquad\quad \text{if } content(msg) = model\_request \\ \qquad ERROR\_MESSAGE_{i,conv} \\ \qquad\quad \text{otherwise} \end{array}$$

For the LEADER_INFO_REQ process, if the request for information is for the leader's goals, a message is sent to the leader's intelligence process to retrieve the information, which is then sent to the requestor. If the request is for part of the leader's model, then a request is sent to the leader's model process, which then sends the model information to the requestor. The goals and the model are specified as part of the X-Machine specification, and the communication causes a change of state in the WSCCS, X-Machine, and Unity Logic specifications.

$$\begin{array}{l} LEADER\_RESOURCE\_REQ_{i,conv,msg} = \\ \quad \mathbf{goals\_channel_i}!(RESOURCE, msg) \\ \quad \rightarrow LEADER\_COM_{i,conv} \end{array}$$

For resource requests, the goals of the leader must be checked to determine whether giving up the resource would affect the leader's current goals. The message is therefore sent to the intelligence process to check against the current goals, to update its goals and model (in case it can give up the resources), and to reply to the requestor as appropriate. The highlighted channel indicates that the goals are also specified in the X-Machine view and that the communication causes a change of state in the WSCCS, X-Machine, and Unity Logic specifications.

$LEADER\_RECEIVE_{i,conv,msg} =$
  case $\mathbf{LEADER\_STATUS\_RECEIVED}_{i,conv,msg}$
    if $content(msg) = status\_returned$
   $\mathbf{LEADER\_INFO\_RECEIVED}_{i,conv,msg}$
    if $content(msg) = info\_returned$
   $\mathbf{LEADER\_RESOURCE\_RECEIVED}_{i,conv,msg}$
    if $content(msg) = resource\_request\_return$
   $\mathbf{ERROR\_MESSAGE}_{i,conv,msg}$
    otherwise

After sending a request to other entities, a leader will receive messages back that give requested status, information, or resources. The above LEADER_RECEIVE is the process that handles the messages that have been sent back. These messages have to be sent to the deliberative part of the leader so the leader's goals and models can be updated. The communication changes the state machines in the other views, and the Unity Logic specification also defines conditions for receiving messages from other spacecraft.

$LEADER\_STATUS\_RECEIVED_{i,conv,msg} =$
  $\mathbf{goals\_channel_i}!(STATUS\_RECEIVED, msg)$
  $\rightarrow LEADER\_COM_{i,conv}$

The LEADER_STATUS_RECEIVED process is executed when the leader receives a status message from another leader. When this happens, a message is sent to the goals channel containing the message so that the goals and model can be updated. The goals may need updating if the status indicates the worker is no longer able to perform some of its functions. This channel is highlighted to indicate a specification in the X-Machine view.

$LEADER\_INFO\_RECEIVED_{i,conv,msg} =$
  $\mathbf{goals\_channel_i}!(INFO\_RECEIVED, msg)$
  $\rightarrow LEADER\_COM_{i,conv}$

The LEADER_INFO_RECEIVED process is executed when the leader receives an informational message from another leader. As above, the message is sent to the goals process for possible updates of the goals and leader's model. This channel, again, is highlighted to indicate a specification in a different view (X-Machine).

$LEADER\_RESOURCE\_RECEIVED_{i,conv,msg} =$
  $\mathbf{goals\_channel_i}!(RESOURCE\_RECEIVED, msg)$
  $\rightarrow LEADER\_COM_{i,conv}$

As with the above processes, when the LEADER_RESOURCE_RECEIVED process is executed, a message is sent to the goals process of the leader for updating of the goals and model, and for taking action if necessary. This channel is highlighted to indicate a specification in a different view (X-Machine).

$ERROR\_MESSAGE_{i,conv,msg} =$
$\quad \mathbf{messenger_i!return}(msg, error)$
$\quad \rightarrow LEADER\_COM_{i,conv}$

If a message does not match any of the other interactions, then an error message is returned to the sending spacecraft. The error message is highlighted because a communication occurs, which causes a state change in the WSCCS, X-Machine, and Unity Logic views.

Messages to a leader from messengers, and messages to a leader from workers, have specifications similar to the above: see [34, 41, 42].

1.4.2 Weighted Synchronous Calculus of Communicating Systems View

To model the ANTS leader spacecraft, WSCCS (Weighted Synchronous Calculus of Communicating Systems), a process algebra, takes into account:

- The possible states (agents) of the leader;
- Actions in each agent-state that would qualify an agent to be "in" those states;
- Relative frequency of each action defined for an agent; and
- The priority of each action defined for an agent.

1.4.2.1 Actions Table 1.1 gives the actions, agent states, and view of priority on the actions of a leader. All of these actions are reflected in other specification views.

Since the state part of the specification is similar to the X-Machine and Unity Logic specifications, all changes in states in WSCCS would also affect the memory and state changes both in the X-Machine specification and in the state machine of Unity Logic, and would be subject to the predicates of Unity Logic. The above highlighted actions cause transitions into the communicating state, are specified as part of the CSP view, and indicate when the CSP specification would cause data to be passed on a CSP channel. In addition, the predicates of Unity Logic specify their own applicable preconditions. The reasoning and processing portions are also related to the CSP specification in the LEADER_INTELLIGENCE part of the specification (which is not included in this report but given in [34, 41, 42].

Continuing with the WSCCS specification, given the information from Table 1.1, we define the various agent-states as

$$AgentD \equiv n_a : a.AgentA + n_b : b.AgentB + n_c : c.AgentC$$

Here, n_a is a weight of the form $n\omega^k$ where n is the relative frequency of the action a, and k denotes the priority of action a, which would then turn agent D into agent A. The addition seen here represents a type of choice between possible actions. Thus, agent D may choose to perform action a, which would turn agent D in to agent A. Agent D makes this choice with frequency n and priority k. Or agent D may choose to perform action b, etc. Using this notation, the leader has agent states defined by the following statements:

Table 1.1 Agent state and actions

Agent State	Actions leading to the agent state	*f*	*p*
	Identity		
Communicating	SendMessageWorker	50	2
	SendMessageLeader	50	2
	SendMessageError	1	1
	ReceiveMessageWorker	50	2
	ReceiveMessageLeader	50	2
	ReceiveMessageError	1	1
Reasoning	ReasoningDeliberative	50	2
	ReasoningReactive	50	2
Processing	ProcessingSortingAndStorage	17	2
	ProcessingGeneration	17	2
	ProcessingPrediction	17	2
	ProcessingDiagnosis	16	2
	ProcessingRecovery	16	2
	ProcessingRemediation	17	2

$$
\begin{aligned}
&Communicating \equiv\\
&\quad 50\omega^3 : ReasoningDeliberative.Reasoning\\
&\quad +50\omega^3 : ReasoningReactive.Reasoning\\
&\quad +50\omega^2 : ReasoningDeliberative.Reasoning\\
&\quad +17\omega^3 : ProcessingSortingAndStorage.Processing\\
&\quad +17\omega^3 : ProcessingGeneration.Processing\\
&\quad +17\omega^3 : ProcessingPrediction.Processing\\
&\quad +16\omega^4 : ProcessingDiagnosis.Processing\\
&\quad +16\omega^4 : ProcessingRecovery.Processing\\
&\quad +17\omega^4 : ProcessingRemediation.Processing
\end{aligned}
$$

According to this statement, a leader, when in a communicating state, has the option (is allowed) to perform any action from the set:

$$
\begin{aligned}
&\{ReasoningDeliberative, ReasoningReactive,\\
&ProcessingSortingAndStorage, ProcessingGeneration,\\
&ProcessingPrediction, ProcessingDiagnosis, ProcessingRecovery,\\
&ProcessingRemediation\}
\end{aligned}
$$

and that the communicating leader will perform *ReasoningDeliberative* with a probability of 50 out of 200 [25%] (the total of all above listed frequencies) and will give that action a priority of 3. The second term in the statements tells us that the communicating leader will perform *ReasoningReactive* with the same 25% probability and priority of 3. The symbol + in this notation denotes that the communicating leader will make a choice between the various allowed actions, and that that choice will be made based on the frequencies and priorities of each allowable action. WSCCS is the only one of the views that takes into account probabilities and priority of the state changes. The probabilities and priorities allow for the calculation of the steady state of the system and therefore what the emergent behavior would be.

A transitional semantics defines what series of actions are valid for a given agent, and allows us to interpret agents as finite state automata represented by a transition graph where nodes represent the agents and the edges between represent the weights and actions.

The agent's transition is defined as:

$$D\lceil\{a,b,c\} \overset{a\left[\frac{n_a}{n}\right]}{\longrightarrow} A\lceil\{a,b,c\}$$

where $a\left[\frac{n_a}{n}\right]$ is the probability of action a occurring and n is the sum of the relative frequencies of the possible actions a, b, and c. Consider the transition written above. This transition definition expresses that agent D can perform only action a, b, or c. The probability of agent D performing action a is $\left[\frac{n_a}{n}\right]$ and the outcome of that action is that agent D becomes agent A, who can perform only action a, b, or c.

The following is a portion of the specification for the communicating part of the swarm. The rest of the communicating part and the processing and reasoning parts are similar.

$$\begin{aligned}
&Communicating\lceil\{ReasoningDeliberative, ReasoningReactive,\\
&ProcessingSortingAndStorage, ProcessingGeneration,\\
&ProcessingPrediction, ProcessingDiagnosis, ProcessingRecovery,\\
&ProcessingRemediation\} \overset{ReasoningReliverative\left[\frac{50\omega^3}{200}\right]}{\longrightarrow}\\
&Reasoning\lceil\{SendMessageWorker, SendMessageWorkerVIAMessenger,\\
&SendMessageLeader, SendMessageLeaderVIAMessenger,\\
&SendMessageError, ReceiveMessageWorker,\\
&ReceiveMessageWorkerVIAMessenger, ReceiveMessageLeader,\\
&ReceiveMessageLeaderVIAMessenger, ReceiveMessageError,\\
&ProcessingSortingAndStorage, ProcessingGeneration,\\
&ProcessingPrediction, ProcessingDiagnosis, ProcessingRecover,\\
&ProcessingRemediation\}
\end{aligned}$$

The above statement about the communicating leader specifies that the communicating leader, when allowed the set of actions:

$$\begin{aligned}
&\{ReasoningDeliberative, ReasoningReactive,\\
&ProcessingSortingAndStorage, ProcessingGeneration,\\
&ProcessingPrediction, ProcessingDiagnosis, ProcessingRecovery,\\
&ProcessingRemediation\}
\end{aligned}$$

will choose the action *ReasoningDeliberative* with a probability of 50% and a priority of 3; and that, when the action is performed, the communicating leader will transition to a reasoning leader who is allowed to choose from the set of actions given below to then be able to transition to another state.

$$\{SendMessageWorker, SendMessageWorkerVIAMessenger, SendMessageLeader, SendMessageLeaderVIAMessenger, SendMessageError, ReceiveMessageWorker, ReceiveMessageWorkerVIAMessenger, ReceiveMessageLeader, ReceiveMessageLeaderVIAMessenger, ReceiveMessageError, ProcessingSortingAndStorage, ProcessingGeneration, ProcessingPrediction, ProcessingDiagnosis, ProcessingRecovery, ProcessingRemediation\}$$

For the other views, each time there is a transition, part of the CSP, X-Machine, and Unity Logic specifications would represent how, and the conditions under which, the communications would be performed.

1.4.2.2 Transition Graph A transition graph derived from these transitions for the ANTS Leader Spacecraft is shown in Figure 1.2. (Nodes represent the agents and the edges between represent the weights and actions.)

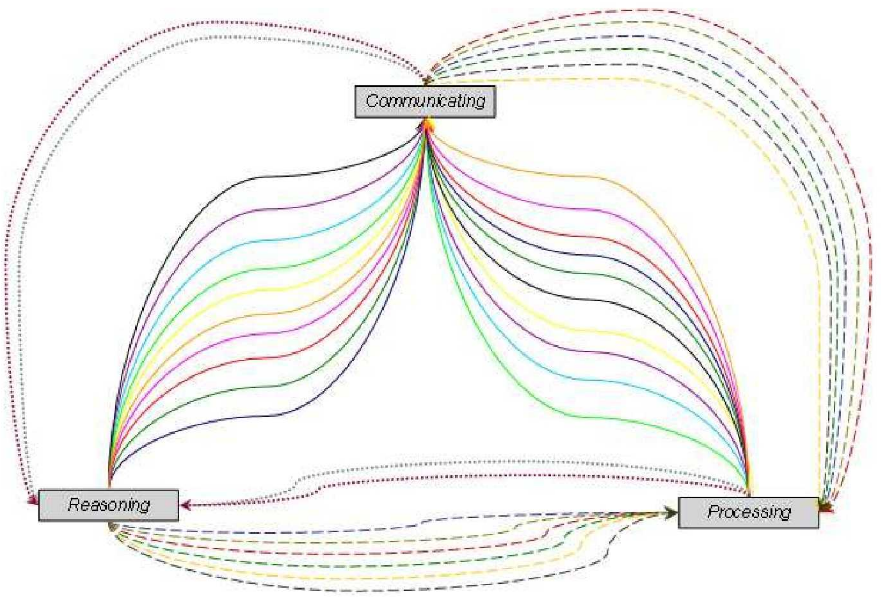

Figure 1.2 Transition graph from WSCCS specification showing weights and actions.

1.4.3 X-Machines

X-Machines are defined as the following tuple:

$$L = (Input, Memory, Output, Q, \Phi, F, start, m_0)$$

where the components of the tuple are defined as:

- $Input$, $Memory$, $Output$, are sets of data.
- Q is a finite set of states.
- Φ is a set of (partial) transition functions where each transition function maps $Memory \times Input \rightarrow Output \times Memory$.
- F is the next-state partial function, $F : Q \times \Phi \rightarrow Q$.
- $start \in Q$ is the initial state.
- $m_0 \in Memory$ is the initial value of memory.

For the ANTS specification, the components of the X-Machine are defined as:

- $Input = \{worker, messenger, leader, error, Deliberative, Reactive, SortAndStore, Generate, Predict, Diagnose, REcover, Remediate\}$.
- $Memory$ is written as a tuple $m = (Goals, Model)$ where $Goals$ describes the goals of the mission and $Model$ describes the model of the universe maintained by the leader. The initial memory is denoted by $(Goals_0, Model_0)$. When the goals and/or model changes, the new tuple will be denoted as $m' = (Goals', Model')$.
- $Output = \{SentMessageWorker, SentMessageMessenger, SentMessageLeader, SentMessageError, ReceivedMessageWorker, ReceivedMessageMessenger, ReceivedMessageLeader, ReceivedMessageError, ReasonedDeliberatively, ReasonedReactively, ProcessedSortingAndStoring, ProcessedGeneration, ProcessedPrediction, ProcessedDiagnosis, ProcessedRecovery, ProcessedRemediation\}$.
- $Q = \{Start, Communicating, Reasoning, Processing\}$.
- $\Phi = \{SendMessage, ReceiveMessage, Reason, Process\}$ where these functions are defined as in Table 1.2.

To see the Leader Spacecraft in these terms, consider Table 1.3 which depicts the states, transition functions, and associated inputs, outputs, and memory. A transition diagram for the ANTS Leader Spacecraft is shown in Figure 1.3. (Nodes represent the states, and the edges between represent the transition functions.)

For viewpoint specification interactions, all of the above would affect either the CSP specification, the WSCCS specification, or the Unity Logic specification. Since

Table 1.2 Leader States and Transitions

Q State	Φ the agent state	$Q' = F(Q, \Phi)$
Start	SendMessage	Communicating
	ReceiveMessage	Communicating
	Reason	Reasoning
	Process	Processing
Communicating	SendMessage	Communicating
	ReceiveMessage	Communicating
	Reason	Reasoning

Table 1.3 ANTS Role: Leader Spacecraft

State	Transition Functions $F(Q, \Phi)$	Function Definition $\Phi(m, \sigma) =$
Start		
Commun.	Send Msg	$\Phi(m, Worker) = (m', SentMessageWorker)$
		$\Phi(m, Messenger) = (m', SentMessageMessenger)$
		$\Phi(m, Leader) = (m', SentMessageLeader)$
		$\Phi(m, Error) = (m', SentMessageError)$
	RecvMsg	$\Phi(m, Worker) = (m', SentMessageWorker)$
		$\Phi(m, Messenger) = (m', SentMessageMessenger)$
		$\Phi(m, Leader) = (m', SentMessageLeader)$
		$\Phi(m, Error) = (m', SentMessageError)$
Reasoning	Reasoning	$\Phi(m, Deliberative) = (m', ReasonedDeliberatively)$
		$\Phi(m, Reactive) = (m', ReasonedDeactively)$
Processing	Processing	$\Phi(m, SortAndStore) = (m', ProcessedSortingAndStoring)$
		$\Phi(m, Generate) = (m', ProcessedGeneration)$
		$\Phi(m, Predict) = (m', ProcessedPrediction)$
		$\Phi(m, Diagnose) = (m', ProcessedDiagnosis)$
		$\Phi(m, Recover) = (m', ProcessedRecovery)$
		$\Phi(m, Remediate) = (m', ProcessedRemediation)$

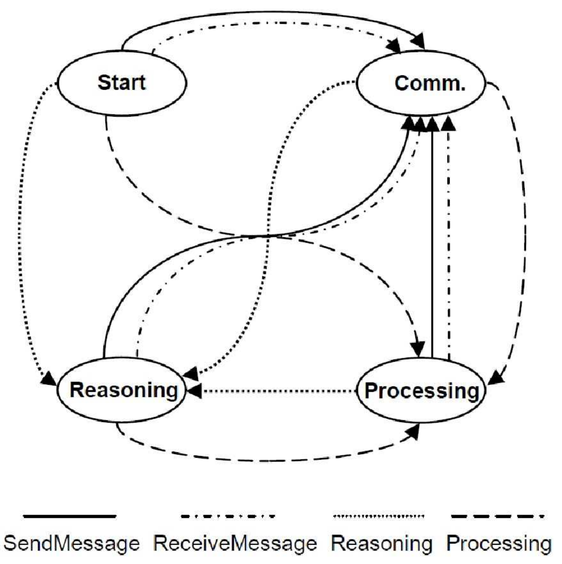

Figure 1.3 Transition diagram for the leader spacecraft as an X-Machine.

the states are the same (with the exception of the start state) any time there is a state change in the X-Machine specification, the priorities and probabilities in the WSCCS specification, as well as the Unity Logic state machine and predicates, would need to be checked. Due to their similarities, the specifications could be combined with the goals and model data added. For the CSP view, any time there is a change to a communicating state, the communication specifications in CSP would take over, particularly the passing of data over the channels. Similarly for Unity Logic, any time there is a communication, the predicates for the logic would affect whether the transition would be taken.

1.4.4 Unity Logic

To model the ANTS leader spacecraft with Unity Logic, we consider states of the leader just as in WSCCS and X-Machine and other state-machine based specification languages. In Unity Logic, we will consider the states of the leader and the actions taken to put the leader in those states, but the notation will appear much closer to classical logic.

Predicates are defined to represent the actions that would put the leader into its various states (Table 1.4). Those predicates then become statements which, if true, would mean that the leader had performed an action that put it into the corresponding state (see Table 1.5).

The communicating part of the leader program would then be specified using the following assertions (the reasoning and processing would be similar):

Table 1.4 Predicates and meanings in X-Machine transition diagram.

Predicate	Meaning
SendMessage(Leader, Worker)	A Leader sent a message to a Worker
SendMessageVIAMessenger(Leader, Worker)	A Leader sent a message to a Worker by relaying it through a Messenger
SendMessage(Leader, Leader)	A Leader sent a message to another Leader
SendMessageVIAMessenger(Leader, Leader)	A Leader sent a message to another Leader by relaying through a Messenger
SendMessageError(Leader)	A Leader sent a message in Error
ReceiveMessage(Leader, Worker)	A Leader received a message from a Worker
ReceiveMessageVIAMessenger(Leader, Worker)	A Leader received a message from a Worker who relayed through a Messenger
ReceiveMessage(Leader, Leader)	A Leader received a message from another Leader
ReceiveMessageVIAMessenger(Leader, Leader)	The Leader received a message from another Leader who relayed it through a Messenger
ReceiveMessageError(Leader)	A Leader received a message in Error
ReasoningDeliberative(Leader)	A Leader is reasoning deliberatively
ReasoningReactive(Leader)	A Leader is reasoning reactively
ProcessingSortingAndStorage(Leader)	The Leader is processing by Sorting, Classifying and/or Storing Data
ProcessingGeneration(Leader)	A Leader is processing by Model Generation
ProcessingPrediction(Leader)	A Leader is processing by prediction of asteroid properties, or by prediction of resource (worker & comm.) availability
ProcessingDiagnosis(Leader)	A Leader is processing for Diagnosis
ProcessingRecovery(Leader)	A Leader is processing for Recovery
ProcessingRemediation(Leader)	A Leader is processing for Remediation

Table 1.5 Predicates and meanings in X-Machine transition diagram.

Program State	Statements which, if true, lead to that program state
	SendMessage (Leader, Worker)
	SendMessageVIAMessenger(Leader, Worker)
	SendMessage(Leader, Leader)
	SendMessageVIAMessenger(Leader, Leader)
	SendMessageError(Leader)
Communicating	ReceiveMessage(Leader, Worker)
	ReceiveMessageVIAMessenger(Leader, Worker)
	ReceiveMessage(Leader, Leader)
	ReceiveMessageVIAMessenger(Leader, Leader)
	ReceiveMessageError(Leader)
Reasoning	ReasoningDeliberative(Leader)
	ReasoningReactive(Leader)
	ProcessingSortingAndStorage(Leader)
	ProcessingGeneration(Leader)
Processing	ProcessingPrediction(Leader)
	ProcessingDiagnosis(Leader)
	ProcessingRecovery(Leader)
	ProcessingRemediation(Leader)

$$[Communicating]ReasoningDeliberative(Leader)\,[Reasoning]$$
$$[Communicating]ReasoningReactive(Leader)\,[Reasoning]$$
$$[Communicating]ProcessingSortingAndStorage(Leader)\,[Processing]$$
$$[Communicating]ProcessingGeneration(Leader)\,[Processing]$$
$$[Communicating]ProcessingPrediction(Leader)\,[Processing]$$
$$[Communicating]ProcessingDiagnosis(Leader)\,[Processing]$$
$$[Communicating]ProcessingRecovery(Leader)\,[Processing]$$
$$[Communicating]ProcessingRemediation(Leader)\,[Processing]$$

Unity Logic then provides a logical syntax equivalent to propositional logic for reasoning about these predicates and the states they imply, as well as for defining specific mathematical, statistical, and other simple calculations to be performed. For the view specification, the predicates would define the conditions when transitions would take place. The state machine specified above is similar to the WSCCS and X-Machine state machines and could be combined with them.

1.5 CONCLUSION

This project has shown how an integration of the formal methods Communicating Sequential Processes (CSP), Weighted Synchronous Calculus of Communicating Systems (WSCCS), X-Machines, and Unity Logic can specify and even predict emergent behavior of a swarm-based mission such as the Autonomous Nano-Technology Swarm (ANTS) concept mission. With CSP providing the inter-process communication view, X-Machines providing the state machine and memory views, WSCCS providing the probability and priority views, and Unity Logic the logic views, these formal methods could provide what is needed for verifying swarms of spacecraft or other swarm-based systems using a viewpoint integration.

There is overlap between the four formal methods, particularly the state machine aspects of WSCCS, X-Machines, and Unity Logic. Thus, *conserving integration* or a *monolithic integration* could provide a more integrated specification. The monolithic integration starts by going back to the base formalisms (often the first order logic definition of the language) and then merging the base formalisms and redefining the semantics of the formalisms. With a conserving integration, the base formalisms are integrated more loosely by preserving the base formalisms [45]. Budget and time resources were not sufficient for completion of the work in this direction, but this type of integration would reduce the overlap in the viewpoint specifications and provide a tighter integration of the formal methods, with the complementary types of formal methods each providing only their strengths, as described above.

ACKNOWLEDGMENTS

This work was supported by the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) and managed by the NASA Independent Verification and Validation (IV&V) Facility. This work was also supported in part by Science Foundation Ireland grant 03/CE2/I303_1 to Lero—the Irish Software Engineering Research Centre. This paper is substantially based on [34].

REFERENCES

1. I. Bakam, F. Kordon, C. Le Page, and F. Bousquet. Formalization of a spatialized multi-agent model using Coloured Petri Nets for the study of an hunting management system. In *Proc. First International Workshop on Formal Approaches to Agent-Based Systems (FAABS I)*, number 1871 in LNAI, Greenbelt, Maryland (USA), April 2000. Springer.
2. Judith Barnard, Jenny Whitworth, and Mike Woodward. Communicating X-Machines. *Information and Software Technology*, 38(6), June 1996.
3. Gerardo Beni. The concept of cellular robotics. In *Proc. 1988 IEEE International Symposium on Intelligent Control*, pages 57–62. IEEE Computer Society Press, Los Alamitos, California (USA), 1988.
4. Gerardo Beni and Jing Want. Swarm intelligence. In *Proc. Seventh Annual Meeting of the Robotics Society of Japan*, pages 425–428, Tokyo (Japan), 1989. RSJ Press.
5. M. Blackburn, R. Busser, A. Nauman, R. Knickerbocker, and R. Kasuda. Mars Polar Lander fault identification using model-based testing. In *Proceedings of the 26th Annual IEEE/NASA Software Engineering Workshop (SEW)*, Greenbelt, MD, December 2001.
6. Eric Bonabeau, Marco Dorigo, and Guy Théraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, New York (USA), 1999.
7. Eric Bonabeau and Guy Théraulaz. Swarm smarts. *Scientific American*, pages 72–79, March 2000.
8. Eric Bonabeau, Guy Théraulaz, Jean-Louis Deneubourg, Serge Aron, and Scott Camazine. Self-organization in social insects. *Trends in Ecology and Evolution*, 12:188–193, 1997.

9. L. Bonnet, G. Florin, L. Duchien, and L. Seinturier. A method for specifying and proving distributed cooperative algorithms. In *Proc. DIMAS-95*, November 1995.

10. Robert Büssow, Robert Geisler, and Marcus Klar. Specifying safety-critical embedded systems with Statecharts and Z: A case study. In Astesiano, editor, *Proc. International Conference on Fundamental Approaches to Software Engineering*, number 1382 in LNCS, pages 71–87, Berlin (Germany), 1998. Springer-Verlag.

11. Shawn Carlson. Artificial life: Boids of a feather flock together. *Scientific American*, November 2000.

12. John Casani, Charles Whetsler, Arden Albee, Steven Battel, Richard Brace, Garry Burdick, Peter Burr, Duane Dippoey, Jeff Lavell, Charles Leising, Duncan MacPherson, Wesley Menard, Richard Rose, Robert Sackheim, and Al Schallenmuller. Report on the loss of the Mars Polar Lander and Deep Space 2 missions. Technical Report JPL D-18709, Jet Propulsion Laboratory, California Institute of Technology, 2000.

13. K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation*. Addison-Wesley Publishing Company, 1988.

14. Edmund M. Clare and Jeannette M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, December 1996.

15. Pamela E. Clark, Steven A. Curtis, and Michael L. Rilee. ANTS: Applying a new paradigm to Lunar and planetary exploration. In *Proc. Solar System Remote Sensing Symposium*, Pittsburgh, Pennsylvania (USA), 20–21 September 2002.

16. Steven A. Curtis, J. Mica, J. Nuth, G. Marr, Michael L. Rilee, and Maharaj K. Bhat. ANTS (Autonomous Nano-Technology Swarm): An artificial intelligence approach to Asteroid Belt resource exploration. In *Proc. Int'l Astronautical Federation, 51st Congress*, October 2000.

17. Steven A. Curtis, Walter F. Truszkowski, Michael L. Rilee, and Pamela E. Clark. ANTS for the human exploration and development of space. In *Proc. IEEE Aerospace Conference*, Big Sky, Montana (USA), 9–16 March 2003.

18. John Derrick, Eerke Boiten, Howard Bowman, and Maarten Steen. Supporting ODP - translating LOTOS to Z. In *First IFIP International workshop on Formal Methods for Open Object-based Distributed Systems*, pages 399–406. Chapman & Hall, 1996.

19. Clemens Fischer. *Combination and Implementation of Processes and Data: From CSP-OZ to Java*. PhD thesis, Universität Oldenburg, Germany, 2000.

20. A. K. Gala and A. D. Baker. Multi-agent communication in JAFMAS. In *Proc. Workshop on Specifying and Implementing Conversation Policies, Third International Conference on Autonomous Agents (Agents '99)*, Seattle, Washington (USA), 1999.

21. A. J. Galloway and W. J. Stoddart. An operational semantics for ZCCS. In M. Hinchey and S. Liu, editors, *Proc. IEEE International Conference on Formal Engineering Methods (ICFEM-97)*, pages 272–282, Hiroshima (Japan), November 1997. IEEE Computer Society Press, Los Alamitos, California (USA).

22. David E. Hiebeler. The swarm simulation system and individual-based modeling. In *Proc. Decision Support 2001: Advanced Technology for Natural Resource Management*, Toronto (Canada), September 1994.

23. M. Hinchey, J. Rash, and C. Rouff. Verification and validation of autonomous systems. In *Proc. SEW-26, 26th Annual NASA/IEEE Software Engineering Workshop*, pages 136–144, Greenbelt, Maryland (USA), November 2001. NASA Goddard Space Flight Center,

Greenbelt, Maryland (USA), IEEE Computer Society Press, Los Alamitos, California (USA).

24. Michael G. Hinchey and Steven A. Jarvis. *Concurrent Systems: Formal Development in CSP*. International Series in Software Engineering. McGraw-Hill International, London (UK), 1995.

25. Michael G. Hinchey, James L. Rash, and Christopher A. Rouff. Towards an automated development methodology for dependable systems with application to sensor networks. In *Proc. IEEE Workshop on Information Assurance in Wireless Sensor Networks (WSNIA 2005), Proc. International Performance Computing and Communications Conference (IPCCC-05) (Reprinted in Proc. Real Time in Sweden 2005 (RTiS2005), the 8th Biennial SNART Conference on Real-time Systems, 2005)*, Phoenix, Arizona (USA), 7–9 April 2005. IEEE Computer Society Press, Los Alamitos, California (USA).

26. C. A. R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, 1978.

27. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science. Prentice Hall International, Englewood Cliffs, New Jersey (USA), 1985.

28. W. Michael L. Holcombe. Mathematical models of cell biochemistry. Technical Report CS-86-4, Sheffield University, UK, 1986.

29. W. Michael L. Holcombe. Towards a formal description of intracellular biochemical organization. Technical Report CS-86-1, Sheffield University, UK, 1986.

30. W. Michael L. Holcombe. X-Machines as a basis for system specification. *Software Engineering*, 3(2):69–76, 1988.

31. Francesco Luna and Benedikt Stefansson. *Economic Simulations in Swarm: Agent-Based Modelling and Object Oriented Programming*. Kluwer Academic Publishers, 2000.

32. H. Van Dyke Parunak and R.S. Vanderbok. Managing emergent behaviour in distributed control systems. In *Proceedings of ISA-Tech'97*, Anaheim, CA, 1997.

33. Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

34. C. Rouff, A. Vanderbilt, M. Hinchey, W. Truszkowski, and J. Rash. Formal methods for swarm and autonomic systems. In *Proc. 1st International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, Cyprus, 30 October – 2 November, 2004.

35. C. Rouff, A. Vanderbilt, M. Hinchey, W. Truszkowski, and J. Rash. Properties of a formal method for prediction of emergent behaviors in swarm-based systems. In *Proc. 2nd IEEE International Conference on Software Engineering and Formal Methods*, Beijing (China), September 2004.

36. Christopher Rouff, Mike Hinchey, Joaquin Pena, and Antonio Ruiz-Cortes. Using formal methods and agent-oriented software engineering for modeling NASA swarm-based systems. In *IEEE Proc. Swarm Intellignece Symposium*, pages 348–355, April 2007.

37. Christopher A. Rouff, Michael G. Hinchey, Walter F. Truszkowski, and James L. Rash. Verifying large numbers of cooperating adaptive agents. In *Proc. 11th International Conference on Parallel and Distributed Systems (ICPADS-2005)*, Fukuoka (Japan), 20–22 July 2005.

38. Christopher A. Rouff, Michael G. Hinchey, Walter F. Truszkowski, and James L. Rash. Experiences applying formal approaches in the development of swarm-based space exploration systems. *International Journal of on software Tools for Technology Transfer. Special Issue on Formal Methods in Industry*, 8(6), 2006.

39. Christopher A. Rouff, James L. Rash, and Michael G. Hinchey. Experience using formal methods for specifying a multi-agent system. In *Proc. Sixth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2000)*, Tokyo (Japan), 2000. IEEE Computer Society Press, Los Alamitos, California (USA).

40. Christopher A. Rouff, Walter F. Truszkowski, Michael G. Hinchey, and James L. Rash. Verification of emergent behaviors in swarm based systems. In *Proc. 11th IEEE International Conference on Engineering Computer-Based Systems (ECBS), Workshop on Engineering Autonomic Systems (EASe)*, pages 443–448, Brno (Czech Republic), May 2004. IEEE Computer Society Press, Los Alamitos, California (USA).

41. Christopher A. Rouff, Walter F. Truszkowski, Michael G. Hinchey, and James L. Rash. Verification of NASA emergent systems. In *Proc. 9th IEEE International Conference on Engineering of Complex Computer Systems*, Florence (Italy), April 2004. IEEE Computer Society Press, Los Alamitos, California (USA).

42. Christopher A. Rouff, Walter F. Truszkowski, James L. Rash, and Michael G. Hinchey. A survey of formal methods for intelligent swarms. Technical Report TM-2005-212779, NASA Goddard Space Flight Center, Greenbelt, Maryland (USA), 2005.

43. Melissa Savage and Manor Askenazi. Arborscapes: A swarm-based multi-agent ecological disturbance model. Working paper 98-06-056, Santa Fe Institute, Santa Fe, New Mexico (USA), 1998.

44. S. Schneider, J. Davies, D. M. Jackson, G. M. Reed, J. Reed, and A. W. Roscoe. Timed CSP: Theory and practice. In *Proc. REX, Real-Time: Theory in Practice Workshop*, volume 600 of *LNCS*, pages 640–675. Springer-Verlag, 3–7 June 1991.

45. C. Suhl. RT-Z: An integration of Z and Timed CSP. In *Proc. 1st International Conference on Integrated Formal Methods (IFM99)*, York (UK), June 1999.

46. D. J. T. Sumpter, G. B. Blanchard, and D. S. Broomhead. Ants and agents: a process algebra approach to modelling ant colony behaviour. *Bulletin of Mathematical Biology*, 63(5):951–980, September 2001.

47. W. F. Truszkowski, M. G. Hinchey, J. L. Rash, and C. A. Rouff. Autonomous and autonomic systems: A paradigm for future space exploration missions. *IEEE Transactions on Systems Man and Cybernetics – Part C: Applications and Reviews*, 36(3), May 2006.

48. Walt Truszkowski, Mike Hinchey, James Rash, and Christopher Rouff. NASA's swarm missions: The challenge of building autonomous software. *IEEE IT Professional*, 6(5):47–52, September/October 2004.

49. Walter F. Truszkowski, Lou Hallock, Christopher A. Rouff, Jay Kerlin, James L. Rash, Michael G. Hinchey, and Roy Sterritt. *Autonomous and Autonomic Systems with Applications to NASA Intelligent Spacecraft Operations and Exploration Systems.* NASA Monographs in Systems and Software Engineering. Springer-Verlag, London (UK), 2009.

50. Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* MIT Press, Cambridge, Massachusetts (USA), 1999.