A Survey of Formal Methods for Intelligent Swarms

Christopher Rouff
SAIC

Walt Truszkowski, James Rash, Mike Hinchey
NASA Goddard Space Flight Center

December 14th, 2004

Table of Contents

1. Introduction

Swarms of intelligent autonomous spacecraft, involving complex behaviors and interactions, are being proposed for future space exploration missions. Such missions provide greater flexibility and offer the possibility of gathering more science data than traditional single spacecraft missions. The emergent properties of swarms make these missions powerful, but simultaneously far more difficult to design, and to assure that the proper behaviors will emerge. These missions are also considerably more complex than previous types of missions, and NASA, like other organizations, has little experience in developing or in verifying and validating these types of missions. A significant challenge when verifying and validating swarms of intelligent interacting agents is how to determine that the possible exponential interactions and emergent behaviors are producing the desired results. Assuring correct behavior and interactions of swarms will be critical to mission success.

The Autonomous Nano Technology Swarm (ANTS) mission is an example of one of the swarm types of missions NASA is considering. The ANTS mission will use a swarm of picospacecraft that will fly from Earth orbit to the Asteroid Belt. Using an insect colony analogy, ANTS will be composed of specialized workers for asteroid exploration. Exploration would consist of cataloguing the mass, density, morphology, and chemical composition of the asteroids, including any anomalous concentrations of specific minerals. To perform this task, ANTS would carry miniaturized instruments, such as imagers, spectrometers, and detectors.

Since ANTS and other similar missions are going to consist of autonomous spacecraft that may be out of contact with the earth for extended periods of time, and have low bandwidths due to weight constraints, it will be difficult to observe improper behavior and to correct any errors after launch. Providing V&V (verification and validation) for this type of mission is new to NASA, and represents the cutting edge in system correctness, and requires higher levels of assurance than other (traditional) missions that use a single or small number of spacecraft that are deterministic in nature and have near continuous communication access.

One of the highest possible levels of assurance comes from the application of formal methods. Formal methods are mathematics-based tools and techniques for specifying and verifying (software and hardware) systems. They are particularly useful for specifying complex parallel systems, such as exemplified by the ANTS mission, where the entire system is difficult for a single person to fully understand, a problem that is multiplied with multiple developers. Once written, a formal specification can be used to prove properties of a system (e.g., the underlying system will go from one state to another or not into a specific state) and check for particular types of errors (e.g., race or livelock conditions). A formal specification can also be used as input to a model checker for further validation.

This report gives the results of a survey of formal methods techniques for verification and validation of space missions that use swarm technology. Multiple formal methods were evaluated to determine their effectiveness in modeling and assuring the behavior of swarms of spacecraft using the ANTS mission as an example system. This report is the first result of the project to determine formal approaches that are promising for formally specifying swarm-based systems. From this survey, the most promising approaches were selected and are discussed relative to their possible application to the ANTS mission. Future work will include the application of an integrated approach, based on the selected approaches identified in this report, to the formal specification of the ANTS mission.

2. Intelligent Swarm Technology Overview

Bonabeau et al. (1997), 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."

Agent swarms are being used as a computer modeling technique and have also been used as a tool to study complex systems (Hiebeler, 1994). In swarm simulations, a group of interacting agents (Weiss, 1999) (often heterogeneous or near heterogeneous agents) has been studied for their emergent behavior. Examples of simulations that have been undertaken are swarms of birds (Reynolds, 1987; Carlson, 2000), business and economics (Luna and Stefansson, 2000) and ecological systems (Savage and Askenaki, 1998). In swarm simulations, each of the agents is given certain parameters that it tries to maximize. In terms of the bird swarms, each bird tries to find another bird to fly with, and it will try to fly off to one side and slightly higher to reduce its drag. Eventually the birds form flocks. Other types of swarm simulations have been developed that exhibit unlikely emergent behavior. The emergent behavior makes the whole greater than the sum of the individuals in the swarm. These emergent behaviors are the sums of often simple individual behaviors, but when aggregated, form complex and often unexpected behaviors. Swarm behavior is also being investigated for use in such applications as telephone switching, network routing, data categorizing, and shortest path optimizations (Bonabeau and Theraulaz, 2000).

Intelligent swarms (Bonabeau et al., 1999; Beni, 1998; Beni and Want, 1989) involve, minimally, simple agents and local interactions (interactions between agents and the environment). There is no central controller directing the swarm; they are self-organizing based on the emergent behaviors of the simple interactions. The emergent behavior is sometimes referred to as the macroscopic behavior and the individual behavior and local interactions as the microscopic behavior. These types of swarms exhibit self-organization since there is no external entity directing their behavior and no one agent has a global view of the intended macroscopic behavior. This type of behavior is observed in insects and flocks of birds.

One of the most challenging aspects of using swarms is how to verify that the emergent behavior of such systems will be proper and that no undesirable behaviors will occur. In addition to emergent behavior in swarms, there are also a large number of concurrent interactions going on between the agents that make up the swarms. These interactions can also contain errors, such as race conditions, that are very difficult to detect until they occur. Once they do occur, it can also be very difficult to recreate the errors since they are usually data and time dependent.

Intelligent swarm technology is based on swarm technology where the individual members of the swarm also have independent intelligence. This makes verifying such systems even more difficult since the swarms are no longer made up of homogeneous members with limited intelligence and communications. With intelligent swarms, members of the swarm may be heterogeneous or homogeneous. Even if members are initially homogeneous, their differing environments may cause them to learn different things and develop different goals and consequently become a heterogeneous swarm. Intelligent swarms may also be made up of heterogeneous elements from the outset, reflecting different capabilities as well as a possible social structure. Verifying such swarms will be difficult due to the complexity of each member but also due to the complex interaction of a large number of intelligent elements. This will create a huge state space, and, since the elements may be learning, the behavior of the individual elements and the emergent behavior of the swarm will be constantly changing and may be difficult to predict.

3. ANTS Mission Overview

The Autonomous Nano-Technology Swarm (ANTS) mission (ANTS Team; Clark et al., 2002; ANTS/PAM website; Curtis et al., 2000; Curtis et al. 2003) will have swarms of autonomous pico-class (approximately 1kg) spacecraft that will search the asteroid belt for asteroids that have specific characteristics. There will be approximately 1,000 spacecraft involved in the mission. The spacecraft will be initially carried to the asteroid belt by a transport ship and then released. Replacement spacecraft will be sent from Earth on an as-needed basis.

There will be several types of spacecraft involved in the mission (Figure 1). Some of the spacecraft, called workers, will have a specialized instrument onboard (e.g., a magnetometer, x-ray, gamma-ray, visible/IR, neutral mass spectrometer) and will gather specific types of data. Some will be coordinators (called rulers) that have rules that decided the types of asteroids and data the mission is interested in and will coordinate the efforts of the workers. The third type of spacecraft are the messengers that will coordinate communications between the workers, rulers and Earth. Each worker spacecraft will examine asteroids they encounter and send messages back to a coordinator that will then evaluate the data and send other appropriate satellites with specialized instruments to the asteroid to gather further information if needed. Approximately 80 percent of the spacecraft will be workers.

To implement this mission, a high degree of autonomy is being planned. A heuristic approach is being considered that provides for a social structure for the spacecraft based on the above hierarchy. Artificial intelligence technologies such as genetic algorithms, neural nets, fuzzy logic and on-board planners are being investigated to assist the mission in maintaining 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 high-latency and low bandwidth communications back to Earth.

4. Formal Approaches and Assurance

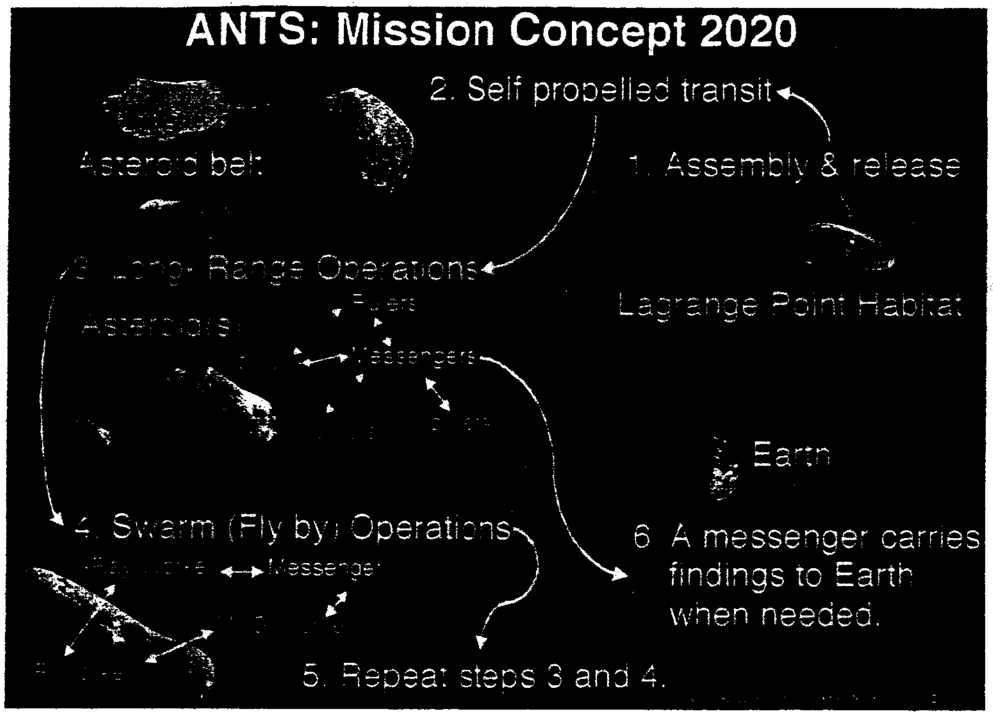

Figure 1: ANTS Mission Concept.

Software engineers are confronting the central dilemma of the field -- the necessity of producing high quality software -- knowing that, using traditional methods, 100% verification and validation (V&V) of non-trivial software systems is impossible. Addressing this dilemma is a crucial problem facing computer science and software engineering in general. For NASA, having software with less than 100% V&V means having less than 100% assurance of mission safety and success. Finding a solution to the software dilemma is a matter of urgency and is a major focus of computer science as well as for NASA.

As mission software becomes more complex, testing it also becomes more difficult. This is especially true of parallel processes and distributed computing, which NASA is increasingly developing and using on missions. Errors in these systems can rarely be found by inputting sample data into the system and checking if the results are correct. These types of errors are time-based and only occur when processes send or receive data at particular times or in a particular sequence. To find these errors, the software processes involved have to be executed in all possible combinations of states (state space) that the processes could collectively be in. The state space grows extremely rapidly (often exponentially) with the number of states in the processes, and becomes increasingly difficult to test with a relatively small number of processes. Traditionally, to get around the state explosion problem, testers have artificially reduced the number of states and approximated the underlying software using restricted models.

Formal methods are proven approaches for assuring the correct operation of complex interacting systems (Hinchey and Jarvis, 1995; Hoare, 1985; Clare and Wing, 1996). Once written, a formal specification can be used to prove properties of a system correct (e.g., the underlying system will go from one state to another, or not into a specific state), check for particular types of errors (e.g., race conditions), as well as used as input to a model checker.

5. Potential Candidates

The following is a list of potential formal approaches that could be used for formally specifying swarm related technologies, such as ANTS. This list gives the name of each of the formal approaches, a brief summary, history, applications that it has been used on, strengths, weaknesses, and tool support.

This list of approaches was determined through a literature search. A high emphasis was given to those approaches that have been used in agent technologies or other highly distributed, concurrent environment, which is the environment of swarms.

The following gives a brief overview of several formal methods that have been used to model agent-based systems.

5.1 Process Algebras

Process algebras generally are made up of the following: a language for describing systems, a behavioral equivalence or inequivalence that allows comparison of system behaviors, and axioms that allow for proofs of equivalence between systems. Some algebras include a refinement ordering to determine whether one system is a refinement of another. Process algebras usually use a handshake mechanism (rendezvous) between processes via a port or channel. One process may wait for data at a channel until another process sends data over the channel (or vice versa). Once the data is exchanged, both processes continue. The receiving process then may execute different processes based on the data received. Internal, non-communications aspects of processes are not reflected by process algebras.

5.1.1 Communicating Sequential Processes (CSP)

5.1.1.1 Summary

Communicating Sequential Processes (CSP) was designed by C.A.R. Hoare (1978; 1985) to specify and model concurrent systems. Systems specified in CSP consist of independently executing processes that communicate over unbuffered, unidirectional channels and use events for synchronization. Processes in CSP are recursively defined as the occurrence of an event followed by a process. The events guard the processes so that the process does not execute until the event occurs. When a process needs to synchronize with another process or send it data, data is sent over a channel and then blocks until the other process reads the data from the channel. If the reading process tries to read data from a channel and there is no data on the channel, it also blocks until data arrives. There are also standard processes, such as STOP, SKIP, RUN and bottom ($\perp$). Choice operators and conditionals also exist that allow for choosing one of many processes to execute depending on a condition.

CSP has a proof system associated with it so that properties of CSP specifications can be proven correct or not correct. Proofs are based on traces of events that can be produced by a specification. Every time an event occurs, the event is listed as part of a trace for the process. A specification has a set of acceptable traces that can occur. By applying the laws of CSP traces to a set of traces, it can be determined whether the given set of traces can be produced by a given specification. It can also be determined the set of possible traces that a specification can produce. Correctness of a specification can then be determined by proving that a sequence of traces can never be produced by the specification and/or that a set of traces can be produced by the specification. In addition, properties such as deadlock and liveness can also be proven as properties of a specification.

5.1.1.2 History

CSP was developed by C.A.R. Hoare and originally presented in a paper in the Communications of the ACM in 1978 (Hoare, 1978). A book (Hoare, 1985) was subsequently published in 1985 that contained updates on CSP that added the named channels and renaming. CSP has also been updated several times with several variations such as Timed CSP (Reed and Roscoe, 1987), CSP-i (Wrench, 1988) and Receptive Process Theory (Josephs, 1992).

It is a popular formal specification notation for concurrent systems and has been widely used to specify concurrent systems. It has often formed the basis of programming language extensions for concurrent systems and other specification systems.

5.1.1.3 Applications

CSP has been used in a wide range of applications for specification of concurrent systems. A few of these include:

- Specification of the LOGOS multi-agent system at NASA Goddard (Rouff et al., 2000; Hinchey et al., 2001),
- Specification of the Open Systems Interconnection (OSI) model (Hinchey and Jarvis, 1995),
- Specification of an operating system by Praxis Systems Ltd (Craigen et al., 1993),
- Specification of the Textronix 11000 oscilloscope,
- Specification of the T 800 Transputer for INMOS Ltd.,
- Specification of the T9000 Transputer for INMOS Ltd,

5.1.1.4 Strengths

The primary strength of CSP is that it was originally developed for describing concurrent systems. It is also a simple language that is easy to read with little training and also easy to write. By modeling systems using processes, events, and channels, the communication between processes is easily modeled and concurrency-related problems easily detected by inspection. CSP can also be used at differing levels of abstraction to give a high-level overview of a system as well as a detailed view. CSP allows for proof of correctness for deadlock and livelock as well as general safety and liveness properties. Other strengths noted of CSP are that it naturally supports specification of nondeterministic systems and is good at modeling resource sharing, process control, and real-time interactions.

Some model checkers have used CSP as a basis for their model checking languages. An example is the Promela model checking language for SPIN (Holzmann, 1991). This makes converting CSP into a model checking language straightforward and the process can even be automated. An advantage of using CSP instead of a model checking language directly is that the resulting CSP specification is more general purpose and can then be translated to a wide range of tools (see below) as opposed to just the model checker.

Many programming languages and other modeling languages are using CSP as a model when adding concurrency features; examples include languages such as Communicating Java Threads (CJT) (Hilderink et al., 1998), and Modula-P (Vollmer and Hoffart, 1992). This means that specifications written in CSP can be transfered to other implementation languages, and code or code fragments can be automatically generated based on CSP specifications.

5.1.1.5 Weaknesses

Due to the simplicity of the language (processes, events, and channels), specifications can become large and therefore difficult to read and understand. Another weakness noted is that CSP cannot explicitly deal with data or algorithmic issues. Data can be sent in messages through channels, but the manipulation of that data must be done in the context of sending the data through the channel. Other weaknesses noted are that specifications can become over-synchronized due to the limited language constructs, namely that (in its pure form) it does not support asynchronous message passing, it does not perform message buffering, and there can be readability issues.

5.1.1.6 Tool Support

There are several tools that support development of specifications in CSP or the implementation of CSP. The following is a partial list:

- CSP2B (Butler, 1999) is a tool that converts CSP specification to B specifications,
- Failures-Divergence Refinement (FDR) (Roscoe et al., 1995) is a model checker based on the theory of CSP,
- Occam (INMOS, 1984) is a parallel processing language based on CSP and implemented on the transputer processor,
- Java Communicating Sequential Processes (JCSP) (Lea, 1999) is a library of Java classes that give programmers a process model based on CSP,
- CCSP is an execution environment for CSP programs. The CSP programs are converted to C and then run as individual process on networked workstations,
- Communicating Java Threads (CJT) is a class library for Java that provides CSP channels, composition constructs, and scheduling of processes.

There are also other tools that use languages that are similar to CSP. One of those is the model checker SPIN (Holzmann, 1991) that uses a language called Promela. CSP is similar enough to

Promela that CSP specifications can be easily convert into Promela, and a converter could also be developed that automatically does the conversion.

5.1.2 Calculus for Communicating Systems (CCS)

5.1.2.1 Summary

The Calculus of Communicating Systems (CCS) (Milner, 1985) is a process algebra that was developed for reasoning about concurrent systems. CCS defines concurrent systems as a set of processes using actions (or events) and operators on the actions. Actions represent external inputs and outputs on ports from the environment or internal computation steps. Operators on actions consist of an action prefix operator, a nil operator, a choice operator (+), a parallel composition operator (|), a restriction operation (\) that permits actions to be localized within a system, a renaming operator [/] that maps actions to other actions, and an invocation function that allows systems to be defined recursively. Similar to CSP, processes in CCS are defined recursively and communicate with other processes through ports. CCS also has a set of axioms that can be used to reason about the processes and prove properties of systems.

5.1.2.2 History

CCS, developed by Robin Milner (1985), was one of the first process algebras. CCS has had several extensions made to it, including CCS with Broadcast (CCS+b) and Temporal CCS (TCCS). CSP and Pi Calculus are also considered to be extensions to CCS.

5.1.2.3 Applications

CCS is being used to specify agents, where instead of a process that is defined by a set of equations, an agent is substituted. It has also been used on the T9000 Transputer for INMOS Ltd. CCS was also extended by Tofts (1991) to model social insect behavior.

5.1.2.4 Strengths

Like CSP, the primary strength of CCS is that it was originally developed for describing concurrent systems. Process algebras are fairly easy to read with little training and also easy to write. By modeling systems using processes, actions and ports, the communication between processes are easily modeled and concurrency related problems can be easily detected by inspection. CCS can also be used to represent systems at different levels of abstraction to give a high-level overview of a system as well as a detailed view. With its refinement ability, CCS can also maintain equivalences between higher level and lower level specifications. In addition, there is the ability to test that two specifications are *bisimulations* of each other (two specifications simulate each other). CCS also allows for proof of correctness as well as deadlock and livelock.

5.1.2.5 Weaknesses

CCS has many of the weaknesses of CSP. Due to the simplicity of the language (processes, actions, and ports), specifications can become large and therefore difficult to read and understand. Another weakness is that CCS cannot explicitly deal with data or algorithmic issues. A limited form of data exchange can be performed by encoding values in port names, which makes the passing of data predefined and based on the name of the port. Specifications can also become overly synchronized due to the limited language constructs; it does not support asynchronous message passing; it does not perform message buffering; and there can be readability issues.

5.1.2.6 Tool Support

CCS is supported by a public domain tool called the Concurrency Workbench (CWB). It is an interactive tool that is available from several sources (e.g., University of Edinburgh (Moller and Stevens)). CWB displays simulations of concurrent systems specified in CCS. It can search for deadlocks, test for equality between agents and can determine if a system satisfies defined properties.

5.1.3 π-Calculus

5.1.3.1 Summary

≥-calculus (Milner et al., 1992) is a process algebra based on CCS that differs from some of the other process algebras in that it supports mobility of processes. It does this by being able to pass a link (channel) as data in a handshake. This allows data links to be passed to other processes and links can then be represented by variable names and compared for equality or inequality, and reasoned about.

There are two versions of the ≥-calculus: monadic calculus and polyadic calculus. The monadic calculus communicates one name at each handshake and the polyadic calculus can communicate zero or more names at each handshake. The two are equivalent (multiple single name communications can be used to represent one polyadic communication). Pi-calculus contains constructs for input and output on links, a silent (empty) prefix, sum operator, parallel composition operator, match and mismatch link comparison, restricted links, and parameterized process names which can be used for invocation.

5.1.3.2 History

The ≥-calculus is an extension to CCS and was first extended to pass link names by Astesiano and Zucca (1984) and Engberg and Nielson (1986). These early versions were viewed by some as overly complicated, and were later refined by Milner, Parrow and Walker (1992). There have been many other calculi that have also been based on the ≥-calculus, including Pict (Pierce and Turner, 1999), Facile (Borgia et al., 1996), Join (Fournet and Gonthier, 1996), Ambients (Cardelli and Gordon, 1998), Spi (Abadi and Gordon, 1998), and POOL (Walker, 1995)).

5.1.3.3 Applications

≥-calculus has been used by several people to model agent-based systems. Esterline et al. have used ≥-calculus to specify the LOGOS multi-agent system (2000) and Kawabe et al.(2000) have developed a ≥-calculus-based system called Nepi^2 to specify communicating software or agents.

5.1.3.4 *Strengths*

≥-calculus' strength is being able to model processes whose interconnections change over time. These types of processes are modeled by ≥-calculus' ability to transfer the name of a communication link (or channel) to another process, which can also pass it on to other processes. This gives it the ability to model systems whose resources vary over time as well as mobile processes. It also has the ability to define links that are private between two or more processes, but the restricted names are also transferable to other processes.

In addition, there is the ability to test that two specifications are *bisimulations* of each other (two specifications simulate each other). CCS also allows for proof of correctness as well as deadlock and livelock.

5.1.3.5 *Weaknesses*

≥-calculus does not provide for data passing between processes (only names of links). Other weaknesses are also similar to other process algebras: specifications can become large and therefore difficult to read and understand, it does not deal explicitly with data or algorithmic issues, specifications can become over-synchronized due to the limited language constructs, asynchronous message passing is not supported, there is no message buffering, and there can be readability issues.

5.1.3.6 *Tool Support*

The Concurrency Workbench (CWB) that is used to support CCS has been extended for ≥-calculus and is called the Mobility Workbench (MWB) (Victor and Moller, 1994). The MWB addresses the polyadic ≥-calculus, and like CWB can be used to decide equivalence, determine whether an agent satisfies a formula, can find deadlocks using a model checker, and provides an interactive simulator.

5.1.4 Input/Output Automata (IOA)

5.1.4.1 *Summary*

Input/output automaton (IOA) are nondeterministic state machines. They can be described as a labeled transition system for modeling asynchronous concurrent systems (Lynch and Tuttle, 1987). An IOA consists of a set of states with a transition function. IOA may have infinitely many states, and an infinite alphabet with strings of infinite length in the language that is accepted by the automata. Actions are classified as input, output or internal. The inputs to the automata are generated by its environment. The outputs and internal actions are generated by the automata with the outputs being sent to the environment. Actions can also have preconditions for them to fire.

An I/O automaton has "tasks"; in a fair execution of an I/O automaton, all tasks are required to get turns infinitely many times in any finite interval. The behavior of an I/O automaton is describable in terms of traces, or alternatively in terms of fair traces. Both types of behavior notions are compositional.

5.1.4.2 History

The input/output automaton model was developed by Lynch and Tuttle (1987). Variants of IOA have been developed that include Hybrid Automata (Lynch et al., 2003) for modeling systems that are a combination of continuous and discrete systems, Timed Automata (Grobauer and Muller, 1999) for reasoning about real-time systems, probabilistic versions (PIOA) (Wu et al., 1997) for specifying systems with a combination of probabilistic and nondeterministic behavior, and dynamic IOA (DIOA) (Attie and Lynch, 2001) for describing systems with run-time process creation, termination, and mobility.

5.1.4.3 Applications

IOA has been used to verify a number of types of systems, including various communication protocols (e.g., TCP) (Smith, 1997) and performance analysis of networks (upper and lower bounds and failures). It has also been used for specification and reasoning about agent-based systems (Tadashi et al., 2000).

5.1.4.4 Strengths

A strength of I/O Automata is that inputs from the environment can not be blocked. This enforces an environment driven model of the system. In addition, IOA can model multiple levels of abstractions of a system, from high-level specifications to detailed algorithms. IOA models are executable, can be simulated, and are highly nondeterministic. IOA is a calculus so I/O automata can be used to generate code. Finally, IOA has constructs for proving correctness of a specification.

5.1.4.5 Weaknesses

A former weakness of I/O Automata ,that there is not a formal algebra for it, has been addressed by the development of a process algebra that describes I/O automata (Nicola and Segala, 1995).

5.1.4.6 Tool Support

Isabelle/HOLCF (Hamberger, 1999) allows for fully formal tool-supported verification and model checking for specifications using I/O automata. Simulators have also been developed for I/O automata (Chefter, 1998).

5.2 Model-Oriented Approaches

5.2.1 Z

5.2.1.1 Summary

Z is based on Zermelo set theory and is used to describe the behavior of sequential processes. In general, concurrency and timing cannot be described in Z (although there are variants and extensions to address these issues).

Z is strongly typed, with types being associated with sets and operators of equality and membership defined for all types. The main construct of Z to describe the functionality of a system is the schema, which is a visual construct that includes a declaration part and an optional predicate part. The declaration part contains named and typed schema components with constraining information. The predicate part contains pre- and post-conditions of the components in the declaration as well as invariants and operations on the components. The schema calculus of Z allows schemas to be combined to form new schemas and describe the functionality of a system as a whole.

5.2.1.2 History

Z was originally developed by Jean-Raymond Abrial at the Programming Research Group at the Oxford University Computing Laboratory (OUCL) and further developed elsewhere since the late 1970s. There have also been several object oriented extensions to Z that include ZERO, MooZ, Object-Z, OOZE, Z++, ZEST, and Fresco, with Object-Z currently being the prominent version.

Z has also been combined with other formal methods, such as CSP, to give it the ability to handle concurrency and timing. An example is Timed Communicating Object Z (TCOZ) that has Object Z's strength in modeling algorithms and data, and CSP's strength in modeling process control and real-time interactions (Mahoney and Dong, 2000).

5.2.1.3 Applications

Z has been used in a wide range of applications. It has been used to formally specify a radiation therapy machine control program, sliding window protocol (Hinchey and Bowen, 1999), and a steam-boiler controller, among others.

For agent-based systems, d'Inverno and Luck (2000; 2001) have used Z to specify an agent framework. In their framework they have specified a four-tiered hierarchy that consists of entities, objects, agents, and autonomous agents. As part of the agent framework specification they specify engagement, cooperation, inter-agent relationships, sociological agent, plans, and goals.

A few of the applications specified by Z include (Craigen et al., 1993):

- Extensions to a secure operating system, a security policy, and a software development toolset by Praxis Systems Ltd,
- IBM's Customer Information Control System,
- Oscilloscope software at Textronix,
- The T 800 Transputer at INMOS Ltd.

Z has been used on a wide number of other projects with a high degree of success.

5.2.1.4 Strengths

Some of Z's strengths include having a basis in set theory and predicate calculus, its ability to model complex data and algorithms, its precise expressions of functions, and its notational variety. Z also has a wide range of tools to support specifications, type checking, typesetting, and verification with some programming languages.

5.2.1.5 Weaknesses

Z's main weakness in terms of agent-based and swarm systems is its sequential nature. It has been faulted for not being good at modeling concurrency. Some of the concurrency issues have been addressed by integrated models that use a process algebra for concurrency and Z for the sequential parts (Mahony and Dong, 2000), or Object-Z for the agent modeling and statecharts for the concurrency (Kawabe et al., 2000). Another weakness that has been noted is that since Z is mathematically based, it is theoretically possible to specify a system that is impossible to construct. In reality, however, this is avoided in that standard practice in the use of Z requires the specifier to define a valid initial state from which all other states are derived by application of valid operations. As a result, the system specified is guaranteed to be possible to construct.

5.2.1.6 Tool Support

There are a range of tools for formatting, type-checking, and aiding in proofs of Z. The following is a partial list:

- **ZTC** – is a type checker that is intended to be compliant with the 2nd edition of Spivey's Z Reference Manual.
- **ZANS** – is a research prototype Z animator.
- **FuZZ** – is a syntax and type checker with a LaTex style option.
- **CADiZ** – is a suite of integrated tools for preparing and type-checking Z specifications for documents.
- **ProofPower** – is a suite of tools that support specifications and proofs. It can also support verification of SPARK-Ada programs against Z specifications.
- **Sola** – is a commercial integrated support tool for Z for automated assistance in the specification construction, proving, and maintenance process. It is intended for system developers and includes an editor, type-checker, and tactical theorem prover. It is no longer actively supported.

- **Z/EVES** – is a tool for analyzing Z specifications. Some of its features include syntax and type checking, schema expansion, precondition calculation, domain checking, refinement proofs, and general theorem proving.
- **CZT** – Community Z Tools is an initiative that is ongoing to coordinate a set of Z tools.

5.2.2 B

5.2.2.1 Summary

The B method uses the Abstract Machine Notation (AMN), which is based on set theory and predicate logic. The AMN uses a finite state machine model (FSM) that supports states (variables of an abstract machine), invariants (constraints/relations between variables), and operations on the environment. Expressions in B can also have guards on them.

Development of a specification in B is done by first specifying the system behavior in AMN, refining the specification, and then implementing the specification. B specifications describe the state variables, invariants between the variables, and operations on the variables. The specification is developed iteratively through refinements of the model until the specification is completed. Verifications and simulations during the development of the specification can also be done using the B toolkit, a set of tools that support the methodology, to prove that invariants are preserved after operations are performed.

5.2.2.2 History

The B method was developed by Abrial (1996), who also developed the Z specification language. B is a relatively new formal method, but has already found a large amount of use in complex systems specifications.

5.2.2.3 Applications

The B-method has been used in a wide range of safety-critical applications. The following are a few of these:

- Railway Signaling System for the Paris rapid transit authority that was safety-critical (Craigen et al., 1993),
- Train excessive speed system (Craigen et al., 1993),
- Train deceleration control (Craigen et al., 1993),
- French census analysis and information system (Hinchey and Bowen, 1999),
- Chemical process controller (Hinchey and Bowen, 1999),
- Hardware circuits.

The B-method has also been modified for specifying distributed cooperative algorithms by adding temporal logic aspects to it (Bonnet et al., 1995).

5.2.2.4 Strengths

An advantage of the B-method is the iterative refinements, so specifications are developed in a top-down fashion. Another advantage is the component-based approach to developing the specifications, which maps well to component-based architectures and development methodologies.

An additional strength of the B method is its tool support. From a B specification, code can be generated, it can be analyzed for correctness, and an animation and proof of correctness can be performed. The ability to easily reuse specifications has also been cited as a strength of the B method and tools.

5.2.2.5 Weaknesses

As is the case with other methods based on finite state machines, specifying concurrent systems can present challenges due to the exponential growth of the state space of such systems.

5.2.2.6 Tool Support

The B method is supported by tools from a number of vendors, including BP International, Edinburgh Portable Compilers, Atelier B, and B-Core. The B Toolkit, from B-Core, includes an analyzer that generates proof obligations, a type checker, an animator, a status checker, and a prover. The B toolkit can also produce C, C++, and Ada code that implements B Specifications.

5.2.3 Finite State Machines (FSMs)

5.2.3.1 Summary

Finite State Machines (FSMs) model behavior using states and transitions between the states. Transitions contain events and conditions needed for the FSM to change states. The conditions act as guards on the transitions and the events are matches to inputs. States changes can occur when a transition from the current state has an event that matches the current input and the condition on the transition evaluates to true. For AI systems, FSMs often represent knowledge systems where the states represent knowledge and the transitions represent rules.

5.2.3.2 History

Finite state machines have been used in specifying AI related systems for a long time. Since FSMs are inherently sequential, they have been modified over time to work in a concurrent environment. Concurrent systems are often described using concurrent FSMs with the ability of the FSMs to communicate with each other either at checkpoints or through buffers. Extensions of FSMs include statecharts, fuzzy state machines (FuSM), and others.

5.2.3.3 Applications

FSMs have been used to specify a wide range of applications and have been very popular in specifying AI related applications. FSMs have also been used to specify multi-agent systems. They are usually modified so that concurrency and communication between the agents can be specified. An example is the Java-based Agent Framework for Multi-Agent Systems (JAFMAS) that uses FSMs to specify multi-agent conversations (Gala and Baker, 1999).

5.2.3.4 Strengths

FSMs are very natural in their expression of behavior and are straightforward to design, program, and execute efficiently. They are also taught to most engineering majors and so are understood by a wide range of people and are easily taught and learned by others. FSMs are also easy to analyze at the state level: states just need to be studied to determine which states have transitions to them and which states they reach and on what conditions and events.

5.2.3.5 Weaknesses

FSMs are inherently sequential, so they are not good by themselves to express concurrent systems, such as agent-based systems. Modifications to FSMs have been made so they can be used in concurrent systems

FSM are also "flat" in the way they are described, with no hierarchy are modularization, so specifications of systems can become very large and difficult to understand (spaghetti-nature of a large number of states and transitions). Because of this, they also do not support top-down or other refinement methodologies (Bowen and Hinchey, 1999). FSMs can also be uneconomical when it comes to transitions, especially for real-time systems, since a high level interrupt would have to be associated with each state. In addition, FSMs, due to the state paradigm, grow exponentially as the size of the system being specified grows linearly. There have been many modifications, such as concurrent FSMs and Statecharts, to overcome some of these shortcomings.

5.2.3.6 Tool Support

Since FSMs have been in use for a long time, there are a number of tools available to support them. There are FSM editors, simulators, and verification tools. Many of these are available through Computer Aided Software Engineering (CASE) tools. Code generators are also available that automatically produce code for a programming language.

5.2.4 Statecharts

5.2.4.1 Summary

Statecharts extend finite state machines by adding hierarchy, concurrency, and communication and were designed to specify complex discrete-event systems. The main advantage of statecharts over FSMs is that statecharts have built in the means to represent concurrency. The specifications can be developed in a hierarchical fashion, which aids in abstraction and top-down or bottom-up development.

5.2.4.2 History

Statecharts were developed by David Harel (1988; 1987; Harel et al., 1987). Statecharts have been widely used on many projects for specification and design of many types of systems. Coleman, Hayes, and Bear (1992) introduced a variant of statecharts called Objectcharts for object-oriented design.

Several integrated versions of Statecharts that in conjunction use formal methods have been introduced. Uselton and Smolka combine statecharts with a process algebra (1994a) and also added the Labeled Transition Systems algebra (1994b) in order to establish formal semantics for statecharts. Andrews, Day, and Joyce (1997) have used statecharts embeded with a typed predicate logic. Other integrated approaches have been introduced for real-time systems that embed the concept of time, such as Sowmya and Ramesh, who extended statecharts with temporal logic (1998).

5.2.4.3 Applications

Statecharts have been used successfully on a wide range of industry projects, and have a large number of advocates. It has also been used to specify agent-based systems by a number of people. A few of them include Kimiaghalam, et al. (2002) who have used a statechart-based approach for specifying agents, Hilaire et al. (2000) who used a combination of Object-Z and Statecharts to specify agent-based systems, and Griss et al. (2002), who use statecharts for defining agent behavior.

5.2.4.4 Strengths

One strength of statecharts is their visual representation: with little training, a person can understand what they mean. The hierarchical nature of statecharts can help in specifying large systems since the hierarchy helps to give high-level descriptions of the system as well as to drill down to levels of increasing detail.

5.2.4.5 Weaknesses

A weakness of statecharts is that development of the statecharts can be more time consuming than textual methods. In addition, statecharts lack the mathematical underpinning that other techniques like Z and CSP are inherently based on, which means that statecharts are not considered a formal language. For this reason, there have been several extensions that add process algebras or other formal languages to statecharts (see above). The hierarchical approach to statecharts can also make it difficult to get an overall picture of how a system works. If not developed well, statecharts can have some of the same understandability problems that FSMs have.

Statecharts also do not involve the notion of time. Transitions are considered to be executed instantaneously, which is not realistic in many real-time systems. With the ability to have multiple transitions occurring from a single state, the problem of race conditions can be embedded in a specification. Infinite loops are also possible, which requires statecharts to be checked for consistency. Statecharts have also been faulted as not being suited for object-oriented design because the broadcast property is incompatible with object-to-object method calls.

5.2.4.6 Tool Support

Statecharts are support by several tools. The STATEMATE case tool by iLogix was the first tool that supported statecharts and has been in use since the early 1990s. It allows statecharts to be created, simulated, analyzed and transformed into code. Another tool called Betterstate, developed by ISI (www.isi.com), supports graphical specification, automatic code generation, validation, and graphical debugging. It has also been integrated with Rational Rose.

5.2.5 Petri Nets

5.2.5.1 Summary

Petri Nets are a graph-based system for specifying asynchronous processes in concurrent systems. Petri nets are represented by the 5-tuple (P, T, I, O, M), where P is a set of places, T is a set of transitions, I is a set of inputs, O is a set of outputs, and M is a set of initial markings.

5.2.5.2 History

Petri Nets were developed in 1962 by Carl Adam Petri (1962) and were one of the first theories to address concurrency issues (Peterson, 1981; Peterson, 1977). Several variants of Petri nets have been developed over the years. Some of the variants include colored Petri nets, hierarchical Petri nets, object-oriented Petri nets, temporal Petri Nets, and G-Nets.

5.2.5.3 Applications

Petri nets have been used extensively to model concurrent systems. Petri nets have been used by several people to specify multi-agent systems (Ferber, 1999). Examples of using Petri Nets for specifying multi-agent systems include:

- Brown (1998) who used hierarchical colored and colored p Petri Nets to specify the NASA Lights-Out Ground Operations Systems,
- Bakam et al. (2000) who used Colored Petri Nets to study a multi-agent model of hunting activity in Cameroon,
- Shen (1998) who used Petri Nets to model mobile agents,
- Xu and Shatz (2001) have used a variant of Petri Nets called G-Nets to model buyer and seller agents in electronic commerce, and
- Weyns and Holvoet (2002) used Colored Petri Nets to study the social behavior of agents.

5.2.5.4 Strengths

Strengths of Petri Nets include the ability to simulate a model and do performance evaluation and verification and validation of a system. They have been widely used and therefore may be intuitive and easy to understand for many people.

5.2.5.5 *Weaknesses*

One Petri Nets weakness is the lack of modularity (van Linder et al., 1998). For complex systems, Petri Net models will become very large and will be difficult to understand or analyze. Another weakness is that many aspects of a system's behavior cannot be specified. For example, there is no way to express ordering between processes, synchronization of processes, or exclusive access to a process (for security).

5.2.5.6 *Tool Support*

There are a large number of tools available for Petri Nets. Tools provide a number of features, including graphical editors, token game animation, simulation, performance analysis, state spaces, place invariants, transition invariants, structural analysis, model checking, deadlock checking, optimization, net reductions, code generation, and reachability checking.

5.2.6 *X-Machines (XM)*

5.2.6.1 *Summary*

X-machines are based on finite state machines (FSM) except they have an internal memory state and transitions between states are labeled as functions which accept input symbols and output symbols based on the action of the function with reference to the internal memory state. X-machines can be thought of as typed FSMs with the set X acting as a memory and also having input and output tapes.

5.2.6.2 *History*

X-machines were developed by the mathematician Samuel Eilenberg in 1974 (1974). In 1986, Mike Holcome started using X-machines for biological specification purposes (1986a; 1986b) and then for system specifications (1988). X-machines have undergone modifications to specify a wider range of systems, such as Stream X-Machines (Gheorghe, 1998) that are used to control a family of distributed grammars, Communicating Stream X-Machines to better model concurrent systems (Barnard et al., 1996), and Object Communicating X-Machines (Barnard, 1999).

5.2.6.3 *Applications*

X-machines were originally used to describe intracellular biochemical organization and model cell biochemistry. Recently they have also been used to specify agent-based systems (Kefalas, 2000) and model the behavior of a bee colony (Gheorghe et al., 2001). X-Machines are also being investigated relative to emergent behavior of agent communities.

5.2.6.4 *Strengths*

X-Machines are more powerful than FSM and can specify complex systems more easily. In addition, the SXM Testing method, which is based on X-Machines, contains a set of rules that ensures complete functional testability of an implementation. The method does constrain the types of systems that can be implemented. X-machine specifications can also prove the correctness of an implementation with respect to the specification.

5.2.6.5 *Weaknesses*

The biggest weakness of X-machines is the lack of tool support. They also appear not to have gained wide support (perhaps due to the lack of tool support).

5.2.6.6 *Tool Support*

There is not a lot of tool support for X-machines. There are tools that can automatically convert an X-machine into Prolog, as well as model check an X-machine.

5.3 Logics

There are several types of logics that have been used and they are used for different applications. Propositional and predicate logics are used to represent factual information. For agents this may be a knowledge base or the agent's environment. These logics use and, or, not, implication, universal, and existential operators. Modal logics are used for different modes of truth, such as possibly true and necessarily true. Denotic logic describes what is obliged to be done. Dynamic logic is like modal logic but is action based. Temporal logic is the logic of time.

5.3.1 Temporal Logic

5.3.1.1 *Summary*

Temporal logic is used to express time-related aspects of systems. There is both modal and predicate approaches to temporal logic. In the original modal temporal logic created by Prior (1957) there were four additional operators to the standard logic operators:

- P, which stands for "It has at some time been the case that ...",
- F, which stands for "It will at some time be the case that ...",
- H, which stands for "It has always been the case that ...", and
- G, which stands for "It will always be the case that ...".

P and F are called weak tense operators, and H and G are called strong tense operators. G is sometimes denoted as _, F as ≥, H as _, and P as _.

In temporal logic, an expression is always true or will be true at some time in the future. There are two types of semantic models used timed specifications based on linear time and branching time. With linear time, a specification is a set of linear states with each state being part of a possible execution sequence (used in CSP traces). With branching time, a specification describes a tree structure of states, with each path in the tree a possible execution sequence (used in CCS). Other differences in temporal logics include discrete vs. dense, and moment-based vs. period-based times.

5.3.1.2 History

Temporal logic was developed by Arthur Prior in 1957 (1957) under the name of Tense Logic and has gone through several modifications by different people for application to different fields. The idea of temporal logic has also been added to other formal methods to give them the basis of time in those methods. Also a wide variety of temporal logics have been developed. Bellini, Mattolini, and Nesi give a good survey of temporal logics in (2000). Variations of temporal logics covered include Propositional Temporal Logic (PTL), Choppy Logic, Branching Time Temporal Logic (BTTL), Interval Temporal Logic, (ITL), Propositional Modal Logic of Time Intervals (PMLTI), Computational Tree Logic (CTL), Interval Logic (IL), Extended Interval Logic (EIL), Real-Time Interval Logic (RTIL), Timed Propositional Temporal Logic (TPTL), Real-Time Logic (RTL), Tempo Reale ImplicitO (TRIO), Metric Temporal Logic (MTL), and Time Interval Logic with Compositional Operators (TILCO). The differences in the different temporal logics range from expressiveness, availability of support tools for executability, and verifiability.

5.3.1.3 Applications

Temporal logic has been widely used for adding timing constraints and sequencing formation in real-time and artificial intelligence applications. In AI, it has been used to find a general framework for temporal representations (Allen, 1984). In specification and verification of concurrent programs, modal temporal logic has been successfully used to specify the timing of concurrent programs running on separate processors (Pnueli, 1977). Temporal logic has also been widely used to add timing to other formal specification languages like Z, Petri nets, Statecharts, and process algebras.

5.3.1.4 Strengths

Strengths of temporal logics are their ability to support proof of correctness related to timed constraints of systems. It has also been found that implementations from temporal logic tend to be efficient. Since temporal logics have been around for a number of years, they are well understood and there is a large class of researchers and practitioners familiar with them. There are also a number of versions of temporal logics available that can suite particular classes of problems. In addition, there are a wide number of tools available to support specifications.

5.3.1.5 Weaknesses

As with other logics, a major weakness is readability as the complexity of the system being specified increases, because of the unstructured nature of temporal logics (a specification is a set of predicates). Therefore, it is best used for simple or narrowly scoped properties of a complex system. There has been some work done to offset some of these problems by adding structure to temporal logic, and thereby increase its readability for larger specifications.

5.3.1.6 Tool Support

A number of tools are available to support various versions of temporal logics. The following is a partial list:

- PTL – the propositional temporal logic (PTL) tautology checker reads formulas in PTL and checks whether they are tautologies, i.e., always true no matter what truth values are assigned to the propositional variables at each instant of time. Typically, PTL can check whether a PTL-formula is a tautology and, if not, check whether the formula is satisfiable; and if the specification and implementation are written in PTL, verify that the implementation implies the specification.
- SteP – Stanford Temporal Prover does computer-aided formal verification of reactive, real-time, and integrated systems based on their temporal specification. It combines model checking with deductive methods to allow the verification of systems, including parameterized (N-component) circuit designs, parameterized (N-process) programs, and programs with infinite data domains.
- TLA+ - a language for writing Temporal Logic of Actions (TLA) specifications. There are three TLA+ tools available: a parser and syntax checker for TLA+ specifications, a model checker and simulator for a subclass of "executable" TLA+ specifications, and a program for typesetting TLA+ specifications.
- TLC – Temporal Logic Checker is a temporal logic assertion checker.
- TimeRover – provides a set of tools for temporal logic that includes temporal rule checking, runtime verification, high-level exception handling, and temporal simulation.

5.3.2 Real Time Logic (RTL)

5.3.2.1 Summary

Real Time Logic (RTL) is a predicate logic that relates the events of a system to their time of occurrence. RTL uses a discrete model of time that allows for reasoning about absolute timing (wall clock) properties in a system. RTL differs from modal temporal logic in that modal temporal logic uses relative timing of events for specifying time, which is qualitative. Since RTL uses a discrete model of time, it uses integers in RTL formulas. RTL uses an occurrence relation that assigns a time value to each occurrence of an event. The occurrence relation is denoted as R(e, i ,t), which means that the *i*-th occurrence of event *e* happens at time *t*. Predicates in RTL are made up from the occurrence relation as well as the mathematical relations ($=, <, \geq, >, \geq$).

5.3.2.2 History

RTL was developed by Jahanian and Mok (1986) in 1986. RTL has been extended by other researchers and combined with other logics. It has been combined with Z for specifying real time systems, temporal linear logic for specifying event-based logical systems, and Presburger arithmetic. The University of Texas Real-Time Systems Group (headed by Mok) supports RTL with ongoing research and the development of supporting tools, such as Modechart and Timetool.

5.3.2.3 Applications

The following are some of the applications in which RTL has been used:

- Verification of the planned performance of the safety-critical system functions of the NASA X-38 space station crew return vehicle multiprocessor system task structure,
- Axiomatic specification of communication protocols,
- The specification of the Real Rime Operating System (RTOS), and
- The verification of real time controllers.

5.3.2.4 Strengths

The primary strength of RTL over temporal logic is its ability to express the exact time an event or action will take place. In temporal logic, only relative times to other events or actions can be expressed. Like other logics, RTL also has the strength that it has the ability to support proof of correctness related to timed constraints of systems.

5.3.2.5 Weaknesses

As with other logics, a major weakness is readability as the complexity of the system being specified increases, because of the unstructured nature of temporal logics (a specification is a set of predicates). Therefore, it is best used for simple or narrowly scoped properties of a complex system.

5.3.2.6 Tool Support

The following are some tools that support RTL:

- MSP.RTL - a tool for producing real time schedulers based on real time logic.
- Modechart – a specification language, simulator, and verifier based on RTL.
- Multiway Decision Graphs (MDG) – a RTL functional verifier.

5.3.3 BDI Logics

5.3.3.1 Summary

Belief, Desires, and Intentions (BDI) is an agent architecture for describing agent behaviors (Georgeff and Lansky, 1987) based on the theory of action in humans by the philosopher M. Bratman (1987). To give formal semantics to BDI architectures, BDI logics were developed (Rao and Georgeff, 1991; 1995) that are multi-modal and extensions to the branching time logic CTL* [149]. The BDI logics allow the BDI architectures to be formally modeled and then proofs of correctness on BDI-based agents can be done. The BDI logics tend to be modal type logics and describe beliefs, desires, intentions and the plans (or plan library) that an agent can follow to achieve its intentions.

5.3.3.2 History

Rao and Georgeff (1991) initially introduced the idea of a logic for BDI architectures and subsequently there has been much work on evolving it, such as Wooldridge (1996a; 1996b;

2000) for plans, Padgham and Lambrix (2000) for capabilities in plans, and Singh and Asher (1990) for intentions. Different people have added on or concentrated on one aspect of the BDI logic to give it more formalism or extend it to cover specific aspects of a BDI agent specification.

5.3.3.3 Applications

BDI logic has been applied to a programming language called AgentSpeak(L) (Rao, 1996) which is based on a restricted first-order language with events and actions. Other BDI-based agent architectures based on BDI logic include the Java Agent Compiler and Kernel (JACK) (Howden et al., 2001), and dMARS (Distributed Multi-Agent Reasoning System) (1996).

5.3.3.4 Strengths

BDI logics tend to be very expressive and formal, so a large number of specifications can be written with a formal foundation. It also has the strength that it is a formal method and properties of the systems it specifies can be proven to be correct. Since BDI logics are based on BDI architectures, agent specifications can be easily mapped into a BDI architecture.

5.3.3.5 Weaknesses

The expressiveness of BDI logics make theorem proving and model checking much more difficult. As with other logics, a major weakness is also decreasing readability as the complexity of the system being specified increases, because of the unstructured nature of logics.

5.3.3.6 Tool Support

An AgentSpeak(L) interpreter is available free for downloading (AgentSpeak website). The interpreter will run AgentSpeak(L) and AgentSpeak(XL) programs. This allows agent specifications written in BDI logic to be executed. A restricted version of AgentSpeak(L), is AgentSpeak(F), which can be model checked. The restricted nature of AgentSpeak(F) allows it to be converted to Promela and then run on the model checker Spin (Holzmann, 1991).

5.3.4 KARO Logic

5.3.4.1 Summary

The KARO (Knowledge, Abilities, Results and Opportunities) logic (Hustadt et al., 2000; van Linder et al., 1998) is a formal system based on modal logic for reasoning about and specifying the behavior of intelligent multi-agent systems. KARO formalizes the notion of knowledge contained within agents and the agents' possible execution of actions. The KARO framework allows agents to reason about their own and other agent's abilities to perform actions, the possible results of those actions, and the availability of the opportunities to take those actions. KARO combines both dynamic and epistemic logic into a single modal logic with additional modal operators, and adds the notion of abilities.

5.3.4.2 *History*

KARO was proposed by van Linder, van der Hoek, and Meyer in 1998 (1998). So it is a relatively new logic and framework. Additional work is also being done on KARO that includes, Hustadt, et al. (2000) who are developing automated proof methods for KARO, Meyer, et al. (2000) who are working on linking KARO to agent programming languages, Aldewereld (2002) who has worked on extending KARO from single-agent to multi-agent, and Dixon et al. (2000) who have applied Computational Tree Logic (CTL) instead of dynamic logic.

5.3.4.3 *Applications*

KARO was developed specifically for modeling agent-based systems. At the time of this writing, specific applications of KARO to specific multi-agent systems has not been found.

5.3.4.4 *Strengths*

KARO is based on modal logic, which has historically been used to describe knowledge, belief, time, obligation, desire and other attributes that apply to agent-based systems. The use of modal logic can be more concise that first-order logics. In addition, modal logic lends itself to logical proofs of correctness and it tends to be more intuitive than first-order logic representations, while at the same time being able to be reducible to first-order logic and those first-order methods and techniques can still be applied.

5.3.4.5 *Weaknesses*

A weakness of KARO that has been discussed is its use of dynamic logic (Dixon et al., 2000). Dynamic logic can become complex in practical applications and specifications can become undecidable and incomplete. A fix to this has been to replace dynamic logic with CTL.

5.3.4.6 *Tool Support*

Since KARO is very new, there are very few tools available for it, with tool support still being developed and proposed. One tool XProof (Valk, 1998) has been used to construct a theorem prover for KARO (Valk, 1999).

5.4 Other Approaches

The following is a list of other approaches that are being used to specify and verify agent-based or swarm-based systems.

5.4.1 *Artificial Physics*

5.4.1.1 *Summary*

Artificial physics (AP) is based on using properties from physics to model constraints and interaction between agents (Spears, W. and Gordon, D., 1999, Shehory et al., 1999). Control of agents in an AP framework is mapped to one of minimizing potential energy (PE). If constraints are violated or performance degrades, PE increases, triggering a reactive response. Global behaviors are automatically computed via local interactions between agents. Given a set of initial conditions and desired global behavior, sensors, effectors, and local force laws can be determined for the desired global behavior to emerge.

As an example of artificial physics, suppose a set of agents are treated as physical particles. Particles move in response to the virtual forces that are exerted upon them by their neighbors - in essence the particles act as if they were part of a molecular dynamics simulation. Particles have a position, mass, velocity, and momentum. Friction is included, for self-stabilization. The net action of the system of particles is to reduce potential energy in a continuously changing virtual potential field.

5.4.1.2 History

The work that is most related to artificial physics is referred to as "swarm intelligence" (Hiebeler, 1994) and "social potential fields" (Reif and Wang, 1994). In swarm intelligence the swarm distribution is determined via a system of linear equations describing difference equations with periodic boundary conditions. The social potential fields method relies on a force-law simulation that is similar to that found in molecular dynamics.

Physicomimetics is also similar to work in robotics, such as "potential field" and behavior-based approaches. Potential field (PF) approaches are used for robot navigation and obstacle avoidance (Khatib, 1986). The emphasis is on a single robot. In a manner similar to physicomimetics, PF approaches model a goal position as an attractive force, while obstacles are modeled with repulsive forces. PF computes force vectors by taking the gradient of an entire potential field, which is very computationally intensive. AP uses force vectors directly, and thus has lower run-time computational overhead. Furthermore, unlike the standard PF approach, AP relies on inter-agent forces, as well as environmental forces.

Behavior-based approaches (Balch, 1998) derive vector information in a fashion similar to physicomimetics. Particular behaviors such as "aggregation" and "dispersion" have some similarity to the attractive and repulsive forces in physicomimetics. However, behavior-based approaches do not make use of potential fields or forces. Rather, they deal directly with velocity vectors and heuristics for changing those vectors

5.4.1.3 Applications

Artificial physics has been used to generate a variety of vehicle formations in simulation and it has demonstrated the capability of clustering agents into subgroups (Spears and Gordon, 1999). Others have used physicomimetics for physical simulations of self-assembly. Schwartz et al. (1998) investigated the self-assembly of viral capsids in a 3D solution. Winfree (1998) has investigated the self-assembly of DNA double-crossover molecules on a 2D lattice. Shehory et al. (1999) used physics-based systems for modeling emergent behavior.

5.4.1.4 Strengths

Physics-based descriptions can be very efficient to execute. Most computer hardware is made to do these types of computations, which can be much more efficient than rule-based or knowledge-based systems. Artificial physics is also very good for representing reactive behavior since the physics-based equations can be quickly executed. Physics-based approaches could also support learning similar to genetic programming, where different formulas are used and then modified, with the best performing formula used. Formulas could also be modified through other reflective processes that examine the performance of a system and modify the formulas for better performance.

5.4.1.5 Weaknesses

Physics-based systems may not be good for deliberative processes in agents. Deliberation often involves examining models or doing symbolic manipulation, which physics-based systems may not be easy to develop to do. Physics-based descriptions of agent-based interaction can be difficult to understand by people who do not have a physics background. Also, most intelligent systems are developed from a logic background, so the logic may have to be translated into a physics-based representation.

5.4.1.6 Tool Support

Tool support for artificial physics is based on current tools for visualizing physical properties. No specific tools have been developed.

5.4.2 Software Cost Reduction (SCR)

5.4.2.1 Summary

SCR is a formal method based on tables for specification and analysis of black-box behavior of complex safety-critical systems (Bharadwaj and Heitmeyer, 1999a). A toolset, called SCR* is available to help automate as much of the method as possible. SCR describes both the system's environment (which is usually nondeterministic) and the system's behavior (usually deterministic). The system is represented as a state machine and the environment is represented as a nondeterministic event generator. An SCR specification represents the state machine's transitions as a set of tables.

The system environment specification includes monitored variables (environmental quantities that the system monitors) and controlled variables (environmental quantities that the system controls) (Bharadwaj and Heitmeyer, 1999b). The system behavior is represented by two relations, NAT and REQ. NAT represents the natural constraints on the system behavior (such as physical laws and the system environment constraints). REQ represents the relationships between the monitored and the controlled quantities that the system must maintain. Tables are used to describe transitions, events, and conditions of a state machine for the system.

5.4.2.2 History

SCR was originally developed in 1978 at the Naval Research Lab (NRL) to document the requirements of the flight program of the Navy's A-7E aircraft. It has been used on a number of safety-critical control system as well as other systems. It has also been extended to specify hardware/software co-design and co-validation (Bharadwaj and Heitmeyer, 1999a).

5.4.2.3 Applications

SCR has been used on a number of projects, including the Navy's A-7E aircraft and the C-130J Flight Program, as well as telephone networks, communications security devices, control systems for nuclear power plants, the International Space Station, the Deep Space-1 spacecraft, and military and civilian flight controls. It has also been used by a number of organizations to specify requirements, including NASA IV&V Center, JPL, Grumann, AT&T, Ontario Hydro, Rockwell, and Lockheed.

5.4.2.4 Strengths

Strengths of SCR include its easy interpretation of the specification through use of the table format. The method scales up well, as was demonstrated in its use on a project that was implemented with 230K lines of Ada. In addition, the formal basis of the method also enables proof of correctness and use in model checkers.

5.4.2.5 Weaknesses

SCR's reliance on a state machine model could make it difficult for modeling large numbers of parallel systems, such as swarm-based systems. Like state machines, SCR is inherently sequential, so it may not be good by itself to express concurrent systems. Modifications, such as those that have been made to FSMs, may enable SCR to better handle concurrent systems.

5.4.2.6 Tool Support

To support the SCR method, there is an integrated toolset called SCR* (Heitmeyer et al., 1998). The toolset includes a specification editor, dependency graph browser, consistency checker, a simulator, and verification tools (including Spin, TAME, and Salsa).

5.4.3 Mathematical Analysis

5.4.3.1 Summary

Mathematical analysis uses mathematical formulas to model or specify a system and then uses mathematical techniques for analyzing the resulting system specification. From the mathematical specification system properties can be proven correct or that they remain in bounds. For swarm-based systems, mathematical models can be either developed at the agent level (microscopic) or the swarm level (macroscopic). Some techniques used are physics-based approaches (see artificial physics); others have used stochastic approaches (Lerman and Galstyan, 2001). Mathematical analysis can be used to study the dynamics of swarms and predict long-term behavior as well as such things as efficiency and steady state characteristics without having to do simulations. It also allows parameters to be found that determine swarm behavior and how the actions of a single member of the swarm affect the entire swarm.

5.4.3.2 History

Mathematical analysis has been used in many different fields. It has been used in biology to study insects and model their macroscopic and microscopic behavior, molecular dynamics, cellular automata and particle hopping. Due to its wide use in a number of fields, there are many reference materials and mathematicians that are experienced in this type of modeling and analysis.

5.4.3.3 Applications

Mathematical analysis has been used for both multi-agent systems and swarm-based systems. Lerman (2000; Lerman and Galstyan, 2001) used a stochastics-based method for modeling a multi-agent system that formed coalitions. Sheory, et al. (1999) used physics-based systems and applied physics-based mathematical techniques to the analysis of multi-agent systems.

5.4.3.4 Strengths

The strengths of a mathematical approach are that it affords a precise model of the system and entails a wide range of analytical methods to analyze the system. In addition, there are a large number of tools and techniques available with which to perform the analysis, and there is a long history of these types of analysis, giving assurance that they are also well understood.

5.4.3.5 Weaknesses

Mathematical models can be difficult to develop and difficult to understand. In addition, mathematical models do not always reflect the complexity of artificial systems, and artificial systems do not always reflect the orderliness of nature.

5.4.3.6 Tool Support

Standard mathematical tools can be used for mathematical analysis, such as Mathematica or Matlab. These types of tools are available from a wide range of sources and have excellent visualization capabilities so that properties of the system can be seen.

5.4.4 Game Theory

5.4.4.1 Summary

Game theory uses mathematical analysis to analyze decision making in conflict situations. It provides for the study of human behavior and choice optimization. It uses probability and other mathematical techniques for analyzing situations and coming up with the best choice. It has been used extensively in economics, politics, management, biology, and social sciences, as well as other sciences, to describe interacting entities. It has recently been applied to agent-based and swarm-based systems as a way of modeling and analyzing agents and their societies.

5.4.4.2 History

Game theory has been traced back to the time of 0-500 AD in the Babylonian Talmud, which is a compilation of ancient law and tradition and serves as the basis of Jewish religious, criminal, and civil law. In the Talmud, a marriage contract problem is described that relates to a modern cooperative game. There are also references to game theory emerging in the 1700s, 1800s, and early 1900s, with the first textbook on game theory being published in 1952.

5.4.4.3 Applications

There have been a number of researchers who used game theory to analyze and verify agent-based systems: a sampling can be found in (Parsons and Wooldridge, 2002). Rudnianski and Bestougeff (2000) have used games of deterrence to analyze and simulate agent-based systems. Others (Osborne, 2003; Tomlin et al., 1998) have used game theory as a way to model agent-based systems as a non-cooperative, zero-sum dynamic game with self-interested agents where the actions of agents are modeled as disturbances to the other agents. Some of these are modeled as 2-player games and others as n-player games. These models have been applied to sharing limited resources (such as airport runways or automated highways) or for collision avoidance. Once a model for a system is developed, properties of the system can be proven correct by showing the model maintains those properties. Game theory has also been used to study biological (Rowe, 1997) and swarm-based systems (Challet and Zhang, 1997).

5.4.4.4 Strengths

The strengths of game theory are its mathematical foundation, the fact that it can be used to describe large entities (such as economies) and that properties of the system can be proven correct. In addition, many agent-based systems can be viewed as games so the models are a natural expression of those systems.

5.4.4.5 Weaknesses

Weaknesses of game theory for multi-agent or swarm-based systems are that the game theory specification does not always reflect the logical or procedural description of the behavior of the system. This means that validation can be more difficult. In addition, the mathematical game theory specification is unfamiliar to many engineers or software specifiers.

5.4.4.6 Tool Support

There are several tools available to assist in developing or analyzing game theory models including:

- Gambit – a library of game theory software and tools for the construction and analysis of games,
- CSWiz – a set of Excel spreadsheet add-ons for solving optimization or equilibrium problems,
- Optimizers for solving a wide range of linear and non-linear equations, available from a number of sources,

- Agent-based simulation (e.g. the Swarm software) tools, available for modeling large game-based simulations,
- Several tools for visualizing curves and equations for game and economic theories

5.4.5 UML

5.4.5.1 Summary

The Unified Modeling Language (UML) is a language for specifying, visualizing and documenting models of software systems (Booch et al., 1999). UML has twelve different diagram types divided into three classes: structural diagrams, behavior diagrams, and model management diagrams. UML does not specify a particular methodology for using the language, so it is methodology independent, though many of the UML tools use a particular methodology.

5.4.5.2 History

UML was developed from three different modeling languages: the Grady Booch method, James Rumbaugh's Object Modeling Technique 2 (OMT-2) method, and Ivar Jacobson's Object-Oriented Software Engineering (OOSE) method (Alhir, 2002). UML supports object-oriented analysis and design in addition to use cases and other system specification techniques.

5.4.5.3 Applications

UML is an industry standard and is widely used. It is likely the most widely used software specification and design technique in use today.

UML has been used to specify agent-based systems. One of the main thrusts for using UML for agents is Agent UML (AUML) (Bauer et al., 2000), which is a standard now being worked on by the Foundation for Intelligent Physical Agents (FIPA) Modeling Technical Committee (Modeling TC). The FIPA AUML standard has class diagrams for specifying the internal behavior of agents and the external environment, and has interaction diagrams. The Modeling TC has also identified modeling areas for Multi- vs. single agent, use cases, social aspects, temporal constraints, deployment and mobility, and workflow/planning, as well as other areas. One of the main challenges of AUML is adding semantics to UML that reflect the autonomy, social structures, and asynchronous communication aspects of agents.

Other work on extending UML for agent specification includes:

- The Agent-Object-Relationship Modeling Language (AORML) (Wagner, 2003), which enhances UML sequence diagrams for specifying agent interactions,
- MASSIF (Mentges, 1999), which uses standard UML sequence diagrams for describing interactions,
- Oechslein, et al. (2001) uses UML Object Constraint Language (OCL) to extend UML to formally specify agents using UML,
- Role-Based Modeling Method for Multi-Agent Systems (RoMAS) (Yan et al., 2003), which uses UML use cases for defining system events and interactions,
- Extension of use cases for specifying agent behavior (Heinze et al., 2000),

- Extensions to UML (Papasimeon and Heinze, 2001) to support the Java Agent Compiler and Kernel (JACK website) agents.

5.4.5.4 Strengths

The major strengths of UML are its wide use in industry, its standardization by Object Management Group (OMG), and the large range of tool support. This means that specifications of agents done in UML will be able to be understood by a wide range of software specifiers and designers, and can be manipulated and analyzed by a wide range of tools.

5.4.5.5 Weaknesses

One of the weaknesses of UML has been its lack of a formal mathematical foundation for its semantics. This means that properties of a UML specification or design cannot be proven correct, and therefore means that it is not a formal method. There has been work in this area to formally define the semantics of UML and make it a formal specification language (OOPSLA'98, 1998; Kazuki, 2001).

5.4.5.6 Tool Support

There are a large number of UML tools available commercially from a range of well-known, mainstream tool companies (e.g., Borland and IBM), so there is excellent support. These tools help developers with modeling systems through diagramming at the specification and design level, as well as additional tools for skeleton code generation based on design diagrams and tools for testing the end system.

5.4.6 Integrated Approaches

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 able to be specified the same way past systems have been developed. Because of this, it appears that many people are combining formal methods into integrated approaches to address some of 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, agent-based, and swarm-based systems.

- Communicating X-Machines (Barnard et al., 1996),
- CSP-OZ – a combination of CSP and Object-Z (Fischer, 2000),
- Object-Z and Statecharts (Bussow et al., 1998),
- Timed Communicating Object Z (Gala and Baker, 1999),

- Temporal B (Bonnet et al., 1995),
- Timed CSP (Reed and Roscoe, 1987),
- Temporal Petri Nets (Temporal Logic and Petri Nets) (Bakam et al., 2000),
- ZCCS – a combination of Z and CCS (Galloway and Stoddart, 1997).

From the wide interest in integrated approaches, it appears that current techniques are not sufficient to model and verify distributed and concurrent systems.

6. Specification Approaches Used for Social, Swarm and Emergent Behavior

The following is a summary of specification techniques that have been used for specifying social, swarm, and emergent behaviors. All of these approaches are based on the formal approaches described above.

- Weighted Synchronous Calculus of Communicating Systems (WSCCS), a process algebra, was used by Tofts to model social insects (1991). WSCCS was also used in conjunction with a dynamical systems approach for analyzing the non-linear aspects of social insects (Sumpter et al., 2001).
- X-Machines have been used to model cell biology (Fournet and Gonthier, 1996; Holcombe, 1986) and modifications, such as Communicating Stream X-Machines (Barnard et al., 1996) also have potential for specifying swarms.
- Dynamic Emergent System Modeling Language (DESML) (Kiniry, 1998), which is a variant of UML, has been suggested for modeling emergent systems.
- Cellular automaton (von Neumann, 1996) has been used to model systems that exhibit emergent behavior (such as land use).
- Simulation approaches are also being investigated to determine emergent behavior and then use a modeling technique to model the behavior. These approaches do not model emergent behavior beforehand, only after the fact.

7. Comparison and Selection of Formal Methods to Specify ANTS

7.1 Comparison of Formal Methods

Tables 1 through 3 compare the formal methods described in the above sections. Table 1 compares the methods described in section 5, relative to several characteristics: support for concurrency, support for algorithm specification, tool support, formal foundation (and the type used), and whether the method has been used to specify agent-based or swarm-based systems in the past.

Table 1: Comparison of candidate formal methods for intelligent swarms.

Name	Concurrency Support	Algorithm Support	Tool Support	Formal Basis	Used in Agent-Based Specs.	Used in Swarm-Based Specs.
Artificial Physics	Yes	Yes	Yes	Yes-(Mathematical)	Yes	Yes (limited)
B	No	Yes	Yes	Yes	Yes	No

				(Set Theory/ Pred. Logic)		
BDI Logic	Yes	No	Yes	Yes (Logic)	Yes	No
CCS	Yes	No	Yes	Yes (Algebraic)	Yes	Yes (WSCCS)
CSP	Yes	No	Yes	Yes (Algebraic)	Yes	No
Finite State Machines	No	Yes	Yes	Yes (Formal Lang.)	Yes	No
Game Theory	Yes	No	Yes	Yes (Mathematical)	Yes	Yes
I/O Automata	Yes	Yes	Yes	Yes (Formal Lang.)	Yes	No
KARO	Yes	No	No (limited)	Yes (Logic)	Yes	No
Mathematical Analysis	Yes	No	Yes	Yes (Mathematical)	Yes	Yes
Petri Nets	Yes	No	Yes	Yes	Yes	No
Pi Calculus	Yes	No	Yes	Yes (Algebraic)	Yes	No
Real Time Logic	Yes	No	Yes	Yes (Logic)	No	No
SCR	No	Yes	Yes	Yes (Formal Lang.)	No	No
Statecharts	Yes	No	Yes	No (Formal Lang.)	Yes	No
Temporal Logic	Yes	No	Yes	Yes (Logic)	Yes	No
UML	Yes	Yes	Yes	No	Yes	No
X-Machines	No	Yes	No (limited)	Yes (Formal Lang.)	Yes	No
Z	No	Yes	Yes	Yes (Set Theory/ Pred. Calc.)	Yes	No

Table 2 compares the integrated formal methods relative to the same factors and characteristics in Table 1. For the tool support, a yes is entered only if there is integrated tool support for the combined languages, not separate tools for each language in the integrated.

Table 2: Comparison of integrated formal methods.

Name	**Concurrency Support**	**Algorithmic Support**	**Tool Support**	**Formal Basis**	**Used in Agent-Based Specs.**	**Used in Swarm-Based Specs.**
Comm. Stream X-Machines	Yes	Yes	No	Yes	Yes	Yes
CSP-OZ	Yes	Yes	No	Yes	Yes	No

Object-Z and Statecharts	Yes	Yes	No	Yes	Yes	No
Temporal B	Yes	Yes	No	Yes	Yes	No
Temporal Petri Nets	Yes	No	No	Yes	Yes	No
Timed Communicating Object Z	Yes	Yes	No	Yes	Yes	No
Timed CSP	Yes	No	Yes	Yes	Yes	No
ZCCS	Yes	Yes	No	Yes	Yes	No

Table 3 compares methods that have been used for modeling or specifying swarm-based systems (computer-based or biological based).

Table 3: Comparison of formal methods used for swarm specifications.

Name	**Concurrency Support**	**Algorithmic Support**	**Tool Support**	**Formal Basis**	**Emergent Behavior Analysis**	**Used in Swarm-Based Specs.**
Cellular Automaton	Yes	Yes	Yes	Yes (FSM)	No	Yes
Com. X-Machines	Yes	Yes	No	Yes (Formal Language)	No	Yes
Unity Logic/ Swarm	Yes	Yes	Yes (limited)	Yes (Logic/FSM)	No	Yes
WSCCS	Yes	No	Some (Prob. Wrkbn)	Yes (Process Algebra)	Yes (Markov Chain)	Yes

8. Evaluation of Methods for Specifying and Analyzing Emergent Behavior

The following is a list of methods that show promise for specification and verification of swarm-based systems:

- Artificial Physics,
- Communicating Sequential Processes,
- WSCCS,
- X-Machines,
- BDI Logic,
- AUML,
- ZCCS and
- Timed Communicating Object Z.

Based on the results of the survey, four of the above formal methods were selected to be used for a sample specification of part of the ANTS mission. These methods were: the process algebras CSP and WSCCS, X-Machines, and Unity Logic. These will be used to describe an ANTS virtual experiment. CSP was chosen as a baseline specification method because the team has had significant experience and success (Rouff et al., 2000 and Hinchey et al., 2001) in specifying agent-based systems with CSP. WSCCS and X-Machines were chosen because they have already been used for specifying emergent behavior by others, apparently with some success. Unity Logic was also chosen because it had been successfully used for specifying concurrent systems and affords a logic-based specification, which offered a contrast to the other methods.

DESML, Cellular Automata, Artificial Physics, and simulation approaches were not used even though they had been used for specifying or evaluating emergent behavior. DESML, though very interesting, was not used because it had not been used or evaluated outside of the thesis it was developed under (though we may be revisiting it at a future time). Cellular Automata were not selected because they did not have any built in analysis properties for emergent behavior and because they have been primarily used for simulating emergent systems (as described in the previous section). Though not used for the specification, it too may be revisited to further examine its strengths. Artificial physics, which is very promising, was not selected because of the newness of the approach and because of the translation that must be done between physics and software behavior. Lastly, simulation techniques were not used due to the fact that verification cannot be undertaken using simulation. This is because there could be emergent or other undesirable behaviors occurring that are not visible or do not become apparent during a simulation, but may exist nonetheless. A formal technique is designed to find exactly these kinds of errors.

The following describes the reasons for selecting CSP, WSCCS, Unity Logic and X-Machines.

8.1.1 CSP

CSP is a process algebra and is very good at specifying the process protocols between and within the spacecraft and analyzing the result for race conditions. Being able to evaluate a system for race conditions is very important, particularly in swarm-based systems, which are highly parallel. From a CSP specification, reasoning about the specification can be done to determine race conditions as well as converted into a model checking language for running on a model checker.

8.1.2 WSCCS

WSCCS provides a process algebra that takes into account the priorities and probabilities of actions performed by the leader and other ANTS spacecraft. It further provides a syntax and large set of rules for predicting and specifying the choices and behaviors of the Leader, as well as a congruence and syntax for determining whether two automata are equivalent. All of this in hand, WSCCS can be used to specify the ANTS spacecraft and to reason about and even predict the behavior of one or more spacecraft. This robustness affords WSCCS the greatest potential for specifying emergent behavior in the ANTS swarm. What it lacks towards that end is an ability to track the goals and model of the ANTS mission in a memory. This may be achieved by blending the WSCCS methods with the memory aspects of X-Machines.

8.1.3 Unity Logic

Unity Logic provides a logical syntax equivalent to simple 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. However, it does not appear to be rich enough to allow ease of specification and validation of more abstract concepts such as mission goals. This same simplicity, however, may make it a good tool for specifying and validating the actual Reasoning programming (as opposed to Reasoning process) portion of the ANTS Leader spacecraft, when the need arises. In short, specifying emergent behavior in the ANTS swarm will not be accomplished well using Unity Logic.

8.1.4 X-Machines

X-Machines provide a highly executable environment for specifying the ANTS spacecraft. It allows for a memory to be kept and it allows for transitions between states to be seen as functions involving inputs and outputs. This allows us to track the actions of the ANTS spacecraft as well as write to memory any aspect of the goals and model. This ability makes X-Machines highly effective for tracking and effecting changes in the goals and model. However, X-Machines does not provide any robust means for reasoning about or predicting behaviors of one or more spacecraft, beyond standard propositional logic. This will make specifying emergent behavior difficult.

8.1.5 Summary

Based on the above evaluation, the following are some of the properties of a formal method needed for specifying swarm-based systems:

- Ability to model and reason about aggregate behavior based on future actions of the individual agents of a swarm (such as provided by WSCCS)
- Ability to model and reason about concurrent processes for detection of race conditions (such as provided by CSP and Unity Logic)
- Ability to model states of an agent of the swarm to assure correctness (such as provided by statecharts, X-Machines or Z)
- Ability to model and reason about persistent information so adaptive behavior can be verified (such as provided by X-Machines).

A blending of the above methods seems to be the best approach for specifying swarm-based systems and analyzing emergent behavior of these systems. Blending the memory and transition function aspects of X-Machines with the priority and probability aspects of WSCCS 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. The idea of merging the above methods is currently being furthered studied as well as adding some of the properties of logic and cellular automata.

9. Conclusion

Swarm-based missions are becoming more important to NASA to enable new science to be performed. These types of missions have many positive attributes but represent a change in paradigm from current types of single spacecraft missions. Swarms require new types of verification and validation techniques to assure their correct operation. To overcome their nondeterministic nature, high degree of parallelism, intelligent behavior, and emergent behavior, new kinds of verification methods need to be used.

This report has presented the results of an investigation into formal method techniques that might be applicable to future swarm-based missions and that can verify their correctness. It also analyzed the properties of these methods to determine the needed attributes of a formal specification language to predict and verify emergent behavior of future NASA swarm-based systems.

Future work will concern the development of a new formal method based on blending aspects of the several formal methods as well as adding additional mathematical techniques from other areas of mathematics that may prove fruitful for predicting the emergent behavior of swarms. We intend to use ANTS and another NASA swarm-based mission to test the capabilities of the resulting formal method. We expect that the resulting formal method could become the basis of other specification languages to support specification and analysis of future swarm-based systems.

10. Acknowledgements

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.

11. References

[1] Abadi, M. and Gordon, A.D. A calculus for cryptographic protocols: The Spi calculus. Jornal of Information and Computation, 143:1-70, 1999.
[2] Abrial, J.R. The B Book: Assigning Programs to Meanings. Cambridge University Press. 1996.
[3] AgentSpeak(L). http://protem.inf.ufrgs.br/cucla/.
[4] Aldewereld, H. Rational Teams: Logical Aspects of Multi-Agent Systems. Master's Thesis, Utrecht University. May, 2002.
[5] Alhir, S. Guide to Applying the UML. Springer, 2002.

[6] Allen, J.F. Towards a general theory of action and time. Artificial Intelligence, 1984, Volume 23, PP 123-154.

[7] Andrews, J.H., Day, N.A. and Joyce, J.J. Using Formal Description Technique to Model Aspects of a Global Air Traffic Telecommunications Network. In 1997 IFIP TC6/WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification (FORTE/PSTV). Edited by Higashino, T. and Togashi, A. Chapman and Hall, pp 417-432, November 1997.

[8] ANTS Prospecting Asteroids Mission (ANTS/PAM). NASA Goddard Space Flight Center. http://ants.gsfc.nasa.gov/.

[9] ANTS team. Protocol for ANTS Encounters. NASA GSFC, Code 695.

[10] Astesiano, E. and Zucca, E. Parametric channels via label expressions in CCS. Theoretical Computer Science, 33:45-64, 1984.

[11] Attie, P.C. and Lynch, N.A. Dynamic Input/Output Automata: A Formal Model for Dynamic Systems. In Proceedings of International Conference on Concurrency Theory, pages 137-151, 2001.

[12] Bakam, I., Kordon, F., Le Page, C., and Bousquet, F. Formalization of a Spatialized Multiagent Model Using Coloured Petri Nets for the Study of an Hunting Management System. In Proceedings of First International Workshop on Formal Approaches to Agent-Based Systems. Springer, LNAI 1871. Greenbelt, MD, April 2000.

[13] Balch, T. Behavioral Diversity in Learning Robot Teams. PhD Thesis, Georgia Institute of Technology, 1998.

[14] Barnard, J. Object COMX: Methodology Using Communicating X-Machine Objects. Journal of Object-Oriented Programming (JOOP). Nov-Dec. 1999.

[15] Barnard, J., Whitworth, J. and Woodward, M. Communicating X-machines. Journal of Information and Software Technology, 38(6), 1996.

[16] Bauer B., Muller J. P. and Odell J. Agent UML: A Formalism for Specifying Multiagent Software Systems. In Proceedings of ICSE 2000 Workshop on Agent-Oriented Software Engineering AOSE 2000, Limerick, 2000.

[17] Bellini, P., Mattolini, R., and Nesi, P. Temporal Logics for Real-Time System Specification. ACM Computing Surveys, 32(1):12-42, March, 2000.

[18] Beni, G. The Concept of Cellular Robotics. In Proceedings of the 1988 IEEE International Symposium on Intelligent Control, pp 57-62, Los Alamitos, CA 1988. IEEE Computer Society Press.

[19] Beni, G. and Want, J. Swarm Intelligence. In Proceedings of the Seventh Annual Meeting of the Robotics Society of Japan, pp 425-428, Tokyo, Japan, 1989, RSJ Press.

[20] Bharadwaj, R. and Heitmeyer, C. Hardware/Software Co-Design and Co-Validation Using the SCR Method. In Proceedings of the IEEE International High Level Design Validation and Test Workshop (HLDVT'99), Nov. 1999. (a)

[21] Bharadwaj, R. and Heitmeyer, C. Model Checking Complete Requirements Specifications Using Abstraction. Automated Software Engineering, 6, 37-68 (1999). (b)

[22] Bonabeau, E. and Theraulaz, G., Swarm smarts. Scientific American, pp. 72-79, March 2000.

[23] Bonabeau, E., Dorigo, M. and Theraulaz, G. Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York, 1999.

[24] Bonabeau, E., G. Theraulaz, et. al.et al. "Self-organization in Social Insects", Trends in Ecology and Evolution, 1997, vol. 12, pp. 188-193.
[25] Bonnet, L., Florin, G., Duchien, L., Seinturier, L. A Method for Specifying and Proving Distributed Cooperative Algorithms. DIMAS'95. November 1995.
[26] Booch, G., Rumbaugh, J. and Jacobson, I. The Unified Modeling Language User Guide. The Addison-Wesley Object Technology Series. Addison-Wesley, Reading, MA. 1999.
[27] Borgia, R., Degano, P., Priami, C., Leth, L. and Thomsen, B. Understanding mobile agents via a non-interleaving semantics for Facile. In Proceedings of SAS'96, Volume 1145 of LNCS. Editors Cousot, R. and Schmidt, D.A. Springer, pages 98-112, 1996.
[28] Bowen, J.P. and Hinchey, M.G. High-Integrity System Specification and Design. Series on Formal Approaches to Computing and Information Technology (FACIT), Springer, 1999.
[29] Bratman, M. Intentions, Plans, and Practical Reason. Harvard University Press, Cambridge. 1987.
[30] Brown, B. High-Level Petri Nets for Modeling Multi-Agent Systems. MS project report, Dept. of Computer Science, North Carolina A&T State University, Greensboro, NC, 1998.
[31] Bussow, R. and Geisler, R. and Klar, M. Specifying Safety-critical Embedded systems with Statecharts and Z: A Case Study. In Proceedings of the International Conference on Fundamental Approaches to Software Engineering. Edited by Astesiano. LNCS 1382, Springer-Verlag, Berlin. Pages 71--87, 1998.
[32] Butler, M. csp2B: A Practical Approach to Combining CSP and B. Technical Report DSSE-TR-99-2. Department of Electonics and Computer Science, University of Southampton. 1999.
[33] Cardelli, L. and Gordon, A.D. Mobile ambients. Proceedings of FoSSaCS'98. Editored by Nivat, M. LNCS, Volume 1378, pages 140-155, Springer, 1998.
[34] Carlson, S. Artificial Life: Boids of a Feather Flock Together. Scientific American, November 2000.
[35] Challet, D. and Zhang, Y.-C. Emergence of Cooperation and Organization in an Evolutionary Game, Physica A 246, 407. 1997.
[36] Chandy, K.M. and Misra, J. Parallel Program Design: A Foundation. Addison-Wesley Publishing Company. 1988.
[37] Chefter, A.E. A Simulator for the IOA language. Mater's thesis, MIT Department of EECS, May 1998.
[38] Clare, E. and Wing, J. Formal Methods: State of the Art and Future Directions. Report by the Working Group on Formal Methods for the ACM Workshop on Strategic Directions in Computing Research, ACM Computing Surveys, vol. 28, no. 4, December 1996, pp. 626-643.
[39] Clark, P. E., Curtis, S. A. and Rilee, M. L. ANTS: Applying a New Paradigm to Lunar and Planetary Exploration. Solar System Remote Sensing Symposium, Pittsburg, 2002.
[40] Coleman, D., Hayes, F. and Bear, S. Introducing Objectcharts, or How to Use Statecharts in Object Oriented Design. IEEE Transactions on Software Engineering, January 1992, pp. 9-18.
[41] Craigen, D., Gerhart, S. and Ralston, T.J. An International Survey of Industrial Applications of Formal Methods (Volume 1: Purpose, Approach, Analysis and Conclusions, Volume 2: Case Studies). NIST Technical Report NIST GCR 93/626-V1 and NIST GCR 93-626-V2, 1993.

[42] Curtis, S. A., J. Mica, J. Nuth, G. Marr, M. Rilee, and M. Bhat. ANTS (Autonomous Nano-Technology Swarm): An Artificial Intelligence Approach to Asteroid Belt Resource Exploration. International Astronautical Federation, 51st Congress, October 2000.
[43] Curtis, S., Truszkowski, W., Rilee, M., and Clark, P. ANTS for the Human Exploration and Development of Space. IEEE Aerospace Conference, 2003.
[44] Curtis, S., Truszkowski, W., Rilee, M., and Clark, P. ANTS for the Human Exploration and Development of Space. IEEE Aerospace Conference, 2003.
[45] Curtis, S.A. , Mica, J., Nuth, J., Marr, G., Rilee, M., Bhat, M. ANTS (Autonomous Nano Technology Swarm): An Artificial Intelligence Approach to Asteroid Belt Resource Exploration. International Astronautical Federation, 51st Congress, October 2000.
[46] d'Inverno, M. and Luck, M. Understanding Agent Systems. Springer-Verlag, 2001.
[47] d'Inverno, M. and Luck, M. Formal Agent Development: Framewok to System. In Proceedings of First International Workshop on Formal Approaches to Agent-Based Systems. Springer, LNAI 1871. Greenbelt, MD, April 2000.
[48] d'Inverno, M., Fisher, M., Lomuscio, A., Luck, M., de Rijke, M., Ryan, M. and Wooldridge, M. Formalisms for multi-agent systems. The Knowledge Engineering Review, 3(12), 1997.
[49] Dixon, C., Fisher, M. and Bolotov, A. Resolution in a Logic of Rational Agency. Proceedings of the 14th European Conference on Artificial Intelligence (ECAI 2000). IO Press, 2000.
[50] Eilenberg, S. Automat, Languages and Machines, Vol. A. Academic Press, 1974.
[51] Emerson, E.A. and Halpern, J.Y. 'Sometimes' and 'not never' revisted: on branching time versus linear time temporal logic. Journal of the ACM, 33(1):151-178, 1986.
[52] Engberg, U. and Nielsen, M. A calculus of communicating systems with label-passing. Technical Report DAIMI PB-208, Computer Science Department, University of Aarhus, Denmark, 1986.
[53] Esterline, A., Rorie, T. Using the p-Calculus to Model Multiagent Systems. . In Proceedings of First International Workshop on Formal Approaches to Agent-Based Systems. Springer, LNAI 1871. Greenbelt, MD, April 2000.
[54] Felder, M., Mandrioli, D. and Morzenti, A. Proving properties of real-time systems through logical specifications and Petr net models. IEEE Transactions on Software Engineering, 20(2):127-141, February 1994.
[55] Ferber, J. Mulit-Agent Systems: An Introduction to Distributed Artificial Intelligence. Addison-Wesley. 1999.
[56] FIPA. Foundation for Intelligent Physical Agents. http://www.fipa.org.
[57] Fischer, Clemens. Combination and Implementation of Processes and Data: from CSP-OZ to Java. Ph.D. Dissertation, Fachbereich Informatik, Universitat Oldenburg. 2000.
[58] Fournet, C. and Gonthier, G. The reflexive chemical abstract machine and the join-calculus. In Proceedings of POPL'96, editors Steel, J.G. ACM, pages 372-385, Jan. 1996.
[59] Gala, A.K. and Baker, A.D. Multi-Agent Communication in JAFMAS. In Workshop on Specifying and Implementing Conversation Policies, Third International Conference on Autonomous Agents (Agents '99). Seattle, Washington, 1999.
[60] Galloway, A.J. and Stoddart, W.J. An operational semantics for ZCCS. In M. Hinchey and S. Liu, editors, the IEEE International Conference on Formal Engineering Methods (ICFEM'97), pages 272--282, Hiroshima, Japan, November 1997. IEEE Computer Society Press.

[61] Georgeff, M.P. and Lansky, A.L. Reactive reasoning and planning. In Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87), pages 677-682, Seattle, WA, 1987.

[62] Gheorghe, M. Stream X-Machines and Grammar Systems. Department of Computer Science, Faculty of Mathematics, Bucharest University, STr. Academiei 14, 70109 Bucharest, Romania. 1998.

[63] Gheorghe, M., Holcombe, M. and Kefalas, P. Computational Models of Collective Foraging. In Proceedings of the 4th International Workshop on Information Processing in Cells and Tissues, IPCAT2001, Lueven, Belgium, August 2001.

[64] Gordon, D. APT Agents: Agents That are Adaptive, Predictable, and Timely. In Proceedings of First International Workshop on Formal Approaches to Agent-Based Systems. Springer, LNAI 1871. Greenbelt, MD, April 2000.

[65] Griss, M.L., Fonseca, S., Cowan, D. and Kessler, R. SmartAgent: Extending the JADE Agent Behavior Bodel. HP Laboratories Technical Report HPL-2002-18. Palo Alto, CA, 2002.

[66] Grobauer, B. and Muller, O. From I/O Automata to Timed I/O Automata. In Proceedings of the 12th International Conference on Theorem Proving in Higher Order Logics, TPHOLs'99. Editors Bertot, Y., Dowek, G., Paulin-Mohring, Ch., and Théry, L.Lecture Notes in Computer Science, Nice, France, 1999, pages 273-290. Springer Verlag.

[67] Haddadi, A. and Sundermeyer, K. Belief-Desire-Intention Agent Architectures. In Foundations of Distributed Artificial Intelligence, G. M. P. O'Hare and N. R. Jennings (eds.). Wiley, New York. 1996, pp. 169-185.

[68] Hamberger, T. Integrating theorem proving and model checking in isabelle/ioa. Technical report, T.U. Munich, Agust 1999.

[69] Harel, D. and Naamad, A. The STATEMATE Semantics of Statecharts. ACM Transactions on Software Engineering Methodology, October 1996, pp. 293-333.

[70] Harel, D. On Visual Formalisms. Communications of the ACM 31(5):514-530, May 1988.

[71] Harel, D. Pnueli, A., Schmidt, J.P., and Sherman, R. On the formal semantics of statecharts. In Procedings of the 2nd IEEE Symposium on Logic in Computer Science, Ithaca, N.Y., June 22-24. IEEE Press, New York, 1987, pp. 54-64.

[72] Harel, D. Statecharts: A visual formalism for complex systems. Science of Computer Programming, 8:231-274, 1987.

[73] Heinze, C., Papasimeon, and Goss, S. Specifying agent Behavior with use cases. In Proceedings of Pacific Rim Workshop on Multi-Agents, PRIMA2000, 2000.

[74] Heitmeyer, C., Kirby, J. Jr., Labaw, B., and Bharadwaj, R. SCR*: A toolset for specifying and analyzing software requirements. In Proceedings of Computer Aided Verification, 10th International Conference, CAV '98. Alan J. Hu, Moshe Y. Vardi (Eds.), Vancouver, BC, Canada, June 28 - July 2, 1998. Lecture Notes in Computer Science 1427, Springer 1998.

[75] Hiebeler, D. The Swarm Simulation System and Individual-based Modeling. In proceedings of Decision Support 2001: Advanced Technology for Natural Resource Management. Toronto, Ontario, Canada, Sept. 1994.

[76] Hilaire, V., Koukam, A., Gruer, P. and Muller, J.P. Formal Specification and Prototyping of Multi-Agent Systems. First International Workshop on Engineering Societies in the Agents' World. August 2000, Berlin.

[77] Hilaire, V., Koukam, A., Gruer, P. and Muller, J.P. Formal Specification and Prototyping of Multi-Agent Systems. In proceedings of Engineering Societies in the Agents World, Springer-Verlag, LNAI 1972, 2000.

[78] Hilderink, G., Broenink, J., and Bakkers, A. A new Java Thread model for Concurrent Programming of Real-time Systems. Real-Time Magazine, pp 30-35, January, 1998.

[79] Hinchey, M. and Bowen, J. Industrial-Strength Formal Methods in Practice. Springer. 1999.

[80] Hinchey, M. Jarvis, S. Concurrent Systems: Formal Development in CSP. McGraw-Hill. 1995.

[81] Hinchey, M., Rash, J. and Rouff, C. Verification and Validation of Autonomous Systems. IEEE/NASA Software Engineering Workshop. November 2001.

[82] Hoare, C.A.R. Communicating Sequential Processes. Communications of the ACM, 21(8):666-677, August, 1978.

[83] Hoare, C.A.R. Communicating Sequential Processes. Prentice Hall, 1985.

[84] Hoare, C.A.R. Communication Sequential Processes. Prentice Hall Series in Computer Science, Hemel Hempstead & Englewood Cliffs. 1985.

[85] Holcombe, M. Mathematical models of cell biochemistry. Technical Report CS-86-4. 1986. Dept of Computer Science, Sheffield University, United Kingdom.

[86] Holcombe, M. Towards a formal description of intracellular biochemical organization. Technical Report CS-86-1. 1986. Dept of Computer Science, Sheffield University, United Kingdom.

[87] Holcombe, W.M.L. X-machines as a Basis for System Specification. Software Engineering Journal, 3(2), 1988, 69-76.

[88] Holzmann, H. J, Design and Validation of Computer Protocols, Prentice Hall Software Series, Englewood Cliffs, NJ, 1991.

[89] Howden, N., Rönnquist, R., Hodgson, A. and Lucas, A. JACK — Summary of an Agent Infrastructure. 5th International Conference on Autonomous Agents. Montreal, Canada, 2001.

[90] Hustadt, U., Dixon, C., Schmidt, R., Fisher, M., Meyer, J., and van Wiebe, H. Verification within the KARO Agent Theory. In Proceedings of First International Workshop on Formal Approaches to Agent-Based Systems. Springer, LNAI 1871. Greenbelt, MD, April 2000.

[91] INMOS, Editor. Occam Programming Manual. Prentice-Hall, Inc., 1984.

[92] Iyengar, J., Truszkowski, W and Mills, F. Describing Intelligent Agent Behaviors. Journal of International Information Management, 10(2), 99-77. 2002.

[93] J.-R. Abrial, J.-R., Biirger, E., and Langmaack, H., editors. Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler, volume 1165 of Lecture Notes in Computer Science. Springer-Verlag, 1996.

[94] Jahanian, F. and Mok, A.K. Safety Analysis of Timing Properties in Real-Time Systems. IEEE Transactions on Software Engineering. SE-12(9) pp. 890-904 (Sept. 1986).

[95] James L. Peterson, Petri Nets, ACM Computing Surveys (CSUR), v.9 n.3, p.223-252, Sept. 1977.

[96] Jazayeri, M., Ghezzi, C., Hoffman, D., Middleton, D., and Smotherman, M. CSP/80: a language for communicating sequential processes. IEEE Compcon, 1980.

[97] Jones, G. and Goldsmith, M. Programming in occam2. Prentice-Hall, 1988.

[98] Josephs, M.B. Receptive Process Theory. Acta Informatica 29(1): 17-31, 1992.

[99] Kawabe, Y., Mano, K., and Kogure, K. The Nepi^2 Programming System: A ≥-calculus-Based Approach to Agent-Based Programming. In Proceedings of First International Workshop on Formal Approaches to Agent-Based Systems. Springer, LNAI 1871. Greenbelt, MD, April 2000.

[100] Kawabe, Y., Mano, K., and Kogure, K. The Nepi^2 Programming System: A p-Calculus-Based Approach to Agent-Based Programming. In Proceedings of First International Workshop on Formal Approaches to Agent-Based Systems. Springer, LNAI 1871. Greenbelt, MD, April 2000.

[101] Kazuki, M. The Semantic Analysis of UML by Behavioral Specification. Master Thesis. Japan Advanced Institute of Science and Technology, School of Information Science. 2001.

[102] Kefalas, P. Modelling an Agent Reactive Architecture with X-Machines. 2000. Dept of Computer Science, CITY Liberal Studies, 13 Tsimiski Str., 54624 Thessaloniki, Greece.

[103] Khatib, O. Real-time Obstacle Avoidance for Manipulators and Mobile Robots. International Journal of Robotics Research, 5(1):90-98, 1986.

[104] Kimiaghalam, B., Homaifar, A., Esterline, A. A Satechart Framework for Agent Roles that Captures Expertise and Learns Improved Behavior. In Proceedings of Second International Workshop on Formal Approaches to Agent-Based Systems (FAABS II). Springer, LNCS. Greenbelt, MD, October 2002.

[105] Kiniry, J.R. The Specification of Dynamic Distributed Component Systems. Masters' Thesis, CS-TR-98-08 1998 California Institute of Technology, Computer Science Department.

[106] Lea, D. Concurrent Programming in Java: Design Principles and Pattern (2nd Edition). Addison-Wesley, 1999.

[107] Lerman, K. Design and Mathematical Analysis of Agent-Based Systems. In Proceedings of First International Workshop on Formal Approaches to Agent-Based Systems. Springer, LNAI 1871. Greenbelt, MD, April 2000.

[108] Lerman, K. and Galstyan, A. A General Methodology for Mathematical Analysis of Multi-Agent Systems. USC Information Sciences Technical Report ISI-TR-529, 2001.

[109] Luna, F. and Stefansson, B. Economic Simulations in Swarm: Agent-Based Modelling and Object Oriented Programming. Kluwer Academic. 2000.

[110] Lygeros, J., Godbole, D.N. and Sastry, S. Verified Hybrid Controllers for Automated Vehicles. IEEE Transactions on Automatic Control, 43(4), April, 1998.

[111] Lynch, N.A. and Tuttle, M.R. Hierarchical Correctness Proofs for Distributed Algorithms. In Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing, ACM, Pages 137-151, August 1987.

[112] Lynch, N.A., Segala, R. and Vaandrager, F.W. Hybrid I/O automata. Information and Computation, to appear. Available as Technical Report MIT-LCS-TR-827d, MIT Laboratory for Computer Science, Cambridge, MA 02139, January 13, 2003.

[113] Mahony, B. and Dong, J.S. Timed Communicating Object Z. IEEE Transactions on Software Engineering, 26(2):150-177, Feb 2000.

[114] McIlraith, S. Modeling and Programming Devices and Web Agents. In Proceedings of First International Workshop on Formal Approaches to Agent-Based Systems. Springer, LNAI 1871. Greenbelt, MD, April 2000.

[115] Mentges, E. Concepts for an agent-based framework for interdisciplinary social science simulation. Journal of Artificial Societies and Social Simulation, 2(2), 1999.

[116] Menzies, T., Cukic, B., and Singh, H. Agents Talking Faster. In Proceedings of First International Workshop on Formal Approaches to Agent-Based Systems. Springer, LNAI 1871. Greenbelt, MD, April 2000.

[117] Meyer, J.J., de Boer, F., van Eijk, R., Hindriks, K. and van der Hoek, W. On Programming KARO Agents. In Cunningham, J. Gabby (Ed.), Proc. Int. Conf. on Formal and Applied Practical Reasoning (FAPR2000). London: Imperial College.

[118] Milner, R. Communication and Concurrency. Prentice-Hall, 1985.

[119] Milner, R. A Calculus of Communicating Systems. LNCS, Vol. 92. Springer-Verlag, Berlin, 1980.

[120] Milner, R., Parrow, J. and Walker, D. A calculus of mobile processes, part I/II. Journal of Inforamation and Computation, 100:1-77, Sept. 1992.

[121] Milner, R., Parrow, J. and Walker, D. A Calculus of Mobile Processes, Parts I and II. Journal of Information and Computation, 100:1-77, 1992.

[122] Moller, F. and Stevens, P. Edinburgh Concurrency Workbench User Manual. http://www.dcs.ed.ac.uk/home/cwb/.

[123] Muthiayen, D. and Alagar, V.S. Formalizing UML for Rigorous Software Development. In Proceedings of the OOPSLA'98 Workshop on Formalizing UML. Eds, Andrade, L., Moreira, A., Deshpande, A. Stuart Kent, S. 1998.

[124] Nayak, P. Pandurang, et al. 1999. Validating the DS1 Remote Agent Experiment. In Proceedings of the 5th International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS-99).

[125] Nicola, R.D. and Segala, R. A Process Algebraic View of I/O Automata. Theoretical Computer Science, 138:391-423, 1995.

[126] Oechslein, C., Klugl, F., Herrler, R. and Puppe, F. UML for Behavior-Oriented Multi-Agent Simulations. In: Dunin-Keplicz, B., Nawarecki, E.: From Theory to Practice in Multi-Agent Systems. LNAI; Second International Workshop of Central and Eastern Europe on Multi-Agent Systems, CEEMAS 2001 Cracow, Poland, September 26-29, 2001, Revised Papers, Springer 2002, Heidelberg.

[127] OOPSLA'98 Workshop on Formalizing UML. Why? How? Addendum to the 1998 proceedings of the conference on Object-oriented programming, systems, languages, and applications. Vancouver, B.C., Canada. October 18, 1998.

[128] Osborne, M.J. An Introduction to Game Theory. Oxford University Press, 2003.

[129] Padgham, L. and Lambrix, P. Agent Capabilities: Extending BDI Theory. In Proceedings of Seventeenth National Conference on Artificial Intelligence - AAAI 2000, Aug 2000, p 68-73.

[130] Papasimeon, M. and Heinze, C. Extending the UML for Designing JACK Agents. In Proceedings of the Australian Software Engineering Conference (ASWEC 01), Canberra, Australia, August 26-27, 2001.

[131] Parsons, S. Gymtrasiewicsz, P. and Wooldridge, M. Game Theory and Decision Theory in Agent-Based Systems. Kluwer, 2002.

[132] Peterson, J. L. Petri Net Theory and the Modeling of Systems. Prentice Hall, Englewood Cliffs, N.J. 1981.

[133] Petri, C.A. Kommunikation mit Automaten. Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962. Second Edition, New York: Griffiss Air Force Base, Technical Report RADC-TR-65--377, Vol.1, 1966, Pages: Suppl. 1, English translation.

[134] Pierce, B.C. and Turner, D.N. Pict: A programming language based on the pi-calculus. In Proof, Language and Internacion: Essays in Honour of Robin Milner. Editors G. Plotkin, C. Stirling, and M. Tofte. 1999.

[135] Pnueli, A. The Temporal Logic of Programs. Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, 1977, pages 46-67.

[136] Prior, A.N. 1957. Time and Modality. Oxford: Oxford University Press.

[137] Rao, A.S. AgentSpeak(L): BDI Agents speak out in a logical computable language. In Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World. Editors, W. Van de Velde and J. W. Perram. Springer-Verlag, LNAI Volume 1038. 1996.

[138] Rao, A.S. and Georgeff, M.P. BDI Agents: From Theory to Practice. Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, CA. June, 1995.

[139] Rao, A.S. and Georgeff, M.P. Modeling Rational Agents within a BDI-Architecture. Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91). Editors James Allen and Richard Fikes and Erik Sandewall. Morgan Kaufmann, 1991.

[140] Reed, G.M. and Roscoe, A.W. Metric Spaces as Models for Real-Time Concurrency. In Proceedings, Workshop on the Mathematical Foundations of Programming Language Semantics, pp 331-343, Lecture Notes in Computer Science, Vol. 298, Springer-Verlag, 1987.

[141] Reif, J. and Wang, H. Social Potential Fields: A Distributed Behavioral Control for Autonomous Robots. In Proceedings of WAFR'94. San Francisco, California, February 1994.

[142] Reynolds, C. W. (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model. Computer Graphics, 1987, 21(4), pp 25-34.

[143] Rilee, M.L., Boardsen, S.A., Bhat, M.K. and Curtis, S.A. Onboard Science Software Enabling Future Space Science and Space Weather Missions. 2002 IEEE Aerospace Conference Big Sky, Montana, 9-16 March 2002.

[144] Roscoe, A.W., et al. Hierarchical compression for model-checking CSP or How to check 10^20 dining philosophers for deadlock. Proceedings of the TACAS symposium, 1995. Springer Verlag, LNCS 1055

[145] Rouff, C., Rash, J., Hinchey, M. Experience Using Formal Methods for Specifying a Multi-Agent System. Sixth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2000) September 11-15, 2000.

[146] Rowe, G.W. Game Theory in Biology. In Physical Theory in Biology: Foundations and Explorations. Edited by Lumsden, C.J., Trainor, L.E.H. and Brandts, W.A. World Scientific, 1997.

[147] Rudnianski, M. and Bestougeff, H. Modeling Task and Teams through Game Theoretical Agents. In Proceedings of First International Workshop on Formal Approaches to Agent-Based Systems. Springer, LNAI 1871. Greenbelt, MD, April 2000.

[148] S. Chalmers and P.M.D. Gray. BDI Agents and Constraint Logic. AISB Journal Special Issue on Agent Technology, 1(1):21-40, 2001.

[149] Savage, M. and Askenaki, M. Arborscapes: A Swarm-based Multi-agent Ecological Disturbance Model. Working paper 98-06-056. 1998. Santa Fe, NM: Santa Fe Institute.

[150] Schuman, S.A. and Pitt, D.H. Object-oriented subsystem specification. In L. G. L. T. Meertens, editor, *Program Specification and Transformation*, pages 313--341. North-Holland, 1987.

[151] Schwartz, R., Shor, P., Prevelige, P., and Berger, B. Local Rules Simulation of the Kinetics of Virus Capsid Self-Assembly. Biophysics, 75:2626-2636, 1998.

[152] Shehory, O., Sarit, K., and Yadgar, O. Emergent cooperative goal-satisfaction in large-scale automated-agent systems. Artificial Intelligence, 1999.

[153] Shen, C.Y.L. Behavior Modeling for Mobile Agents. UCI Undergraduate Research Journal, 1998.

[154] Sing, M. and Asher, N. Towards a formal theory of intentions. In Logics in AI, Editor van Eijck, J. Springer-Verlag, LNAI Vol 478, pp 472-486. 1990.

[155] Singh, M., Anand, R., and Georgeff, M. Formal Methods in DAI: Logic-Based Representation and Reasoning. In Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Edited by Weiss, G. The MIT Press. 1999.

[156] Smith, M. Formal verification of TCP and T/TCP, Ph.D. Thesis. MIT, Department of Electrical Engineering and Computer Science. September 1997.

[157] Sowmya, A. and Ramesh, S. Extending Statecharts with Temporal Logic. IEEE Transactions on Software Engineering, Vol 24, No 3, March 1998, pp 216-229.

[158] Spears, W. and Gordon, D., "Using artificial physics to control agents," in Proceedings of the IEEE International Conference on Information, Intelligence, and Systems, November, 1999, Charlotte, NC.

[159] Spivey, J. M. The Z Notation: A Reference Manual. Prentice Hall, Hemel Hempstead, 2nd edition. 1992.

[160] Sumpter, D.J.T., Blanchard, G.B. and Broomhead, D.S. Ants and Agents: A Process Algebra Approach to Modelling Ant Colony Behaviour. Bulletin of Mathematical Bilogy. 2001, 63, 951-980.

[161] Surka, D., Campbell, M., Schetter, T. Controlling Multiple Satellite Constellations Using the TEAMAgent System. In Proceedings of First International Workshop on Formal Approaches to Agent-Based Systems. Springer, LNAI 1871. Greenbelt, MD, April 2000.

[162] Tadashi Araragi and Paul Attie and Idit Keidar and Kiyoshi Kogure and Victor Luchangco and Nancy Lynch and Ken Mano. On Formal Modeling of Agent Computations. In Proceedings of First International Workshop on Formal Approaches to Agent-Based Systems. Springer, LNAI 1871. Greenbelt, MD, April 2000.

[163] The dMARS V1.6.11 System Overview, Technical Report, Australian Artificial Intelligence Institute (AAII), 1996.

[164] Tofts, C. Describing social insect behaviour using process algebra. Transactions on Social Computing Simulation. 1991. 227-283.

[165] Tomlin, C., Pappas, G. and Sastry, S. Conflict Resolution for Air Traffic Management: A Case Study in Multi-Agent Hybrid Systems. IEEE Transactions on Automatic Control, 43(4), April, 1998.

[166] Uselton, A.C. and Smolka, S.A. A Process Algebraic Semantics for Statecharts via State Refinement. In Proceedings of IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET), June 1994. (a)

[167] Uselton, A.C. and Smolka, S.A. A Compositional Semantics for Statecharts using Labeled Transition Systems. Proceedings of CONCUR'94 – Fifth International Conference on Concurrency Theory. Uppsala, Sweden, August 1994. (b)

[168] Valk, J. Generic Theorem Proving. Master's Thesis, Delft University of Technology, 1998.

[169] Valk, J., Tonino, H. Bos, A. and Witteveen, C. Automated Theorem Proving for the KARO-architecture. Foundations and applications of Collective Agent Based Systems (CABS). In ESSLLI 99 Workshop, Utrecht, 1999.

[170] van de Mortel-Fronczak, J.M., Rooda, J.E. and de Greeff, L.A.J. Real-time Concurrent Programming as a Structured Approach to Modelling of Manufacturing Systems. In R.D. Schraft et al (eds.), Proceedings of FAIM'95, Stuttgart, Germany, June 1995, pp. 247–258.

[171] van Linder, B., van der Hoek, W. and Meyer, J-J. Ch. Formalizing abilities and opportunities of agents. Fundamenta Informaticae, 34(1,2):53–101, 1998.

[172] Victor, B. and Moller, F. The Mobility Workbench – A Tool for the Pi-Calculus. Technical Report DoCS 94/95, Department of Computer Science, Uppsala University, Sweden, 1994.

[173] Vollmer, J. and Hoffart, R. Modula-P, a language for parallel programming: Definition and implementation on a transputer network. In Proceedings of the 1992 International Conference on Computer Languages ICCL'92, Oakland, California, pages 54-64. IEEE, IEEE Computer Society Press, April 1992.

[174] Vollmer, J. and Hoffart, R. Modula-P, a language for parallel programming: Definition and implementation on a transputer network. In Proceedings of the 1992 International Conference on Computer Languages ICCL'92, Oakland, California, pages 54-64. IEEE, IEEE Computer Society Press, April 1992.

[175] von Neumann, John. Theory of Self-Reproducing Automat. University of Illinois Press, Urbana, Illinois. 1996. Edited and completed by Burks, A.W.

[176] Wagner, G. The Agent-Object-Relationship Meta-Model: Towards a Unified Conceptual View of State and Behavior. Information Systems 28:5 (2003), pp. 475-504.

[177] Walker, D. Objects in the pi-calculus. Journal of Information and Computation, 116(2):253-271, 1995.

[178] Weiss, G. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. The MIT Press. 1999.

[179] Weyns, D., Holvoet, T. A Colored Petri Net for a Multi-Agent Application. In Proceedings of Second Workshop on Modeling of Objects, Components, and Agents. Pages 121-140, Aarhus, Denmark, August 2002.

[180] Winfree, E. Simulations of Computing by Self Assembly. DI-MACS: DNA-Based Computers, June 1998.

[181] Wooldridge, M. A Logic of BDI Agents with Procedural Knowledge. Working Notes of 3rd ModelAge Workshop: Formal Models of Agents. Editor, Pierre-Yves Schobbens. Sesimbra, Portugal. 1996.

[182] Wooldridge, M. Practical reasoning with procedural knowledge: A logic of BDI agents with know-how, Practical Reasoning. Proceedings of FAPR'96 (D. Gabbay and H-J. Ohlbach, eds.), LNAI, vol. 1085, Springer Verlag, 1996, pp. 202--213.

[183] Wooldridge, M. Reasoning about Rational Agents. Intelligent Robot and Autonomous Agents Series. MIT Press, Cambridge, MA. 2000.

[184] Wrench, K.L. CSP-I: An implementation of communicating sequential processes. Software-Practice and Experience, 18(6):545-560, June 1988.

[185] Wu, S.H, Smolka, S.A., and Stark, E.W. Composition and behaviors of probabilistic I/O automata. Theoretical Computer Science, 176(1-2):1-38, 1997.

[186] Xu, H. and Shatz, S.M. An Agent-based Petri Net Model with Application to Seller/Buyer Design in Electronic Commerce. In Proceedings of Fifth International Symposium on Autonomous Decentralized Systems. Dallas, Texas, March 2001.

[187] Yan, Q., Shan, L., Mao, X. and Qi, Z.. RoMAS: a role-based modeling method for multi-agent system. In Proceedings of Second International Conference on Active Media Technology. May 2003.