2005

Safety in Multiagent Systems by Policy Randomization

Praveen Paruchuri  
*University of Southern California*, paruchur@usc.edu

Milind Tambe  
*University of Southern California*, tambe@usc.edu

Sarit Kraus  
*Bar-Ilan University, Ramat Gan*, sarit@cs.biu.il

Fernando Ordonez  
*University of Southern California*, fordon@usc.edu

Don M. Dini  
*University of Southern California*, ddini@usc.edu

Follow this and additional works at: [http://research.create.usc.edu/published_papers](http://research.create.usc.edu/published_papers)

Recommended Citation  
[http://research.create.usc.edu/published_papers/43](http://research.create.usc.edu/published_papers/43)

This Article is brought to you for free and open access by CREATE Research Archive. It has been accepted for inclusion in Published Articles & Papers by an authorized administrator of CREATE Research Archive. For more information, please contact gribben@usc.edu.
Second International Workshop on Safety and Security in Multiagent Systems

University of Utrecht

26th July 2005

Sponsored by The Boeing Company

Mike Barley, Fabio Massacci, Haralambos Mouratidis and Amy Unruh (editors)
Preface

As intelligent autonomous agents and multi-agents systems applications become more pervasive, it becomes increasingly more important to understand the risks associated with using these systems. Incorrect or inappropriate agent behaviour can have harmful effects including financial cost, loss of data, and injury to humans or systems.

Thus, security and safety are two central issues when developing and deploying such systems. We refer to a multiagent system's security as the ability of the system to deal with threats that are intentionally caused by other intelligent agents and/or systems, and the system's safety as its ability to deal with any other threats to its goals.

In complex and rich environments, such as multiagent system environments, it is often necessary to involve the agents of the system in achieving some of these design goals, by making the goals explicit for the agent itself. For example, the agent must be aware of user-specified safety conditions if it is going to avoid violating them. This often means that an agent needs to be able to identify, assess, and mitigate many of the risks it faces. This is particularly true when the agent is going to be deployed in dangerous environments without immediate user input; for example, command of a spacecraft where communication with mission control involves considerable delays.

Moreover, agents often integrate such activities as deliberately planning to achieve their goals, dynamically reacting to obstacles and opportunities, communicating with other agents to share information and coordinate actions, and learning from and/or adapting to their environments. Because agents are often situated in dynamic environments, these activities are often time-sensitive. These aspects of agents make the process of developing, verifying, and validating safe and secure multiagent systems more difficult than for conventional software systems. Hence, new and different techniques and perspectives are required to assist with the development and deployment of such systems.

This workshop will serve as a forum to gather academics, researchers, practitioners, and students from the fields of safety, security and multiagent systems.
Programme Committee
The Programme Committee was chaired by Mike Barley, Fabio Massacci, Haralambos Mouratidis and Amy Unruh and comprised:

Eduardo Alonso (City University of London - UK)
James Bailey (University of Melbourne - Australia)
Steve Chien (JPL - USA)
Subrata Das (Charles River Analytics - USA)
Paolo Giorgini (University of Trento - IT)
Henry Hexmoor (University of Arkansas – USA)
Jan Jurjens (TUM - Germany)
Tom Karygiannis (NIST - USA)
Ed Kazmierczak (University of Melbourne – Australia)
Igor Kotenko (SPIIRA - Russia)
Vic Lesser (University of Massachusetts - USA)
Antonio Manna (University of Malaga – Spain)
Gordon Manson (University of Sheffield - UK)
Fabio Martinelli (CNR/IIT - IT)
Eduardo Fernandez Medina Paton (Universidad de Castilla-La Mancha - Spain)
David Musliner (Honeywell - USA)
Marian Nodine (Telcordia – USA)
Lin Padgham (RMIT - Australia)
Stefan Poslad (Queen Mary University of London – UK)
Anita Raja (University of North Carolina - USA)
Marco Roveri (IRST - Italy)
Nora Erika Sanchez (ITESM - Mexico)
Paul Scerri (CMU – USA)
Diana Spears (University of Wyoming - USA)
Michael Weiss (Carleton University - Canada)
Eric Yu (University of Toronto - Canada)
Wei Zhang (Boeing - USA)
Table of Contents

Towards Using Simulation to Evaluate Safety Policy for Systems of Systems
Robert Alexander, Martin Hall-May, Georgios Despotou, Tim Kelly ............... 5

A compositional proof system for agent behaviour
Dominique Fournier, Bruno Mermet, Gaele Simon ................................. 22

Multiagent modeling and simulation of agents’ competition for network
resources availability
Igor Kotenko, Alexander Ulanov............................................................ 27

Analyzing Dangers in Multiagent Rescue using DEFACTO
Janusz Marecki, Nathan Schurr, Milind Tambe, Paul Scerri ....................... 44

Extending the Buddy Model to Secure Variable Sized Multi Agent Communities
John Page, Arkady Zaslavsky, Maria Indrawan .................................... 59

Fault-Tolerant Information Sharing Networks
Jisun Park, K. Suzanne Barber .............................................................. 76

Safety in Multiagent Systems by Policy Randomization
Praveen Paruchuri, Don M Dini, Milind Tambe, Fernando Ordonez,
Sarit Kraus ......................................................................................... 89

The Use of MAS Rescue Simulation to Assess the Effectiveness of Planning for
Urban Disasters: A Preliminary Study
Tomoichi Takahashi, Nobuhiro Ito ....................................................... 102

Safe Agents in Space: Preventing and Responding to Anomalies in the Autonomous
Sciencecraft Experiment
Daniel Tran, Steve Chien, Gregg Rabideau, Benjamin Cichy ...................... 107

An Overview of SECMAP Secure Mobile Agent Platform
Suat Ugurlu, Nadia Erdogan ............................................................... 122
Towards Using Simulation to Evaluate Safety Policy for Systems of Systems

Robert Alexander, Martin Hall-May, Georgios Despotou, and Tim Kelly

Department of Computer Science
University of York, York, YO10 5DD, UK.
{robert.alexander, martin.hall-may, georgios.despotou, tim.kelly}@cs.york.ac.uk

Abstract. The increasing role of Systems of Systems (SoS) in safety-critical applications establishes the need for methods to ensure their safe behaviour. One approach to ensuring this is by means of safety policy — a set of rules that all the system entities must abide by. This paper proposes simulation as a means to evaluate the effectiveness of such a policy. The requirements for simulation models are identified, and a means for decomposing high-level policy goals into machine-interpretable policy rules is described. It is then shown how the enforcement of policy could be integrated into a simple agent architecture based around a blackboard. Finally, an approach to evaluating the safety of a system based on simulation runs is outlined.

1 Introduction

Large-scale military and transport Systems of Systems (SoS) present many challenges for safety. Attempts to define the term ‘SoS’ have been controversial — attempts can be found in [1] and [2]. It is easy, however, to identify uncontroversial examples, Air Traffic Control and Network Centric Warfare being the most prominent. These examples feature mobile components distributed over a large area, such as a region, country or entire continent. Their components frequently interact with each other in an ad-hoc fashion, and have the potential to cause large-scale destruction and injury. It follows that for SoS that are being designed and procured now, safety has a high priority.

In order to ensure the safe behaviour of SoS, the behaviour of the individual system entities must be controlled, as must the overall behaviour that emerges from their individual actions and interactions. One way to achieve this is to impose a system-wide safety policy, which describes the rules of behaviour which agents in the system must obey. Due to the geographically distributed nature of many entities, the policy typically cannot be directly enforced by some external controller (as in security policy); rather, the entities must comply with it individually. Evaluating the effectiveness of such a decentralised policy is not straightforward in a complex system, since the overall safe behaviour only emerges from the behaviour of the entities themselves.

An SoS is a complex multi-agent system (MAS) in which many entities have a mobile physical presence. The agents within this MAS are in themselves very
complex. This complexity means that formal analysis methods and conventional system safety techniques are not adequate for evaluating the safety of an SoS. In this paper, we propose simulation as a viable alternative.

Once the ‘baseline’ model of an SoS has been evaluated, the analysis can be repeated for a range of candidate safety policies. Based on the results of this analysis, candidate policies can be modified or discarded, and the process repeated until a satisfactory safety policy is found. We believe that simulation is a valuable tool for the development of SoS safety policy.

1.1 Structure of this Paper

The following section describes the problems faced in analysing and ensuring the safety of SoS. Section 3 introduces the concept of safety policy. Section 4 describes what is required from a simulation engine and simulation model. Section 5 outlines an approach to implementation of an SoS as a multi-agent simulation that satisfies the identified requirements. Section 6 highlights how safety cannot be considered in isolation from other dependability attributes. Section 7 describes some issues and challenges that need to be tackled, and section 8 presents a summary.

2 The Problem of SoS Safety Analysis

The Oxford English Dictionary [3] defines safety as “The state of being safe; exemption from hurt or injury; freedom from danger.” In system safety engineering, it is common to restrict the definition of ‘hurt or injury’ to the injury or death of humans. For the purposes of this paper, we will restrict ourselves to this definition. It can be noted, however, that the approach presented can easily be expanded to cover alternative conceptions of safety, such as those including avoidance of material loss.

The problems faced by safety analysts when attempting to analyse SoS fall into three categories: the difficulty of performing hazard analysis, the restricted means by which safety features can be introduced, and the problem of ‘System Accidents’. In their discussion of functional hazard analysis, Wilkinson and Kelly [4] note that these problems are present in conventional systems. The characteristics of SoS, however, exacerbate them.

2.1 Hazard Analysis

In a conventional system, such as a single vehicle or a chemical plant, the system boundary is well-defined and the components within that boundary can be enumerated. Once hazard analysis has been performed to identify events that may cause injury or death, safety measures can be introduced and the risk of accidents computed from the probabilities of various failures. Conventional techniques such as fault tree analysis are effective in this task.

In an SoS, the necessary hazard analysis is itself very difficult. When hazard analysis postulates some failure of a component, the effect of that failure must
be propagated through the system to reveal whether or not the failure results in a hazard. The system boundary is not well defined, and the set of entities within that boundary can vary over time, either as part of normal operation (a new aircraft enters a controlled airspace region) or as part of evolutionary development (a military unit receives a new air-defence system). Conventional tactics to minimise interactions may be ineffective, because the system consists of component entities that are individually mobile. In some cases, particularly military systems, the entities may be designed (for performance purposes) to form ad-hoc groupings amongst themselves. Conventional techniques may be inadequate for determining whether or not some failure in some entity is hazardous in the context of the SoS as a whole.

It follows from this that a hazard analysis approach is needed which can reveal hazards caused by failure propagation through complex systems and that can consider the effect of multiple simultaneous failures.

2.2 Ensuring Safety

A purely functional design with no safety features is unlikely to be adequately safe. Therefore, design changes need to be made in order to reduce safety risk to acceptable levels. In a conventional monolithic system, there are many features that can be introduced to prevent or mitigate hazards; examples include blast doors, interlocks, and pressure release valves.

The SoS that are considered here contain many mobile agents with a high degree of autonomy. Such ‘hard’ safety features are therefore not available. Consider, for example, air traffic control. If a controller wants to prevent a given aircraft from entering an airspace region (say, one reserved for an airshow) then he or she can instruct the aircraft to fly around it. The controller cannot, however, physically prevent the aircraft from flying into the region. (In a military scenario there are more drastic measures for dealing with aberrant agents, particularly if they are unmanned.)

Therefore, achieving safety in an SoS will rely to a large extent on responsible behaviour from the individual agents. In order to achieve this, agents need to know what behaviour is acceptable in any given circumstance. It follows from this that system designers and operators need to know how the agents in the system can safely interact.

2.3 System Accidents

Perrow, in [5], discusses what he calls ‘normal accidents’ in the context of complex systems. His ‘Normal Accident Theory’ holds that any complex, tightly-coupled system has the potential for catastrophic failure stemming from simultaneous minor failures. Similarly, Leveson, in [6] notes that many accidents have multiple necessary causes; in such cases it follows an investigation of any one cause prior to the accident (i.e. without the benefit of hindsight) would not have shown the accident to be plausible.

An SoS can certainly be described as a ‘complex, tightly-coupled system’, and as such is likely to experience such accidents. It can also be noted that a
‘normal accident’ could result from the combination of apparently safe, normal behaviours which are safe in isolation but hazardous in combination. Imagine, for example, a UAV that aggressively uses airspace and bandwidth under some circumstances. This may be safe when the UAV is operating on its own, but not when it is part of larger SoS.

It follows from this that an SoS safety analysis approach will need to be able to capture the effects of interactions between multiple simultaneous failures and normal agent behaviour.

3 What is Safety Policy?

3.1 Background on Policy

The belief that numerous independently designed and constructed autonomous systems can work together synergistically and without accident is naïve, unless they are operating to a shared set of rules which is informed by a high level view of the system. In existing systems of systems such rules already exist, to a degree, because otherwise such systems would be nothing more than an uncoordinated collection of parts. Burns, in [7]: “The proper functioning of the network as a whole is a result of the coordinated configuration of multiple network elements whose interaction gives rise to the desired behaviours.”

The problems that we face, however, are that often these rules or procedures are either not explicitly expressed, not well understood or are inconsistent. Similarly, they typically do not consider the inter-operating systems as a whole SoS, or simply do not address the safety aspects arising from this inter-operation. A term that can be used to encompass such rules and procedures is policy. Whilst some existing work covers security policy, no work yet deals with a policy for the safe operation of a system of systems.

The Oxford English Dictionary [3] defines ‘policy’ as:

“A course of action or principle adopted by a government, party, individual, etc.; any course of action adopted as advantageous or expedient.”

Intuitively, therefore, a policy guides the action of an individual or group according to some criteria. Foreign policy, for example, is a familiar concept from everyday language and sets out ground rules for guiding a nation’s diplomatic interactions with other nations. Similarly, common law attempts to curtail undesirable—and hence illegal—behaviour and promote desirable behaviour amongst the populace.

Much of government policy, however, confuses policy with ‘goal-setting’. Although some definitions of policy mention goals, they are in the context of policy goals, or high-level actions, such as “the system is to operate safely at all times” or “no University applicant should be discriminated against based on his/her ability to pay tuition fees”, as distinct from targets, e.g. “to ensure 50% of school-leavers continue to higher education”. Policy can therefore be thought of as being orthogonal, but complementary, to plans and goals.

Policy is defined in the literature in various ways, but the most generally applicable system-oriented definition is given in [8]:
“A policy is a rule that defines a choice in behaviour of a system.”

This definition is distinct from that used in, for example, reinforcement learning, where a prescriptive policy maps from perceived internal state to a set of actions. Indeed, it can be seen that policy is persistent [9]; policy is not a single action which is immediately taken, because a policy should remain relatively stable over a period of time. Any policy containing one-off actions is brittle, in that it cannot be reused in a different context and quickly becomes out-of-date and invalid.

Most organisations issue policy statements, intended to guide their members in particular circumstances [9]. Some provide positive guidance, while others set out constraints on behaviour. To take a simple example as an illustration, consider a mother who asks her child to go to the corner shop to buy a pint of milk. She may lay down two rules with which the child must comply on this trip:

1. The child must not talk to strangers.
2. The child must use the pedestrian crossing when crossing the road.

The first of these rules defines what the child is allowed to do, specifically it prescribes conversation with people with whom the child is not previously acquainted. The second statement expresses the obligation that the child should take a safe route across the road, namely by using the pedestrian crossing. Together these rules form a policy that guides the behaviour of the child on his journey to the corner shop. The rules are invariant to the child’s ‘mission’; they still hold whether the child is going to buy a loaf of bread or a dozen eggs, or not going to the corner shop at all.

3.2 Systems of Systems and Safety Policy

According to Bodeau [10], the goal of SoS engineering is “to ensure the system of systems can function as a single integrated system to support its mission (or set of missions).” Among the principle concerns of SoS engineering that Bodeau identifies are interoperability, end-to-end performance, maintainability, reliability and security. Unfortunately, he neglects to mention safety.

Wies, in [11], describes policy as defining the desired behaviour of a system, in that it is a restriction on the possible behaviour. Leveson extends this sentiment to say that the limits of what is possible with today’s (software-based) systems are very different to the limits of what can be accomplished safely [6]. In terms of collaborative groups of systems, SoS, whose behaviour has been observed to be non-deterministic, a policy is a mechanism to create order or (relative) simplicity in the face of complexity. Sage and Cuppan [12] talk of “abandoning the myth of total control”, while Clough [13] describes it as creating a system that is “deterministic at the levels that count”, i.e. at the ‘black-box’ level, and Edwards [14] observes the need to “selectively rein in the destructive unpredictability present in collaborative systems”.

In discussing policy many different terms are employed, such as rule, procedure, convention, law, and code of conduct. The presence of so many terms
would seem to suggest a lack of clarity about what policy is, but these terms can be viewed as policy at different levels of abstraction. Often policy specifications cause confusion by combining statements at high and low levels of abstraction [11].

Policy statements or goals can be organised into a hierarchy, with the most abstract at the top. There is a need to refine from these abstract policies down to implementable, atomic procedures. Existing goal-oriented techniques and notations, such as GSN [15], KAOS [16] and TROPOS [17], provide a basis for the decomposition of high-level goals. Specifically, the Goal Structuring Notation (described by Kelly in [15]) allows the explicit capture of contextual assumptions, for example assumptions made about other agents’ behaviour, and of strategies followed to perform the decomposition.

At the lowest level of abstraction policies can be expressed in terms of the permissions, obligations and prohibitions of individual and groups of agents. In this paper, an approach is suggested for decomposing and implementing policy goals motivated by safety concerns in a simulation of an SoS. The effect of this policy is to moderate the behaviour of the agents such that no accidents occur in the simulated SoS.

4 Requirements on the Simulation Engine and Models

Multi-agent Simulation has previously been used in a safety context, for example to evaluate the safety of proposed changes to the US National Airspace System [18] and to study the relationship between road intersection layout and automobile accidents [19]. As noted by Ferber in [20], such simulations “make it possible to model complex situations whose overall structures emerge from interactions between individuals”.

However, not all multi-agent simulations are suitable for safety analysis. In order to perform safety analysis using simulation, there are two key requirements that must be satisfied by the simulation environment and the models that it contains. Firstly, the simulation must be able to generate the types of hazards and accidents that are of concern, without the emergent system behaviour being described in advance. Secondly, it must be possible to detect these situations when they occur.

For example, consider a system to be analysed that involves flocking Unmanned Air Vehicles (UAVs). Given a description of how the entities behave, in terms of flight control, attempting to achieve mission goals, and collision avoidance, it must be possible to run a simulation of a typical mission scenario and see what flight paths the entity behaviour would generate. It must also be possible to detect whether these flight paths would lead to collisions, or hazardous loss of separation.

From the general requirements above, and by looking at the nature of the accidents we are concerned with, a number of more detailed requirements can be derived. These requirements are discussed in the following sections.
4.1 Sharing of Physical Space

Safety-critical accidents must, by their nature, occur in physical space. At the point of an accident, it is through physical interaction that humans are injured or killed. It follows that a safety-related simulation must have a clearly-defined model of space and time interactions. Models that abstract away such details (e.g. by maintaining only relative time ordering, or by dividing geography into large, arbitrarily shaped regions) will not be able to capture the necessary interactions. It can be noted that although physical space is needed to actually effect an accident, many accidents have causes which can be traced back to events and interactions at the control system or communication levels.

4.2 Autonomous Entity Behaviour

The SoS that are of concern to us involve entities with a large degree of autonomy. Many SoS that are being developed now feature unmanned vehicles, and their autonomous behaviour is an important issue for safety analysis. Negative emergent behaviour, resulting from the interaction of many such vehicles, is a particular concern. It is therefore important to model autonomous behaviour. Autonomous agents are also needed in order to simulate deviation from expected scenario courses; entities must be able to make plausible decisions and actions once the situation has departed from the expected course of events. The simulation cannot, therefore, rely on a single centralised plan of action. The entity models must be capable of some degree of planning and decision-making so as to achieve their goals in the face of unexpected obstacles.

4.3 Local and Shared Entity World Views

A common cause of accidents in many systems is a discrepancy between the mental model of one agent (be it a UAV or a chemical plant worker) and the actual state of the world. Each agent has a local world model based on the information that they have perceived directly, that they have received in communication from others, and that they have inferred from the information from the other two sources. For example, an airline pilot can observe other aircraft in their immediate area, can receive notification from an air traffic controller of upcoming weather obstacles, and can infer from the ATC’s instructions that the course they have been placed on is free from either. Increasingly, automated systems are used to share data between agents in a system. Examples include air traffic control centres exchanging data on aircraft that are moving from one region to another, and a group of fighter aircraft having access to the combined vision cones of all their radars (this is sometimes referred to as ‘data fusion’). This exchange provides many benefits (potentially including safety benefits, as agent knowledge is increased), but also raises new kinds of hazards. For example, if an agent misidentifies a friendly aircraft as hostile, a data fusion system may propagate that ‘knowledge’ to many other agents, some of whom may be in position to threaten the friendly aircraft.
4.4 Communication Between Entities

As mentioned above, entities can supplement their world model through communication with other agents. Communication also incorporates command and control relationships, which affect the behaviour of subordinate agents. Errors in communication may, consequently, cause accidents either by modifying an agent’s world model or by instructing the agent to perform an unsafe action.

4.5 Proxy Measures of Safety

Although a simulation model may generate explicit accidents, a safety analyst cannot rely on this. As in the real world, accidents in a well-modelled simulated SoS will be rare; they will be avoided due to subtleties of time and distance. For example, a collision between a UAV and a manned aircraft may be repeatedly avoided in a series of different runs, with the two aircraft coming close to collision but never actually colliding. For the case of policy, the number and severity of accidents is therefore too crude a measure for the safety of a given policy. There is therefore a need for surrogate measures (e.g. counting near misses rather than just collisions), offering greater resolution than a simple casualty count.

4.6 Introducing Expected Variation

In a model that is solely concerned with performance, for example, it may be sufficient to capture only average performance over time. For a safety model, this is not sufficient; specific, high-cost events must be captured. Therefore, simulations must not only be performed with idealised models of expected entity behaviour; they must also cover all anticipated failure modes. For example, it must be possible to specify that a UAV included in a simulated system has no functioning IFF (Identify Friend-or-Foe) capability, or that one of its engines is operating at reduced maximum thrust.

Going beyond simple failures, it is also desirable to be able to implement different behaviours. Each entity has a set of default behaviours, and the developer of an entity model may provide a set of optional or alternative behaviours. By swapping behaviours in and out from this larger set, variations in overall entity behaviour can be introduced that may affect the results of the simulation run. An example would be swapping a cautious target identification behaviour for a more aggressive one that made fewer checks before deciding that an entity was hostile.

5 Implementing a Multi-Agent Simulation of an SoS

5.1 Describing Policy in a Machine-Interpretable Form

Policy has to be implemented by individual agents — even if there is a central ‘master controller’ for the whole system, in the systems we are dealing with it will not be able to enforce policy by fiat. Therefore policy has to be decomposed into rules that are expressed in terms of individual agent behaviour. It follows that for any given entity type, policy must be expressed such that it involves:
- Responding to states or events that the agent is capable of observing
- Making decisions that are within the scope of the agent’s intelligence and world model (this is particularly important for non-human agents)
- Taking actions that the agent is capable of performing

To this end, policy is decomposed in the context of an SoS model. This model embodies the contextual assumptions of other agents’ behaviour, knowledge and capabilities. Policy decomposition proceeds with increasing specificity, working top-down from a high-level goal to policy statements on individual agents or sets of agents. Goal Structure Notation (see section 3.2) allows the explicit capture of the strategies by which the decomposition is achieved and the context necessary for such a decomposition.

Figure 1 illustrates an excerpt from a possible policy hierarchy for the UK civil aerospace Rules of the Air [21]. The policy decomposition starts from an obvious high-level goal, ‘No collisions shall occur in the civil aviation SoS’, which is at the top of the diagram. This goal is then decomposed hierarchically, leading eventually to a number of low-level goals (leaf nodes on the diagram; due to space limitations, only two of these are shown). These lowest-level goals correspond to policy rules that can be implemented directly by agents; in the diagram, the goal ‘Below1000’ has been annotated by a machine-interpretable version of the corresponding policy rule.

The low-level policy statements are expressed as one of three types:

**Permit** Describes the actions that an agent is permitted to perform or conditions that it is permitted to satisfy.

**Forbid** Describes the actions that an agent is forbidden to perform or conditions that it is forbidden to satisfy.

**Oblige** Describes the actions that an agent is obliged to perform.

In contrast to policy definition languages such as Ponder [22], we do not attempt to define those actions which an agent is forbidden to perform as well as those which it is obliged not to perform. As mentioned in section 2.2 it is not always possible to prevent agents from performing actions contrary to policy. Unlike security policies, which often assume the presence of an access control monitor, safety policy cannot assume that aberrant agent behaviour can be blocked by an external controller. For example, in air traffic control, there is no external way to stop a wayward aircraft from straying into a forbidden region of airspace. In a sense, the system operator must rely on agents to police themselves.

There must also be a design decision about the overall permissiveness of the SoS. A policy model can either be open or closed: the former allowing all actions which are not expressly forbidden, while the latter forbids all those actions that are not explicitly permitted. The presence of both permit and forbid in this policy model would therefore appear redundant. This is not so, however, given that exceptions to rules can be expressed in the opposite modality. For instance, in an open policy model, a policy rule may forbid the low flying of an aircraft; exceptions to this rule (e.g. for take-off and landing) can be expressed...
as permissions. The more specific permissions must then take precedence over the more general blanket policy to forbid low flying.

5.2 Implementation in an Agent Architecture

The implementation of policy at the agent level is tied closely to the details of the agent architecture used. In the current work, an architecture is proposed based on the ‘C4’ architecture developed by the Synthetic Characters group at the MIT Media Lab. This is a blackboard architecture, in that it contains a series of subsystems that communicate only through a structured shared memory space. C4 is described by Isla et al in [23]. The blackboard architecture is valuable in that it allows discrete behaviours to be loosely coupled, and hence allows variant behaviours to be easily swapped in and out as described in section 4.6.

Our proposed architecture is depicted in figure 2. The core of the system is the blackboard, which is divided into several sub-boards. Of particular note is the outgoing action board, which determines what the agent actually does after each decision cycle. Each agent has several behaviours, which act by making changes to the blackboard (including the action space). The arbitration strategy is simple — on each time ‘tick’, all behaviours are processed in turn (from top to bottom in the diagram).

The arbiter also has a role in enforcing adherence to safety policy. It can be seen that one of the first behaviours to be processed is the Policy Processor, which compares the current policy to the current state and ‘fires’ all the policy rules that apply. This generates a set of permitted and forbidden actions, which is written to the policy sub-board. This sub-board is hereafter read-only — the other behaviours can observe it, in order that they might propose only permitted actions, but they cannot change it. The policy sub-board is regenerated on each tick, as changes in the environment may change which policy rules now apply.

Policy rule firings generate tuples of the form (operator, action, list of parameters). For example:

- (FORBID, change-speed, < 180 knots)
- (FORBID, enter-region, 2000m radius of [15000,5100])
- (PERMIT, attack-target, entity#127)

The rule-firings and the behaviours use the same ontology. As noted above, behaviours can see which PERMIT and FORBID policy rules are active at the current time, and modify their behaviour accordingly. As a supplement to this, or an alternative, the arbiter may check proposed actions (against the policy board) and reject those that are against the rules. This could seem redundant, since the behaviours are part of the agent, and hence as much a trusted source as the arbiter itself. It is easier, however, to build a reliable policy enforcer than it is to build behaviours that always conform to policy. Likewise, it is easier to build a behaviour that chooses its action taking into account what policy currently permits, rather than build one that tries whatever action seems best, then tries to respond when the action is forbidden.
No collisions
No collisions shall occur in the civil aviation SoS

Entities
Entities that should be avoided are other aircraft, the ground and fixed objects

FixedObjects
Fixed objects are attached to the ground and have some height

Collision strategy
Decomposition over all entities with which an aircraft can collide

Aircraft collision
An aircraft shall not collide with other aircraft

Ground collision
An aircraft shall not collide with the ground or fixed objects

VisualRange
The pilot of an aircraft shall maintain a minimum visual range from the cockpit

VisAllAirspace
Decomposition over nature of control of airspace

VisRangeOutContrAirspace
The pilot of an aircraft shall maintain a minimum visual range from the cockpit outside controlled airspace (26)

VisRangeInContrAirspace
The pilot of an aircraft shall maintain a minimum visual range from the cockpit within controlled airspace (25)

VisRangeClassAirspace
Decomposition over all classes of airspace

VisRangeInClassBAirspace
The pilot of an aircraft shall maintain a minimum visual range from the cockpit within class B airspace (25)(1)

VisRangeInClassCDEAirspace
The pilot of an aircraft shall maintain a minimum visual range from the cockpit within class C, D and E airspace (25)(2)

ClassA
Flights in class A airspace are assumed to require no minimum visibility

Key to Symbols
Goal
Strategy
Context
Solved by
In context of
UndevelopedGoal

fig. 1. Example Policy Decomposition for Rules of the Air
An alternative to the policy enforcement role of the arbiter is that of a monitor, which notes policy violations but does not prevent them. This is particularly valuable during development of an agent.

An advantage of the blackboard model is that the behaviours are loosely coupled to each other; they interact only through the blackboard. This means that behaviours can be added, removed or changed at any time, without changing the other behaviours. This relates to the requirements identified in section 4.6.

5.3 Evaluating the Safety Achieved by the Policy

In order to evaluate the level of safety that has been achieved, the SoS agents must be configured to use the policy, then simulation runs must be performed for a variety of representative scenarios. Further variation can be introduced through failures and variant behaviours applied to agents. The level of safety achieved by the system operating with the policy can be evaluated by measuring the number of accidents and incidents, and the worst near-incidents, that occurred across all runs with that policy.

Once the SoS model has been configured and the set of runs decided on, measures must be put in place to measure the level of safety achieved by the system. From the definition of safety presented in section 2, it can be seen that two types of event need to be counted.

The first type is accidents, which corresponds to “exemption from hurt or injury” in the definition. Examples of such accidents include collisions between vehicles and military units firing on friendly forces. The set of possibilities is quite small, and they can be easily detected.

From “freedom from danger” we can derive another class of event, the incident or ‘near miss’. Examples include activations of an aircraft’s collision-
avoidance system, separation between two aircraft falling below some safe level, and queries to a superior of the form “Is X hostile?” when X is in fact friendly. Unlike actual accidents, a great many types of such incidents can be described. It can be noted that many incidents correspond to hazards; when an incident occurs, it may be the case that an accident could happen without any other deviation from normal behaviour.

The great value of counting incidents as opposed to accidents is that accidents are extremely rare — in the real world, accidents are often avoided by (seemingly) sheer chance. Measures that track incidents can be given variable sensitivity, so that they can be adapted to the level of risk that is actually exhibited in the system. For example, it is desirable to calculate actual collisions accurately, so that their effects on the unfolding scenario can be modelled realistically. This is especially true if a multi-criteria analysis is being performed, for example with performance as well as safety being analysed. By comparison, an incident measure based on aircraft proximity can be as sensitive (or insensitive) as is required since triggering it only affects the statistics gathered, not the events that follow it.

For both accidents and incidents, it is possible to weight events by a measure of their severity. Consider one policy that generated a number of minor accidents against another that caused a single accident with massive loss of life. A simple approach is to count the human casualties (or potential casualties, in the case of an incident) that could result from an event. The safest policy is then the one that caused the smallest loss of simulated lives over all the scenarios that were considered. Weighing accidents against incidents is more difficult, however; there is the question here of model fidelity, and the consequent fear that an incident in the simulation might have been an accident in the real system.

The means of detecting accidents during a simulation run are well understood as they are an essential part of many non-safety simulations. Providing a large range of incident detectors is less straightforward, and some of these will raise performance challenges. This is, however, beyond the scope of this paper.

If two policies cannot be compared because their accident and incident counts are zero or very low, a third technique is possible. For a variety of measures, perhaps the same measures as those used for incidents, the worst magnitude achieved could be tracked. An obvious example is violation of aircraft separation; rather than just counting the number of occasions on which this occurred, the minimum separation achieved can be recorded. The minimum for the policy is then the minimum over all runs. An example of this can be seen in Benson [24].

6 Dependability Conflicts in Systems of Systems

Safety is an important system attribute, but it is not the only consideration when developing an SoS. There are other important attributes such as availability, performance and security. The term dependability is commonly used to encompass all such system attributes [25]. Attempting to address all these different attributes can result in competing objectives; consequently there are conflicts
that need to be resolved and trade-offs that need to be made in order to achieve the optimum characteristics for a system.

In SoS, conflicting objectives (and hence trade-offs) are inevitable; probably the most obvious are conflicts between performance and safety. An example is the reduction of minimum aircraft vertical separation (RVSM), within controlled airspace. In RVSM airspace, aircraft fly closer to each other, greatly increasing the number of aircraft that can be served by an ATC centre within a certain period of time. This has obvious performance benefits (reduction of delays, more flights during peak hours), but it raises some serious safety concerns. Special safeguards (changes to either sub-system design or operational policies) are therefore required.

If an SoS is developed with safety as the highest priority, it will be possible to devise policies that constrain the interactions of system agents to the safest that are possible. However, such an approach might unacceptably decrease the overall performance of the system. For example, there is no point in introducing an extremely safe air traffic policy if doing so reduces the throughput of the system to uneconomic levels. In order that safety is not achieved at the unacceptable detriment of other attributes, it is important to model the effect on all attributes of safety-related changes.

Performance acceptability criteria differ depending on the particular system mission. Therefore, the required performance level and its criticality (based on which we determine our willingness to compromise safety in favour of performance) are defined with consideration of the system’s context and operational objectives. Simulation provides a way to evaluate the different dependability attributes of the system in different contexts, by running a set of representative operational scenarios. This provides a basis for achieving a satisfactory trade-off between the different attributes.

7 Issues and Challenges

7.1 Model Fidelity and Statistical Significance

No novel and untried systems of systems will enter operation with only simulated evidence of its safety. Simulation, however, gives a guide to the behaviour of the system which informs, and is supplemented by, further analysis. It is particularly valuable in that it can reveal the emergent behaviour of a complex system in a variety of contexts; it is difficult if not impossible to acquire knowledge of this by other means.

Even when the fidelity of a given simulation is considered inadequate to assess the safety of a system, it can provide confidence that a given policy is viable, and help judge relative superiority to other candidates. (For an example of this, see Benson in [24]). Perhaps most importantly, the simulation analysis can reveal flaws in a policy that would not have been apparent in manual analysis.

The problem of model fidelity, and of the validity of any results that are gained through simulation, is a serious one and affects all applications of simulation analysis, not just safety. This is a longstanding controversy in the field of
robotics; discussion can be found in Brooks [26] and Jakobi [27]. In the current context, one key requirement for usefulness is that the simulation be able to exhibit emergent behaviour.

7.2 Volume of Processing

As noted above in section 5.3, evaluating the safety of the system requires a large number of scenarios to be simulated. For each of these simulations, a large range of failures and variant behaviours need to be considered. Combinations of failures and behaviours are also important.

It follows that the possible set of simulation runs is extremely large. A na"ıve approach would be to run all possible combinations of scenario, failures and behaviours. However, as discussed by Hoeber in [28], such exhaustive exploration is intractable even for simple simulations and modest numbers of inputs.

There is therefore a need for more targeted exploration of the state space. In [29], Dewar et al discuss some experimental designs that can be used for reducing the number of combinations that need to be run. Many such designs, however, deal poorly with systems in which the interesting phenomena result from combinations of inputs.

One other approach would be to concentrate on and prioritise those combinations of failures that were statistically most likely. A useful selection criteria can be based on the potential for certain types of SoS failures to occur together, as discussed by the authors in [30].

8 Summary

In this paper, we have presented the case for using safety policy to ensure the safe behaviour of complex systems of systems, and suggested multi-agent simulation as a means of evaluating the effectiveness of such policies. It is clear that analysing SoS is difficult, particularly when they are highly decentralised. Simulation offers an approach to dealing with some of these difficulties.

An approach to policy evaluation has been proposed, whereby an SoS is exercised through a variety of simulations for each candidate policy, with a range of failures and behaviour modifications being introduced. The level of safety provided by each policy can be assessed by measuring the values of various safety-related parameters. This concept can be extended further, using simulation to consider the trade-off between safety and performance.

A number of challenges remain, such as limitations in the fidelity of models and the number of runs needed to get statistically valid results. The authors are currently working on tools and examples to demonstrate the concepts described in this paper.

References

A compositional proof system for agent behaviour

Dominique Fournier, Bruno Mermet, Gaëlle Simon

LIH, Université du Havre

Abstract. Multi-agent systems (MAS) are distributed systems that are very hard to validate. In this paper, we present a new proof system dedicated to agent behaviour validation. This proof system is goal oriented, indeed it allows to compositionally prove the agents whose behaviour is described in terms of goals and goal decomposition operators.

1 Introduction
Our research deals with methods and models of multiagent systems (MAS) in order to help designers to manage the complexity of MAS. We do not aim at developing yet another agent model: there are already numerous ones. We aim at helping to develop MAS whose behaviour can be proved and therefore can ensure operational safety of MAS. As a consequence, our approach consists in 4 steps: an agentification method that helps to determine agents; an agent design model to help to design an agent behaviour; a proof system to prove the agent model; and an implementation model that can be automatically generated. In this article, we focus on the third step: the proof system. The other steps are presented in other papers [7,4]. In the following section, we briefly present a state of the art about proof systems. In the third section, our agent design model, relying on a goal decomposition tree (GDT) is summarised. The core of this article is in section four, explaining our compositional proof system. Finally, we conclude on our work in the last section.

2 State of the art
There is a great number of formal methods like the \( \pi - calculus \), Unity, Z, VDM, B or TLA+. However, they are not suitable for specifying MAS because of the following limitations: lack of expressivity, lack of structuration, impossibility to prove liveness properties, language hard to understand. So, developing a new formal model dedicated to MAS seems necessary. Many models have been proposed and are presented in the next section.

2.1 Formal MAS models
The behaviour of a MAS is so complex that standard methods do not provide enough guaranty to enlarge the use of MAS in industry. Indeed, because of their complexity, it is hard to know whether a MAS is bug-free or not. Thus, many works are being performed to develop formal methods dedicated to MAS [3]. For instance, Wooldridge et al. developed the Gaia method [8] for MAS development. For the first time, a MAS is specified twice: in terms of behaviour and in terms of
invariant properties. Thus, the behaviour is described by the liveness properties: the bases to prove MAS are introduced in Gaia.

This kind of methods allow to formally specify the system, helping the designer to consider every potential situation and thus reducing the number of potential bugs; yet, they do not allow to perform proofs.

2.2 MAS proof systems
There are essentially two ways to prove the correctness of a specification: model checking and theorem proving.

Model checking is a verification method consisting in testing all the situations which may be encountered by the system. Recently, works like [5], have used Unbounded model checking, a very promising extension of model checking. With such methods, model checking is performed on formula, allowing to group together similar situations, verifying many cases in one step. But, if the complexity is reduced, it is still there and very complex systems are always unverifiable.

A recent and very interesting work is the one performed by Bracciali et al. [1] about PROSOCS agents. PROSOCS agents are agents whose behaviour is described by goal decomposition rules à la Prolog parameterised by time variables. However, the system is limited to propositional logic formula. Moreover, the system is quite complex because of the number of models that are used to define PROSOCS agents. So, the system can not be easily used to program agents.

3 Decomposing goals into subgoals
In order to be able to implement and validate an agent, its behaviour is specified with a Goal Decomposition Tree (GDT). In such a structure, the top goal of an agent is decomposed into subgoals that contribute to achieve the top goal. Several works (like [2]) have already pointed out the advantage to have such a declarative description of goals. Nodes of a GDT correspond to goals the agent has to solve. Several kinds of goals have been defined using three different criteria. This typology of goals is useful for the verification of the consistency of the tree. Inside a GDT, a goal is decomposed into subgoals using decomposition operators. Each operator is associated to a local proof schema that is used to prove the agent behaviour.

3.1 Goals and typology of goals
In the context of a Goal Decomposition Tree, a goal is defined by a name and a satisfaction condition. Satisfaction conditions (SC) are expressed using temporal logic. A goal is considered to be solved if its satisfaction condition is logically true. Three criteria have been defined in order to distinguish more precisely different ways to manage goals decomposition. The first criterion to distinguish goals corresponds to the situation of the goal in the tree: elementary and intermediate goals. The second criterion used to define goals is related to the goals satisfiability. Using this criterion, two kinds of goals are again distinguished: Necessarily Satisfiable goals (NS) and Not Necessarily Satisfiable goals (NNS). Necessarily Satisfiable goals (NS) ensure that, once all what must be done to solve the goal has been executed, the satisfaction condition of the goal is always
true. The last criterion considers the lazyness of the goals. When a goal is a lazy goal (L), its satisfaction condition is evaluated before considering its actions or its decomposition. If the satisfaction condition is true, the goal is considered to be solved, and the subtree is not used. If the satisfaction condition is false, the set of actions or the decomposition are executed. On the contrary, for a not lazy goal (NL), the associated set of actions or decomposition is always executed.

### 3.2 Decomposition operators

In this section, available decomposition operators are described. Before describing each operator, let precise what means a goal decomposition. Let $A$ be the goal to be decomposed with the decomposition operator $Op$ to obtain the subgoals $B$ and $C$. The semantics of this decomposition is that the satisfaction of $Op(B, C)$ implies the satisfaction of $A$.

**And and Or operator:** these operators correspond to the well-known logical operators adapted to a temporal context. The decomposition semantics of *And* states that if a goal $A$ can be decomposed in $And(B, C)$, then $A$ can be satisfied (its satisfaction condition is true) if $B$ and $C$ can be satisfied. This two subgoals can be solved in any order. If one of these two goals can not be solved, the parent goal $A$ is considered to be not solved. The two subgoals of this decomposition can be either necessarily satisfiable either not necessarily satisfiable. However, if at least one of the subgoals is not necessarily satisfiable, the parent goal is automatically not necessarily satisfiable. It is necessarily satisfiable otherwise.

**SeqAnd and SeqOr operators:** these operators are sequential. Indeed, the only difference with the non-sequential ones is that the two subgoals must be solved in the order specified by the operator.

**SyncSeqAnd and SyncSeqOr operators:** these operators are synchronized version of the sequential operators. Unlike the previous, these operators ensure that the two subgoals (if they are both solved) are solved without any interruption by an other agent.

**Iter operator:** this operator is an unary one. The main difference between this operator and the others is that its behaviour depends on the satisfaction condition of the parent goal. Indeed when a goal $A$ is decomposed in $Iter(B)$, the decomposition is parameterised by the satisfaction condition of $A$. As a consequence, one must read "$A$ is decomposed in $Iter_{CSA}(B)$". The parent goal will be satisfied after several satisfaction steps of the subgoal. This operator is very important because it allows to take into account a progress notion inside a goal solving process. The parent goal is always necessarily satisfiable. Indeed, the behaviour of the operator implies that the solving process of the subgoal stops when the satisfaction condition of the parent goal is true.

### 4 Proof schema and Context propagation

The main goal of the proof is to prove the GDT built for an agent (i.e. to prove that the behaviour specified by the tree always satisfies the main goal of the agent). For each decomposition operator we have defined a proof schema and a context propagation schema. In the following, we illustrate these schemas on the *SeqAnd* operator.
Notations

During the proof process, we have to consider an invariant of the system $I_S$ produced by the method SPACE as it has been presented in [6]. This invariant must be preserved by each agent. Moreover, each agent has also its own invariant properties $I_A$ (defining for instance the type of its own variables). So, the property $I = I_S \land I_A$ must be an invariant property of an agent $A$. As a consequence, $I$ can be used as an hypothesis, and we also have to prove that $I$ remains true after the goal resolution. Let us precise that for each goal $G$ we also manage a local context $C_G$ as an hypothesis.

Moreover, to take into account temporal constraints we use substitutions $T$, a transformation function, in order to unify the semantics of the different variables used. For instance, in the formula $T(SC_A)$, unprimed variables correspond to unprimed variables of the formula $T(SC_B)$ whereas primed variables of $T(SC_A)$ correspond to primed variables of $T(SC_C)$. Moreover, primed variables of $T(SC_B)$ correspond to unprimed variables of $T(SC_C)$, so to make this property explicit, all these variables are replaced by new ones as following: each variable $x$ of $T(SC_C)$ (or $x'$ of $T(SC_B)$) is replaced by a new variable $x_{tmp}$.

Last but not least, $SC_B$, the projection of $SC$ on the internal variables of the agent is used because these variables are the only ones that can not be changed between the resolution of $B$ and the resolution of $C$.

Proof schema

To prove such a decomposition, $B \ SeqAnd C \Rightarrow A$ has to be proved.

If $A$ is lazy, we can consider that $SC_A$ is not verified (by definition of lazy goals), so we introduce $\neg SC_A$ in the proof schema and the context propagation. Moreover, the final state of the goal $B$ corresponds to the initial state of the goal $C$. Thus, we have to prove:

$I \land C_A \land \neg SC_A$

$\Rightarrow$

$T(I) \land ([v' := v_{tmp}] T(SC_B) \land [v := v_{tmp}] T(SC_C) \Rightarrow T(SC_A))$

If $A$ is not lazy with a similar reasoning, we obtain the following proof schema:

$I \land C_A \Rightarrow T(I) \land ([v' := v_{tmp}] T(SC_B) \land [v := v_{tmp}] T(SC_C) \Rightarrow T(SC_A))$

Context propagation

<table>
<thead>
<tr>
<th>$A$ is lazy</th>
<th>$A$ is not lazy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_B := C_{A_0} \land \neg SC_A$</td>
<td>$C_B := C_A$</td>
</tr>
<tr>
<td>$C_C := SC_B$</td>
<td></td>
</tr>
</tbody>
</table>

5 Conclusion

In this article, we present a complete proof system for agents behaviours. Of course, using our model, an agent designer must specify each goal by a satisfaction condition. This may seem difficult, but the experience shows that rapidly,
unexperienced designers can write the right satisfaction condition. A major characteristic of the proof system we defined is that it allows a compositional proof, that is to say that proofs can be performed independently at each level of the goal decomposition tree (GDT). As a consequence, proofs are generally not too difficult to perform. Moreover, if a part of the GDT is modified, the proofs associated to the other parts of the tree are still valid.

An important question that remains is: is it possible to prove all the characteristics of a MAS with our technique? At the moment, most safety and liveness properties of the agents can be proved. There is a lack of proof capabilities about properties of the whole system. In fact, the invariant partially solves this problem, allowing to prove invariant properties of the system, but it is neither structured nor easy to produce, and it is not possible to prove liveness properties about the global MAS. Our future work will consist in solving these difficulties.

Finally, the following question is also interesting: isn’t-it too hard to prove a MAS to be really performed? Proving a program, moreover a distributed program like MAS, is a difficult process, because a distributed system is a very complex one. Moreover, a proof process is expensive. However, it may be less expensive than an error, and the model we propose structures the agent in order to make the proof easier.

References

Multiagent modeling and simulation of agents’ competition for network resources availability

Igor Kotenko and Alexander Ulanov

St.-Petersburg Institute for Informatics and Automation, Russia
39, 14th Liniya, St. Petersburg, 199178, Russia
ivkote@iias.spb.su; ulanov@iias.spb.su

Abstract. The paper considers an approach to modeling and simulation of competition in Internet between the antagonistic teams of software agents for network resources availability. We differentiate two teams: (1) the attack team realizing Distributed Denial of Service (DDoS) attacks and (2) the defense team protecting against these attacks. Teamwork based approach for modeling and simulation of agents’ competition is considered. The structure and operation of agents’ teams, their ontologies, main classes and roles of agents are defined. We describe in detail the case-study which demonstrates main ideas of the approach suggested.

1 Introduction

Permanently magnified variety and complexity of cyber-attacks and gravity of their consequences highlights urgent necessity for strong protection mechanisms of computer systems. Especially it is fair in connection with integration of computer systems on the basis of the Internet, not having state boundaries, centralized control and uniform security policy.

Unfortunately, the existing theoretical base for network security in large-scale systems does not correspond to the indicated tendencies. We think the majority of problems is caused by immaturity of logical foundations for construction of integrated adaptive security systems [4]. To our opinion, it is stipulated mainly by insufficient attention to fundamental works, which, on the one hand, consider network security as a complex task of organizational and technical competition between security systems and malefactors’ systems [6], and, on the other hand, are based on exploratory modeling and simulation of indicated processes.

The issues of modeling and simulation of network security have been actively researched for more than thirty years. The various formal and informal models of particular protection mechanisms are developed, but practically there are not enough works formalizing complex antagonistic character of network security. Understanding of network security as uniform holistic system is extremely hampered. It depends on great many interactions between different cyber warfare processes and is determined by dynamic character of these processes and different components of computer systems. Especially it is fair in conditions of the Internet evolution to a free decentralized distributed environment in which a huge number of cooperating and
antagonistic software components (agents) interchange among themselves and with people by large information contents and services. Modeling and simulation of these aspects is supposed to put as a basis of our research. This will allow developing an integrated approach to construction of network security systems which can operate in aggressive antagonistic environment.

One of the most harmful classes of attacks aiming at destruction of network resources availability is Denial of Service (DoS). The purpose of DoS is isolation of a victim host, i.e. creation of a situation in which a remote host can not communicate with external world. The basic feature of Distributed Denial of Service (DDoS) attacks is coordinated use of enormous remote hosts-zombies for generation of ill-intentioned traffic [21, 22].

The purpose of our research is twofold. Firstly, we try to develop formal basis for adaptive co-evolving agent-based modeling and simulation of antagonistic agents’ teams. Secondly, we aim to suggest an approach for modeling and simulation of agents’ competition in the Internet for network resources availability. In this paper we investigate our approach on an example of implementing DDoS attacks and protecting against them. The rest of the paper is structured as follows. Section 2 reviews relevant works and outlines suggested common approach for modeling and simulation of antagonistic agents’ team competition in the Internet. Section 3 describes the issues of modeling and simulation of attack agents’ team. Section 4 presents the aspects of modeling and simulation of defense agents’ team. Section 5 outlines architecture and main user interfaces for software prototype developed. Conclusion outlines the main results of the paper and future work directions.

2 Teamwork based Approach and Related Work

The agents’ team realizes teamwork, if the team members fulfill joint operations for reaching the common long-time goal in a dynamic external environment at presence of noise and counteraction of opponents. The teamwork is something greater, than simply coordinated set of personal actions of individual agents. It is accepted to speak, that in teamwork the agents collaborate. The collaboration is a special sort of a coordinated activity of the agents, in which they jointly solve some task or fulfill some activity for reaching a common goal. The main problem at organization of the agents’ teamwork is how it is possible to provide actions of the agents as united team in a situation, when each agent realize own intentions by personal operations executed in parallel or sequentially with operations of other agents [5, 12, 15, 29, 30, 34].

Several models and software/hardware products have been developed for producing agents’ teamwork [5]. One of the approaches, known as the joint intentions theory, is offered in [3]. It states a common framework determining a team behavior and an interaction of the team members. The more formalized approach, known as the shared plans theory, is offered in [10, 11]. In [29, 30] the key ideas of both approaches are generalized and used at creation of software toolkit for development of applications in the field of teamwork of the agents. The general intentions of agents are determined in a hierarchical reactive plan. This plan describes actions of the team as well as the actions of particular agents. The coordinated tasks are carried out due to
installation of constraints on agents’ roles. GRATE* [12] is an implementation of teamwork using the Joint Responsibility model. This model includes concepts of common goals and instructions (recipes). The individual commitments determine how an agent should operate in a context of teamwork. OAA (Open Agent Architecture) [20] uses a blackboard-based framework that allows individual agents to communicate by means of goals posted on blackboards controlled by facilitator agents. CAST (Collaborative Agents for Simulating Teamwork) [34] supports teamwork using a shared mental model. The mental model includes team processes, team structures and the capability of each teammate. In RETSINA-MAS [7], agents have own copy of a common partial plan. Each agent estimates its opportunities to the requirements of the team goal. In “Robocup Soccer” [28, 15], agents have common knowledge operating their cooperative behavior. COGNET/BATON [35] is a system for simulation of teamwork of people with use of intelligent agents. Team-Soar [13] is a model implemented for testing a theory of team decision making.

In our approach it is offered that the agents’ teamwork is organized by the group (team) plan of the agents’ actions. In result, a team has a mechanism of decision-making about who will execute particular operations. As in the joint intention theory, the basic elements, allowing the agents’ team to fulfill a common task, are common (group) intentions, but its structuring is carried out in the same way as the plans are structured in the shared plans theory [10]. The common (group, individual) intention and commitment are associated with each node of a general hierarchical plan. These intention and commitment manage execution of a general plan, providing necessary flexibility. During functioning each agent should possess the group beliefs concerning other team-mates. For achievement of the common beliefs at formation and disbandment of the common intentions the agents should communicate. All agents’ communications are managed by means of common commitments built in the common intentions. For this purpose it is supposed to use the special mechanism for reasoning of agents on communications. Besides it is supposed, that agents communicate only when there can be an inconsistency of their actions. It is important for reaction to unexpected changes of network environment, redistributing roles of the agents which failed or unable to execute the general plan, and also at occurrence of not planned actions. The mechanisms of the agents’ interaction and coordination are based on three groups of procedures [29]: (1) Coordination of the agents’ actions (for implementation of the coordinated initialization and termination of the common scenario actions); (2) Monitoring and restoring the agents’ functionality; (3) Communication selectivity support (for choice of the most “useful” communications).

The specification of the plan hierarchy is carried out for each role. The following elements of the plan should be described: initial conditions, when the plan is offered for fulfillment; conditions for finishing the plan execution (these conditions can be as follows: plan is fulfilled, plan is impracticable or plan is irrelevant); actions fulfilled at the team level as a part of the common plan. For the group plans it is necessary to express joint activity. To cope with the information heterogeneity and distribution of intrusion sources and agents used we apply ontology-based approach and special protocols for specification of shared consistent terminology. The ontology of network security problem and application domain is specified on the basis DAML+OIL.

The suggested technology for creation of the malefactors-agents’ team (that is fair for other subject domains) consists in realization of the following chain of stages [16]:

29
(1) Formation of the subject domain ontology; (2) Determination of the agents’ team structure; (3) Determination of agents’ interaction-and-coordination mechanisms (including roles and scenarios for roles exchanges); (4) Specification of agents’ plans; (5) Assignment of roles and allocation of plans between agents; (6) State-machine based implementation of teamwork.

The agents’ team structure is described in terms of a hierarchy of group and individual roles. Leaves of the hierarchy correspond to the roles of individual agents, but intermediate nodes - to group roles. One agent can execute a set of roles. Agents can exchange roles in progress of plan execution. Agents’ coordination is carried out by message exchange. As the agents’ teams operate in antagonistic environment agents can fail. The lost functionalities are restored by redistributing the roles of failed agents between other agents and (or) cloning new agents.

Agent-based modeling and simulation of network security in the Internet assumes that agents’ competition is represented as a large collection of semi-autonomous interacting agents. The aggregate system behavior emerges from evolving local interactions of agents in a dynamically changing environment specified by computer network model. We assume to select at least two antagonistic agents’ teams effecting on computer network as interconnected set of resources: (1) Attack system is a team of attack agents (automatically generating DDoS attacks); (2) Defense system is a team of security agents (for intrusion protection, data sensing and information fusion, intrusion detection, and incident response).

The main task of defense systems against DDoS is to accurately detect these attacks and quickly respond to them [31]. It is equally important to recognize the legitimate traffic that shares the attack signature and deliver it reliably to the victim [23]. Traditional defense include detection and reaction mechanisms [33]. Different network characteristics are used for detection of malicious actions (for example, the source IP address [27], the traffic volume [8], and the packet content [26]). To detect the abnormal network characteristics, many methods can be applied (for instance, statistical [18], cumulative sum, pattern matching, etc). As a rule, the reaction mechanisms include filtering [25], congestion control [19] and traceback [17].

Let us consider some papers that provide defense from DDoS attacks by cooperative actions of different components. The paper [1] proposes a model for an Active Security System, comprising a number of components that actively cooperate in order to effectively react to a wide range of attacks. COSSACK [26] forms a multicast group of defense nodes which are deployed at source and victim networks. Each defense node can detect the attack and alert the other nodes. This system combines multicast communications, traditional intrusion detection components, network topology, vulnerability information, and blind detection techniques. In [14] an architecture called Secure Overlay Services (SOS) is described. This architecture uses a combination of secure overlay tunneling, routing via consistent hashing, and filtering. A collaborative DDoS defense system proposed in [32] consists of routers which act as gateways. They detect DDoS attacks, identify and drop attack packets. Gateways are installed and communicate only within the source and the victim domains. In [31] the distributed defense system for protecting web applications from DDoS attacks is described. This system is deployed in both victim and attacker source end. In DefCOM (Defensive Cooperative Overlay Mesh) [23], a peer-to-peer network of cooperative defense nodes is used. When an attack occurs, nodes close to the victim...
detect this and alert the other nodes. Core nodes and those in vicinity of attack sources suppress the attack traffic through coordinated rate limiting. DefCOM nodes are classified into three categories: Alert generator nodes detect the attack and deliver an alarm to the rest of the peer network; Rate limiter nodes limit traffic; and Classifier nodes differentiate between legitimate and attack packets. In [2] two perimeter-based defense mechanisms are suggested. These mechanisms rely completely on the edge routers to cooperatively identify the flooding sources and establish rate-limit filters to block the attack traffic.

In our approach attack and defense systems are represented as antagonistic teams of agents. The purpose of attack team consists in defining vulnerabilities and implementation of security threat directed on availability of network resources by executing DDoS attacks. The purpose of defense team is protection of computer network and own components from DDoS attacks. We have a goal to model and simulate different defense mechanisms. Agents of antagonistic teams compete to reach opposite intentions. Agents of the same team cooperate to achieve common intention (to fulfill attack on computer network or to defense the network).

3 Modeling and Simulation of Attack Agents’ Team

The main idea of DDoS attack is “denial of service” of some network resource by joint operations of many components acting on attack side. Thus, the original task of violation of a resource availability is divided into a set of simple subtasks of “denial of service” that are ordered to particular specialized components. On the upper level the joint goal remains the same for all components. On the lower level the sub-goals are formed. Their achievement is needed to solve the joint goal. The components are interacting to coordinate local solutions. This is needed to achieve the required quality of joint “denial of service” solution.

The components of DDoS attack system are software entities having at least the following properties: autonomy; joint goal and the list of actions for this goal achievement; initial knowledge (about itself, interacting entities and environment) given by the developer; soft knowledge or tough algorithms that permit to process input data; communication and interaction mechanisms for joint goal achievement. These properties allow considering each component of the attack system as an intelligent agent and a set of them as an agents’ team.

Analysis of current DDoS methods allows choosing two main types of attack components: (1) coordinators of other components and (2) direct DoS executors.
Hence we select two classes of agents: master and daemon. The master is managing the sub-team of daemons. If for representation of an attack team we do not use several hierarchical levels, the master is only one in the team. Its function is coordination of daemons’ actions. Thus master is playing a coordinator role. Daemons are doing actual attack actions. They can exchange information for joint goals achievement. Daemons play an executor role. So the attack team is a two-level system (figure 1). Masters act on the higher level directly fulfilling the malefactor’s tasks. They make decisions: when to start the attack, what target to attack, what is the attack intensity. Masters coordinate daemons’ actions by sending commands. Daemons act on lower level. After receiving the messages from masters, they start or finish sending the attack packets or change the attack intensity.

On the initial stage, the master and daemons are deployed on the available (compromised) hosts in the Internet. The malefactor sets the joint team goal – to perform DDoS attack against some of network resources. The master generates the parameters of attack and sends them to available daemons. Further daemons act. Their local goal is to execute commands. So they are sending attack packets to the given host. It is believed that the team goal is achieved if the given resource is completely or partially inaccessible (it deepens on the malefactor goal). Daemons periodically send to the master the messages that they are able to work. While receiving these messages the master controls the attack rate. If there is no message from one of daemons then the master makes decision to change the attack parameters, for example, the intensity of attack. Master can implement it by sending a command to change intensity. The malefactor can stop the attack. In this case he/she sends to the master the command “finish the attack”. Then the master sends the corresponding commands to daemons. After receiving these commands they stop the attack.

The developed ontology of attack team comprises a hierarchy of notions specifying the activities of attack team on different layers of detail. In this ontology, the hierarchy of nodes splits into two subsets according to the macro- and micro-layers of the domain specifications. All nodes of the ontology are divided into intermediate and terminal. The nodes specifying a set of methods for generating DDoS attacks and main network notions make up a top level of the ontology. At lower levels, different classes of DoS-attacks and notions needed to implement the attack are detailed. A low-level fragment of attack ontology representing the classes of agents and their properties is depicted in figure 2. The low-level fragments of ontologies have been developed using MASDK toolkit [9].

In the phase of deploying the agents, they get the following properties: (1) Master properties (MasterProps): the port for interaction, the IP address of agent’s host; (2) Daemon properties (DaemonProps): the port for attack execution, the port for interaction, the IP

Fig.2. The fragment of attack ontology (the screenshot of MASDK ontology editor)
address of agent’s host. The agents are identified by the attributes
agent_class_name_id. The identifiers are used for interacting between the agents. The
master forms the attack parameters that are defined by the Attack notion. The attack
parameters are: intensity, start of attack (yes/no), the IP address of attacked host and
its port. The daemon stores and uses these parameters.

Agent “Daemon”: “Executor” Role. The daemon tries to receive and execute the
master’s commands. Depending on situation its local goal may be: receive a
command from master; execute a command (start the DoS attack on the given host;
finish the attack; change the attack intensity); send to the “master” the
message about ability to work.

The agent can interact with the master
in a coordinator role. In figure 3 it is
represented what protocol can be used for
interaction of agents with different roles.

The master plays a coordinator role
(this fact is marked in figure by rhomb
on the cross of lines directed from Master
and Coordinator). The master initiates
interaction (in figure this is marked by
triangle on the cross of lines directed
from Master and AttackProtocol).

In the developed prototype the message from the masters to daemon has the
following format:

<table>
<thead>
<tr>
<th>Start the attack: yes/no</th>
<th>IP address of attack target</th>
<th>Port of attack target</th>
<th>Intensity of attack (in packets per second)</th>
</tr>
</thead>
</table>

Agent in an executor role receives the message while listening to the given TCP
port. If the “start the attack” field has the value “yes” the agent starts the DoS attack
on the given host with given intensity. The message with label “no” serves for
stopping the attack. The resources of executor role are a part of host resources,
including processor time, memory domain and network resources used by given ports.
When losing the control on one of these resources the agent stops to function.

It is necessary to assign a set of parameters that affects on the agent behavior with
given role and also a set of variables that describes this process. The execution of DoS
attack “UDP flood” for agent in an executor role consists in sending UDP-packets on
the given address with given intensity (rate). The main variables that describe the
agents’ activity are the IP address and port of attack target and the intensity of packet
sending. The intensity can be changed for lowering channel bandwidth without its
denial of service. Port numbers are set when the agent starts working.

Agent “Master”: “Coordinator” Role. The master coordinates the activities of
agents playing a role executor. Depending on situation the local goals of master can be:
receiving the commands from the malefactor; forming the messages for agents in
the executor role; processing the messages about the state of agents in the daemon
role; sending the messages for agent in the executor role.

The agent can interact with the agent of daemon class in an executor role (figure
3). The master acts as an initiator of interaction. The daemon needs the following data
from malefactor: the moments of time to start and stop the attack; the IP address and port of attacked host; the intensity of attack. These parameters can be given by malefactor due to user interface. Further the master forms the message and sends it to executors. He knows beforehand the addresses and listened ports for all agents playing an executor role.

4 Modeling and Simulation of Defense Agents’ Team

The analysis of defense systems against DDoS attacks allow to discover the following their important features: (1) Defense systems are built on the basis of various components (sensors, filters, load balancers, queues, etc.) that fulfill different subtasks but serve to joint defense task; (2) The component set and functionality of defense systems depend on the place of the system deployment (external network, routers, gates, firewalls, internal network, servers, etc.); (3) Defense systems have several processing levels (initial processing of traffic, generating statistical information, detection, filtering, load balancing, tracing, etc.) on which the particular sub-tasks of complex defense task are solved.

The common approach to defense against DDoS attacks consists in the following. Sensors collect information about the normal network traffic for building a model of normal traffic. Then a special analyzer compares in real-time the current traffic with the model of normal traffic. The defense system tries to detect anomalies, trace back their sources and generates the recommendations of how to cut off or to lower the malicious traffic. The system administrator chooses countermeasures that are implemented by protection components.

The agents of defense team have the common joint goal: to defense the given host or network from a DDoS attack. According to common approach, the following classes of defense agents can be set (figure 4): “Sensor” (agent of initial processing of information); “Detector” (agent of attack detection); “Filter” (agent of filtering); “Investigator” (agent of investigation and forensic). Each class of agents is represented in the figure only by one instance.

![Fig.4. The generalized structure of defense team.](image)

In the initial stage the agents are deployed on the corresponding to their roles hosts: sensor – on the way of traffic passing to defended host; detector – on any host of defended subnet; filter – on the entrance to the defended subnet; investigator – on any available host beyond the defended subnet. The defense team includes a number of sensors. Sensor process information about network packets, collects statistic data, determines the size of overall traffic and the addresses of n hosts that make the greatest traffic. The detection agents (detectors) make decision if a danger of DDoS attack exists and what hosts are the attack sources. The filtering agents (filters) can
use different mechanisms of filtering the malicious packets. Investigators try to trace back the sources of DDoS attacks and neutralize them by defeating (“killing”) corresponding attack agents. In addition, there can be used also the agents – managers interacting with a security administrator and configuring the defense system.

The developed ontology comprises a hierarchy of notions specifying activities of team of defense agents directed to protection from attacks on different layers of detail. The nodes determining the high levels of defense mechanisms (system, network, global) and main network notions make up the top level of the ontology. The high level fragment of DDoS defense mechanisms ontology is depicted in figure 5. At the bottom levels of the ontology these nodes are described by particular defense mechanisms [21, 33]. Different types of nodes corresponding to system level defense mechanisms can be used. For example, scanning tools check presence of DDoS-agents in the host file system, and also scan the ports frequently used by attackers. Mechanisms of client bottlenecks are directed on creating bottleneck processes on the zombie hosts used for DDoS-attacks to limit their attacking ability. Mechanisms of moving target defense consist in changing IP address to avoid being attacked.

The low level fragment of defense ontology is represented in figure 6. The goal of defense team is to protect the host TargetHost with given IP-address from DDoS attacks. The notion TargetPort contains the open ports of host TargetHost. The attribute host_id is the host identifier of the host. The attributes of notion agent name (class) Props (including InvestigationProps, FilterProps, DetectorProps and SensorProps) for all agents are as follows: IP address of agents’ host (IP_address) and the port for interaction (port). The attributes agent_name (class) id are the agent identifiers.

![Diagram of Defense Mechanisms](image)

**Fig.5.** The high-level fragment of DDoS mechanisms ontology
Agent “Sensor”: “Collector of Statistics” Role. Sensor can have the following sub-goals: initial processing of network packets; statistics collection by calculation of input traffic parameters from all external hosts together and separately; determining of the hosts that make the most intensive traffic; forming and sending the messages to detector about traffic statistics and chosen hosts. Figure 7 shows what protocol can be used for interaction. Sensor is an interaction initiator. Sensor forms the new portion of statistics and determines the five hosts with the greatest traffic every $k$ seconds. If traffic from these hosts would be blocked later, the agent will determine next five hosts. Then the addresses of the hosts and amount of their traffic are sent to detector. The amount of overall traffic is also sent. The address and the port of receiver are known beforehand.

Agent “Detector”: “Detector” role. The goal of the detector agent is to detect the attack. The local sub-goals may be: receiving and processing the messages from sensor; making decisions about detection of attack; forming and sending the messages to other agents. Playing this role the agent can interact with sensor in role “statistics collector”, filter and investigator in corresponding roles. The agent initiates the interaction with the last two agents (see figure 7). Every $k$ second a detector receives from sensor the statistics about traffic. If detector determines that BPS is more than 80% from maximum channel bandwidth, it determines the DDoS attack. The detector sends its decision and five addresses to filter and investigator. The addresses and ports of recipients are known beforehand.

Agent “Filter”: “Filter” Role. The agent filters the traffic dropping malicious packets directed to defended host or net. The local sub-goal may be: receiving the messages from detector; filtering packets on the basis of data from detector. The agent can interact with detector. The initiator of interaction is detector (see figure 7). Every $k$ seconds a filter receives from detector the data. If it is said in the message that
DDoS attack takes place, then filter begins to drop the packets from given IP addresses. Filter is a program that is deployed on the router (on the entrance to defended subnet).

![Fig.7. The meta-model of defense team (the screenshot of MASDK meta-model editor)](image)

**Agent of “Investigation”: “Investigation” Role.** The agent goal is to identify the attack agents and to defeat them. The investigator examines if the hosts with given addresses contain the attack agents and tries to “kill” them. Thus the local sub-goals may be: receiving and processing the messages from detector; search for suspicious hosts; defeating the attack agents on suspicious hosts. Playing this role the agent can interact with detector. Detector serves as interaction initiator (see figure 7).

### 5 Case-study: Example of Attack and Defense Simulation

To choose the network simulation tool the analysis of existing network simulators (including NS2, OMNeT++ [24], SSF Net, J-Sim and others) was made. The main requirements that were used to choose a network simulator are as follows: the detailed implementation of various protocols that are engaged in DDoS attacks (from the network layer and higher) in order to implement the main known DDoS attacks; the possibility to write and attach own modules for implementing the agent-based approach; the possibility to change the simulation parameters during simulation; the simulator implementation for Windows and Linux and requirement of cross-platform; advanced graphic user interface; free for use in scientific research.

In the paper we present the results of network simulation generated by using the OmnetPP INET Framework.

One of network fragments used for simulation is depicted in figure 8. The hosts designated as “cli[i]” are personal computers connected to the Internet. The connection speed is 100Mb/s. The hosts designated as “r-j” are routers. They are connected to each other with speed 512Mb/s. “Srv” is a (Web) server available to clients from the Internet. It has the services on TCP and UDP protocols. “Firewall” is a firewall deployed for secure access to the server “srv” from the Internet. The connection speed of “firewall” and “srv” is 100Mb/s.
On the initial phase of simulation the “Attack” and “Defense” teams are created. The “attack” team contains five agents of class “daemon” and one agent of class “master”. The “daemons” are deployed on the hosts “cli[1]”–“cli[5]”, “master” – on the host “cli[0]” (figure 8).

Fig.8. Structure of computer network used for simulation

Master contains the following modules (fig. 9a): ad_tcpapp – a TCP application controlled by the agent; a_masterdrv – a driver managing all components included in agent structure. Each “master” host contains the following modules (figure 9b): Ppp – is responsible for the data link layer; networkLayer – is responsible for network layer; pingApp – is responsible for applications that use ICMP; tcp – is the module serving TCP; udp – is the module serving UDP; tcpApp[0] – is agent “master”. As it is showed in figure 9b, the “master” host has IP address «111.222.0.3» and connection speed 10Mb (marked with blue font). The amounts of received (rcv:0) and sent (snt:0) packets are also depicted.

The defense team consists of two “sensors”, “detector”, “filter” and agent of “investigation”. The host under defense is “srv”. The first “sensor” is deployed on the host “r2”, the second – on the host “firewall”, “detector”– on the host “cli[10]”, “filter” – on the host “r2” and the agent of “investigation” – on the host “cli[9]”. The first “sensor” processes the traffic which is external for the defended sub-network (before the “r2” router, “filter” and “firewall”). The second “sensor” processes the internal traffic (after “filter” and “firewall”).
The defense agents are deployed similarly to the attack agents (figure 9). All defense agents use the TCP port #3000 for interaction. All attack agents use the TCP port #2000 for interaction and “daemons” use UDP port #2001 for sending the attack packets. Hosts “cli[6]”–“cli[8]” are used for generating the background traffic between legitimate clients and server. The size of background traffic is not more than 20% of server channels bandwidth.

After creating the agents’ teams in some moment of time, the malefactor sends to the “master” the command to attack the host with IP address “145.236.0.17” (“srv”) on the port “17” with intensity “100 packets per second”. Then the “master” sends the attack command to all “daemons”. “Daemons” receive these messages. On the basis of received data, the corresponding parameters of “daemons” are changed (figure 10).

The main parameters that define the “daemons” attack are as follows: local_port – the port for sending the attack packets; dest_port – the port of the attacked host; message_length – the length of attack packet in bits; message_freq – the time interval between sending the packets; dest_addresses – the IP address of the attacked host.

If the “UDP flood” attack is set then “daemons” begin to send UDP packets to the server “srv”. In figure 11 these packets are marked with yellow arrows. The red circle is a packet just sent by a “daemon” from host “cli[4]” to “srv”. While performing attack, “daemons” are sending periodically to the “master” messages about their ability to work. Receiving the messages from “daemons”, the “master” controls the progress of DDoS attack.

During functioning the defense agents’ team, “sensors” are sending to the “detector” with given rate (for example, every 10 seconds) the statistic data about
traffic. In figure 11, the yellow arrow designates packets with the statistic data sent from the “sensor” located on the “firewall” to the “detector” on the host “cli[10]”.

After the beginning of attack, a sudden increase of malicious traffic (directed to “srv”) happens. The $BPS$ parameter exceeds 80% from channel 100Mb bandwidth. The “detector” receives the information about the network situation from “sensors”. This information contains $BPS$ and the list of the five most active hosts. As $BPS$ is exceeded, the “detector” makes decision about attack detection and sends the message about this fact to the “filter” and the agent of “investigation”.

The “filter” starts to drop the packets directed from “cli[0]”–“cli[5]” to “srv”. This is the reason why the size of internal subnet traffic (after “filter”) lowers.

After a short period of time, on the basis of data from internal “sensor”, the “detector” makes decision that there are normal traffic conditions ($BPS < 80\%$ from 100Mb). But on the basis of data from external “sensor” (deployed on the “r2” host), the “detector” “sees” that the attack is not finished.

The agent of “investigation” tries to discover and defeat the attack agents. At some moment of time, it succeeded in discovering the tracks of two attack agents deployed on the hosts “cli[1]” and “cli[3]”. Later the agent of “investigation” (cloning on the host “cli[3]” the components of defeating attack agents) succeeds to defeat (“kill”) the agent “daemon” located on the host “cli[3]”.

As a result, the malicious traffic directed to “srv” (before “filter”) essentially lowers. The agent “detector” changes the rules of filtering for the agent “filter”. At this moment the joint goal of defense team is considered as fulfilled.

After the fixed period of time the “master” does not receives the message about ability to work from “daemon” from the host “cli[3]”. Therefore to restore the needed intensity of attack, the “master” makes decision to increase the intensity of sending the malicious packets by other “daemons”. The “master” sends to the residuary “daemons” the attack command with enlarged intensity parameter. “Daemons” change the attack rate. This change causes the corresponding reaction of defense agents. The simulation continues further (till the given moment of time) representing the different “steps” of counteracting agents’ teams.
6 Conclusion

In the paper we investigated basic ideas of the modeling and simulation of agents’ competition in the Internet between the teams of software agents for network resources availability. The technology for creation of the agents’ team was suggested and described. We developed the approach to be used for investigating different aspects of competition of teams in network environment. We presented the structure of a team of agents, specifications of hierarchies of agent plans, agent interaction-and-coordination mechanisms, and agent role-assignment mechanisms. Software prototype was developed using the OmnetPP INET Framework [24], Visual C++ and MASDK [9]. We imitated different classes of DDoS attacks and defense mechanisms. Experiments with the prototype have been conducted, including the investigation of attack scenarios against networks with different structures and security policies.

Our future theoretical work is directed on development of formal basis for agent-based modeling and simulation of antagonistic agents’ teams on an example of agents’ competition in the Internet. Our future practical goal is conducting experiments to both evaluate computer network security and analyze the efficiency
and effectiveness of security policy against different DDoS attacks. So the further development of our modeling and simulation framework and software tools will consist of developing more realistic environment, including improvement of capabilities of the attack and defense agents teams by expansion of the attack and defense mechanisms classes, and implementing more sophisticated attack and defense scenarios.

Acknowledgment

This research is being supported by grants 04-01-00167 of Russian Foundation of Basic Research and partly funded by the EC as part of the POSITIF project (contract IST-2002-002314).

References


Analyzing Dangers in Multiagent Rescue using DEFACTO

Janusz Marecki¹, Nathan Schurr², Milind Tambe³, and Paul Scerri⁴

¹ University of Southern California
Los Angeles, CA 90089
marecki@usc.edu

² University of Southern California
Los Angeles, CA 90089
schurr@usc.edu

³ University of Southern California
Los Angeles, CA 90089
tambe@usc.edu

⁴ Carnegie Mellon University
Pittsburgh, PA 15213
pscerri@cs.cmu.edu

Abstract. Enabling interactions of agent-teams and humans for safe and effective Multiagent rescue is a critical area of research, with encouraging progress in the past few years. However, previous work suffers from three key limitations: (i) limited human situational awareness, reducing human effectiveness in directing agent teams, (ii) the agent team’s rigid interaction strategies that jeopardize the rescue operation, and (iii) lack of formal tools to analyze the impact of such interaction strategies. This paper presents a software prototype called DEFACTO (Demonstrating Effective Flexible Agent Coordination of Teams through Omnipresence). DEFACTO is based on a software proxy architecture and 3D visualization system, which addresses the three limitations mentioned above. First, the 3D visualization interface enables human virtual omnipresence in the environment, improving human situational awareness and ability to assist agents. Second, generalizing past work on adjustable autonomy, the agent team chooses among a variety of “team-level” interaction strategies, even excluding humans from the loop in extreme circumstances. Third, analysis tools help predict the dangers of using fixed strategies for various agent teams in a future disaster response simulation scenario.

1 Introduction

Multi agent safety addressed in this paper is interpreted in the way that a team of agents ensures the safety of civilians or buildings in case of an emergency situation. Analyzing the safety of using multi agent teams interacting with humans is critical in a large number of current and future applications[2, 5, 14, 3]. For example, current efforts emphasize humans collaboration with robot teams in space explorations, humans teaming with robots and agents for disaster rescue, as well as humans collaborating with multiple software agents for training [4, 6].
This paper focuses on the challenge of improving the effectiveness and analysing the dangers of human collaboration with agent teams. Previous work has reported encouraging progress in this arena, e.g., via proxy-based integration architectures [10], adjustable autonomy [13, 4] and agent-human dialogue [1]. Despite this encouraging progress, previous work suffers from three key limitations. First, when interacting with agent teams acting remotely, human effectiveness is hampered by low-quality interfaces. Techniques that provide telepresence via video are helpful [5], but cannot provide the global situation awareness. Second, agent teams have been equipped with adjustable autonomy (AA) [14] but not the flexibility critical in such AA. Indeed, the appropriate AA method varies from situation to situation. In some cases the human user should make most of the decisions. However, in other cases human involvement may need to be restricted. Such flexible AA techniques have been developed in domains where humans interact with individual agents [13], but whether they apply to situations where humans interact with agent teams is unknown. Third, current systems lack tools to analyze the impact of human involvement in agent teams, yet these are key to flexible AA reasoning.

We report on a software prototype system, DEFACTO (Demonstrating Effective Flexible Agent Coordination of Teams through Omnipresence), that enables agent-human collaboration and addresses the three shortcomings outlined above. First, DEFACTO incorporates a visualizer that allows the human to have an omnipresent interaction with remote agent teams. We refer to this as the Omni-Viewer, and it combines two modes of operation. The Navigation Mode allows for a navigable, high quality 3D visualization of the world, whereas the Allocation Mode provides a traditional 2D view and a list of possible task allocations that the human may perform. Human experts can quickly absorb on-going agent and world activity, taking advantage of both the brain’s favored visual object processing skills (relative to textual search, [9]), and the fact that 3D representations can be innately recognizable, without the layer of interpretation required of map-like displays or raw computer logs. The Navigation mode enables the human to understand the local perspectives of each agent in conjunction with the global, system-wide perspective that is obtained in the Allocation mode.

Second, to provide flexible AA, we generalize the notion of strategies from single-agent single-human context [13]. In our work, agents may flexibly choose among team strategies for adjustable autonomy instead of only individual strategies; thus, depending on the situation, the agent team has the flexibility to limit human interaction, and may in extreme cases exclude humans from the loop. Third, we provide a formal mathematical basis of such team strategies. These analysis tools help agents in flexibly selecting the appropriate strategy for a given situation.

We present results from detailed experiments with DEFACTO, which reveal two major surprises. First, contrary to previous results [14], human involvement is not always beneficial to an agent team— despite their best efforts, humans may sometimes end up hurting an agent team’s performance. Second, increasing the number of agents in an agent-human team may also degrade the team performance, even though increasing the number of agents in a pure agent team under identical circumstances improves team performance. Fortunately, in both surprising instances above, DEFACTO’s flexible AA strategies alleviate such problematic situations.
2 DEFACTO System Details

DEFACTO consists of two major components: the Omni-Viewer and a team of proxies (see Figure 1). The Omni-Viewer allows for global and local views. The proxies allow for team coordination and communication, but more importantly also implement flexible human-agent interaction via Adjustable Autonomy. Currently, we have applied DEFACTO to a disaster rescue domain. The incident commander of the disaster acts as the user of DEFACTO. This disaster can either be “man made” (terrorism) or “natural” (earthquake). We focus on two urban areas: a square block that is densely covered with buildings (we use one from Kobe, Japan) and the University of Southern California campus, which is more sparsely covered with buildings. In our scenario, several buildings are initially on fire, and these fires spread to adjacent buildings if they are not quickly contained. The goal is to have a human interact with the team of fire engines in order to save the most buildings. Our overall system architecture applied to disaster response can be seen in Figure 1. While designed for real world situations, DEFACTO can also be used as a training tool for incident commanders when hooked up to a simulated disaster scenario.

Fig. 1. DEFACTO system applied to a disaster rescue.
2.1 Omni-Viewer

Our goal of allowing fluid human interaction with agents requires a visualization system that provides the human with a global view of agent activity as well as showing the local view of a particular agent when needed. Hence, we have developed an omnipresent viewer, or Omni-Viewer, which will allow the human user diverse interaction with remote agent teams. While a global view is obtainable from a two-dimensional map, a local perspective is best obtained from a 3D viewer, since the 3D view incorporates the perspective and occlusion effects generated by a particular viewpoint. The literature on 2D- versus 3D-viewers is ambiguous. For example, spatial learning of environments from virtual navigation has been found to be impaired relative to studying simple maps of the same environments [11]. On the other hand, the problem may be that many virtual environments are relatively bland and featureless. Ruddle points out that navigating virtual environments can be successful if rich, distinguishable landmarks are present [12].

To address our discrepant goals, the Omni-Viewer incorporates both a conventional map-like 2D view, Allocation Mode (Figure 2-c) and a detailed 3D viewer, Navigation Mode (Figure 2-a). The Allocation mode shows the global overview as events are progressing and provides a list of tasks that the agents have transferred to the human. The Navigation mode shows the same dynamic world view, but allows for more freedom to move to desired locations and views. In particular, the user can drop to the virtual ground level, thereby obtaining the world view (local perspective) of a particular agent. At this level, the user can “walk” freely around the scene, observing the local logistics involved as various entities are performing their duties. This can be helpful in evaluating the physical ground circumstances and altering the team’s behavior accordingly. It also allows the user to feel immersed in the scene where various factors (psychological, etc.) may come into effect.

In order to prevent communication bandwidth issues, we assume that a high resolution 3D model has already been created and the only data that is transferred during the disaster are important changes to the world. Generating this suitable 3D model environment for the Navigation mode can require months or even years of manual modeling effort, as is commonly seen in the development of commercial video-games. However, to avoid this level of effort we make use of the work of You et. al. [15] in rapid, minimally...
assisted construction of polygonal models from LiDAR (Light Detection and Ranging) data. Given the raw LiDAR point data, we can automatically segment buildings from ground and create the high resolution model that the Navigation mode utilizes. The construction of the USC campus and surrounding area required only two days using this approach. LiDAR is an effective way for any new geographic area to be easily inserted into the Omni-Viewer.

We use the JME game engine to perform the actual rendering due to its cross-platform capabilities. JME is an extensible library built on LWJGL (Light Weight Java Game Library), which interfaces with OpenGL and OpenAL. This environment easily provided real-time rendering of the textured campus environment on mid-range commodity PCs. JME utilizes a scene graph to order the rendering of geometric entities. It provides some important features such as OBJ format model loading (which allows us to author the model and textures in a tool like Maya and load it in JME) and also various assorted effects such as particle systems for fires.

2.2 Proxy: Teamwork and Adjustable Autonomy

We have built teams based on previous proxy software [13], that is in the public domain. The proxies were extended to our domain in order to take advantage of existing methods of communication, coordination, and task allocation for the team. However, these aspects are not the focus of this paper.

Instead, we focus on another key aspect of the proxies: Adjustable Autonomy. Adjustable autonomy refers to an agent’s ability to dynamically change its own autonomy, possibly to transfer control over a decision to a human. Previous work on adjustable autonomy could be categorized as either involving a single person interacting with a single agent (the agent itself may interact with others) or a single person directly interacting with a team. In the single-agent single-human category, the concept of flexible transfer-of-control strategy has shown promise [13]. A transfer-of-control strategy is a preplanned sequence of actions to transfer control over a decision among multiple entities, for example, an $AH_1H_2$ strategy implies that an agent ($A_T$) attempts a decision and if the agent fails in the decision then the control over the decision is passed to a human $H_1$, and then if $H_1$ cannot reach a decision, then the control is passed to $H_2$.

Since previous work focused on single-agent single-human interaction, strategies were individual agent strategies where only a single agent acted at a time.

An optimal transfer-of-control strategy optimally balances the risks of not getting a high quality decision against the risk of costs incurred due to a delay in getting that decision. Flexibility in such strategies implies that an agent dynamically chooses the one that is optimal, based on the situation, among multiple such strategies ($H_1A$, $AH_1$, $AH_1A$, etc.) rather than always rigidly choosing one strategy. The notion of flexible strategies, however, has not been applied in the context of humans interacting with agent-teams. Thus, a key question is whether such flexible transfer of control strategies are relevant in agent-teams, particularly in a large-scale application such as ours.

DEFACTO aims to answer this question by implementing transfer-of-control strategies in the context of agent teams. One key advance in DEFACTO, however, is that the strategies are not limited to individual agent strategies, but also enables team-level strategies. For example, rather than transferring control from a human to a single agent,
a team-level strategy could transfer control from a human to an agent-team. Concretely, each proxy is provided with all strategy options; the key is to select the right strategy given the situation. An example of a team level strategy would combine $A_T$ Strategy and $H$ Strategy in order to make $A_T H$ Strategy. The default team strategy, $A_T$, keeps control over a decision with the agent team for the entire duration of the decision. The $H$ strategy always immediately transfers control to the human. $A_T H$ strategy is the conjunction of team level $A_T$ strategy with $H$ strategy. This strategy aims to significantly reduced the burden on the user by allowing the decision to first pass through all agents before finally going to the user, if the agent team fails to reach a decision.

3 Mathematical Model of Strategy Selection

We develop a novel mathematical model for these team level adjustable autonomy strategies in order to enable team-level strategy selection. We first quickly review background on individual strategies from Scerri [13] before presenting our team strategies. Whereas strategies in Scerri’s work are based on a single decision that is sequentially passed from agent to agent, we assume that there are multiple homogeneous agents concurrently working on multiple tasks interacting with a single human user. We exploit these assumptions (which capture the features of our domain) to obtain a reduced version of our model and simplify the computation in selecting strategies.

3.1 Background on individual strategies

A decision, $d$, needs to be made. There are $n$ entities, $e_1 \ldots e_n$, who can potentially make the decision. These entities can be human users or agents. The expected quality of decisions made by each of the entities, $EQ = \{EQ_{e_i,d}(t) : \mathcal{R} \rightarrow \mathcal{R}\}_{i=1}^n$, is known, though perhaps not exactly. $P = \{P_T(t) : \mathcal{R} \rightarrow \mathcal{R}\}$ represents continuous probability distributions over the time that the entity in control will respond (with a decision of quality $EQ_{e,d}(t)$). The cost of delaying a decision until time $t$, denoted as $\{W : t \rightarrow \mathcal{R}\}$. The set of possible wait-cost functions is $W$. $W(t)$ is non-decreasing and at some point in time, $\Gamma$, when the costs of waiting stop accumulating (i.e., $\forall t \geq \Gamma, \forall W \in W, W(t) = W(\Gamma)$).

To calculate the EU of an arbitrary strategy, the model multiplies the probability of response at each instant of time with the expected utility of receiving a response at that instant, and then sum the products. Hence, for an arbitrary continuous probability distribution if $e_c$ represents the entity currently in decision-making control:

$$EU = \int_0^{\infty} P_T(t) EU_{e_c,d}(t) \cdot dt$$

Since we are primarily interested in the effects of delay caused by transfer of control, we can decompose the expected utility of a decision at a certain instant, $EU_{e_c,d}(t)$, into two terms. The first term captures the quality of the decision, independent of delay costs, and the second captures the costs of delay: $EU_{e_c,d} = EQ_{e,d}(t) - W(t)$. To calculate the EU of a strategy, the probability of response function and the wait-cost calculation must reflect the control situation at that point in the strategy. If a human, $H_1$ has control at time $t$, $P_T(t)$ reflects $H_1$’s probability of responding at $t$. 

3.2 Introduction of team level strategies

**AT Strategy:** Starting from the individual model, we introduce team level \( AT \) strategy, denoted as \( AT \) in the following way: We start with Equation 2 for single agent \( AT \) and single task \( d \). We obtain Equation 3 by discretizing time, \( t = 1, ..., T \) and introducing set \( \Delta \) of tasks. Probability of agent \( AT \) performing a task \( d \) at time \( t \) is denoted as \( P_{a,d}(t) \). Equation 4 is a result of the introduction of the set of agents \( AG = a_1, a_2, ..., a_k \). We assume the same quality of decision for each task performed by an agent and that each agent \( AT \) has the same quality so that we can reduce \( EQ_{a,d}(t) \) to \( EQ(t) \). Given the assumption that each agent \( AT \) at time step \( t \) performs one task, we have \( \sum_{d \in \Delta} P_{a,d}(t) = 1 \) which is depicted in Equation 5. Then we express \( \sum_{a=a_1}^{a_k} \sum_{d \in \Delta} P_{a,d}(t) \times W_{a,d}(t) \) as the total team penalty for time slice \( t \), i.e., at time slice \( t \) we subtract one penalty unit for each not completed task as seen in Equation 6. Assuming penalty unit \( PU = 1 \) we finally obtain Equation 7.

\[
EU_{a,d} = \int_0^\infty P_{\gamma a}(t) \times (EQ_{a,d}(t) - W(t)) \, dt \tag{2}
\]

\[
EU_{a,\Delta} = \sum_{t=1}^{T} \sum_{d \in \Delta} P_{a,d}(t) \times (EQ_{a,d}(t) - W_{a,d}(t)) \tag{3}
\]

\[
EU_{AT,\Delta} = \sum_{t=1}^{T} \sum_{a=a_1}^{a_k} \sum_{d \in \Delta} P_{a,d}(t) \times (EQ_{a,d}(t) - W_{a,d}(t)) \tag{4}
\]

\[
EU_{AT,\Delta,AG} = \sum_{t=1}^{T} (|AG| \times EQ(t) - (|\Delta| - |AG|) \times t) \times PU \tag{5}
\]

\[
EU_{AT,\Delta,AG} = |AG| \times \sum_{t=1}^{T} (EQ(t) - (|\Delta| - |AG|) \times t) \tag{6}
\]

**H Strategy:** The difference between \( EU_{H,\Delta,AG} \) and \( EU_{AT,\Delta,AG} \) results from three key observations: First, the human is able to choose strategic decisions with higher probability, therefore his \( EQ_H(t) \) is greater than \( EQ(t) \) for both individual and team level \( AT \) strategies. Second, we hypothesize that a human cannot control all the agents \( AG \) at disposal, but due to cognitive limits will focus on a smaller subset, \( AG_H \) of agents (evidence of limits on \( AG_H \) appears later in Figure 5-a). \( |AG_H| \) should slowly converge to \( B \), which denotes its upper limit, but never exceed \( AG \). Each function \( f(AG) \) that models \( AG_H \) should be consistent with three properties: i) if \( B \to \infty \) then \( f(AG) \to AG \); ii) \( f(AG) < B \); iii) \( f(AG) < AG \). Third, there is a delay in human decision making compared to agent decisions. We model this phenomena by shifting \( H \)
to start at time slice $t_H$. For $t_H - 1$ time slices the team incurs a cost $|\Delta| \times (t_H - 1)$ for all incomplete tasks. By inserting $EQ_H(t)$ and $AG_H$ into the time shifted utility equation for $A_T$ strategy we obtain the $H$ strategy (Equation 8).

**A_T-H Strategy:** The $A_T-H$ strategy is a composition of $H$ and $A_T$ strategies (see Equation 9).

$$EU_{H, \Delta, AG} = |AG_H| \times \sum_{t=t_H}^{T} (EQ_H(t) - |\Delta| \times (t_H - 1))$$

$$EU_{A_T-H, \Delta, AG} = |AG| \times \sum_{t=1}^{t_H-1} (EQ(t) - (|\Delta| \times |AG|))$$

$$EU_{A_T-H, \Delta, AG} = |AG_H| \times \sum_{t=t_H}^{T} (EQ_H(t) - (|\Delta| \times |AG_H|) - (t - t_H)))$$

**Strategy utility prediction:** Given our strategy equations and the assumption that $EQ_{H, \Delta, AG}$ is constant and independent of the number of agents we plot the graphs representing strategy utilities (Figure 3). Figure 3 shows the number of agents on the x-axis and the expected utility of a strategy on the y-axis. We focus on humans with different skills: (a) low $EQ_H$, low $B$ (b) high $EQ_H$, low $B$ (c) low $EQ_H$, high $B$ (d) high $EQ_H$, high $B$. The last graph representing a human with high $EQ_H$ and high $B$ follows results presented in [13] (and hence the expected scenario), we see the curve of $AH$ and $A_T-H$ flattening out to eventually cross the line of $A_T$. Moreover, we observe that the increase in $EQ_H$ increases the slope for $AH$ and $A_T-H$ for small number of agents, whereas the increase of $B$ causes the curve to maintain a slope for larger number of agents, before eventually flattening out and crossing the $A_T$ line.

### 4 Experiments and Evaluation

Our DEFACTO system was evaluated in three key ways, with the first two focusing on key individual components of the DEFACTO system and the last attempting to evaluate the entire system. First, we performed detailed experiments comparing the effectiveness of Adjustable Autonomy (AA) strategies over multiple users. In order to provide DEFACTO with a dynamic rescue domain we chose to connect it to a simulator. We chose the previously developed RoboCup Rescue simulation environment [8]. In this simulator, fire engine agents can search the city and attempt to extinguish any fires that have started in the city. To interface with DEFACTO, each fire engine is controlled by a proxy in order to handle the coordination and execution of AA strategies. Consequently, the proxies can try to allocate fire engines to fires in a distributed manner, but can also transfer control to the more expert user. The user can then use the Omni-Viewer in Allocation mode to allocate engines to the fires that he has control over. In order to focus
Fig. 3. Model predictions for various users.

(a) Low B, Low EQh  
(b) Low B, High EQh  
(c) High B, Low EQh  
(d) High B, High EQh

Fig. 4. Performance of subjects 1, 2, and 3.
on the AA strategies (transferring the control of task allocation) and not have the users ability to navigate interfere with results, the Navigation mode was not used during this first set of experiments.

The results of our experiments are shown in Figure 4, which shows the results of subjects 1, 2, and 3. Each subject was confronted with the task of aiding fire engines in saving a city hit by a disaster. For each subject, we tested three strategies, specifically, $H, AH$ and $ATH$; their performance was compared with the completely autonomous $AT$ strategy. $AH$ is an individual agent strategy, tested for comparison with $ATH$, where agents act individually, and pass those tasks to a human user that they cannot immediately perform. Each experiment was conducted with the same initial locations of fires and building damage. For each strategy we tested, varied the number of fire engines between 4, 6 and 10. Each chart in Figure 4 shows the varying number of fire engines on the x-axis, and the team performance in terms of numbers of building saved on the y-axis. For instance, strategy $AT$ saves 50 building with 4 agents. Each data point on the graph is an average of three runs. Each run itself took 15 minutes, and each user was required to participate in 27 experiments, which together with 2 hours of getting oriented with the system, equates to about 9 hours of experiments per volunteer.

Figure 4 enables us to conclude the following:

– *Human involvement with agent teams does not necessarily lead to improvement in team performance.* Contrary to expectations and prior results, human involvement does not uniformly improve team performance, as seen by human-involving strategies performing worse than the $AT$ strategy in some instances. For instance, for subject 3, human involving strategies such as $AH$ provide a somewhat higher quality than $AT$ for 4 agents, yet at higher numbers of agents, the strategy performance is lower than $AT$. While our strategy model predicted such an outcome in cases of High B, Low EQH, the expected scenario was High B, High EQH.

– *Providing more agents at a human’s command does not necessarily improve the agent team performance.* As seen for subject 2 and subject 3, increasing agents from 4 to 6 given $AH$ and $ATH$ strategies is seen to degrade performance. In contrast, for the $AT$ strategy, the performance of the fully autonomous agent team continues to improve with additions of agents, thus indicating that the reduction in $AH$ and $ATH$ performance is due to human involvement. As the number of agents increase to 10, the agent team does recover.

– *No strategy dominates through all the experiments given varying numbers of agents.* For instance, at 4 agents, human-involving strategies dominate the $AT$ strategy. However, at 10 agents, the $AT$ strategy outperforms all possible strategies for subjects 1 and 3.

– *Complex team-level strategies are helpful in practice.* $ATH$ leads to improvement over $H$ with 4 agents for all subjects, although surprising domination of $AH$ over $ATH$ in some cases indicates that $AH$ may also a useful strategy to have available in a team setting.

Note that the phenomena described range over multiple users, multiple runs, and multiple strategies. The most important conclusion from these figures is that flexibility is necessary to allow for the optimal AA strategy to be applied. The key question...
is then whether we can leverage our mathematical model to select among strategies. However, we must first check if we can model the phenomenon in our domain accurately. To that end, we compare the predictions at the end of Section 3 with the results reported in Figure 4. If we temporarily ignore the “dip” observed at 6 agents in \(AH\) and \(ATH\) strategies, then subject 2 may be modeled as a High \(B\), High \(EQ_H\) subject, while subjects 1 and 3 modeled via High \(B\), Low \(EQ_H\). (Figure 5-(b) indicates an identical improvement in \(H\) for 3 subjects with increasing agents, which suggests that \(B\) is constant across subjects.) Thus, by estimating the \(EQ_H\) of a subject by checking the “H” strategy for small number of agents (say 4), and comparing to \(A\) strategy, we may begin to select the appropriate strategy.

Unfortunately, the strategies including the humans and agents (\(AH\) and \(ATH\)) for 6 agents show a noticeable decrease in performance for subjects 2 and 3 (see Figure 4), whereas our mathematical model would have predicted an increase in performance as the number of agents increased (as seen in Figure 3). It would be useful to understand which of our key assumptions in the model has led to such a mismatch in prediction.

**Table 1.** Total amount of allocations given.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>(H)</th>
<th>(AH)</th>
<th>(ATH)</th>
</tr>
</thead>
<tbody>
<tr>
<td># of agents</td>
<td>4</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>Subject 1</td>
<td>91</td>
<td>92</td>
<td>154</td>
</tr>
<tr>
<td>Subject 2</td>
<td>138</td>
<td>129</td>
<td>180</td>
</tr>
<tr>
<td>Subject 3</td>
<td>117</td>
<td>132</td>
<td>152</td>
</tr>
</tbody>
</table>

![Fig. 5.](a) \(AG_H\) and (b) \(H\) performance

The crucial assumptions in our model were that while numbers of agents increase, \(AG_H\) steadily increases and \(EQ_H\) remains constant. Thus, the dip at 6 agents is essentially affected by either \(AG_H\) or \(EQ_H\). We first tested \(AG_H\) in our domain. The amount of effective agents, \(AG_H\), is calculated by dividing how many total allocations each subject made by how many the \(AT\) strategy made per agent, assuming \(AT\) strategy...
Fig. 6. Task allocation overload for the team of 6 agents

Fig. 7. Number of agents per fire assigned by subjects 1, 2, and 3
effectively uses all agents. Figure 5-(a) shows the number of agents on the x-axis and the number of agents effective used, $AG_H$, on the y-axis; the $A_T$ strategy, which is using all available agents, is also shown as a reference. However, the amount of effective agents is actually about the same in 4 and 6 agents. This would not account for the sharp drop we see in the performance. We then shifted our attention to the $EQ_H$ of each subject. One reduction in $EQ_H$ could be because subjects simply did not send as many allocations totally over the course of the experiments. This, however, is not the case as can be seen in Table 1 where for 6 agents, the total amount of allocations given is comparable to that of 4 agents. To investigate further, we checked if the quality of human allocation had degraded. For our domain, the more fire engines that fight the same fire, the more likely it is to be extinguished and in less time. For this reason, the number of agents that were tasked to each fire is a good indicator of the quality of allocations that the subject makes. Our model expected the number of agents that each subject tasked out to each fire would remain independent of the number of agents. Figure 7 shows the number agents on the x-axis and the average amount of fire engines allocated to each fire on the y-axis. $AH$ and $A_T H$ for 6 agents result in significantly less average fire engines per task (fire) and therefore less average $EQ_H$.

The next question is then to understand why for 6 agents $AH$ and $A_T H$ result in lower average fire engines per fire. One hypothesis is the possible interference among the agents’ self allocations vs human task allocations at 6 agents. Table 2 shows the number of task changes for 4, 6 and 10 agents for $AH$ and $A_T H$ strategies, showing that maximum occurs at 6 agents. A task change occurs because an agent pursuing its own task is provided another task by a human or a human-given task is preempted by the agent. Thus, when running mixed agent-human strategies, the possible clash of tasks causes a significant increase task changes, resulting in the total amount of task allocations overreaching the number of task allocations for the $A$ strategy (Figure 6). While the reason for such interference peaking at 6 may be domain specific, the key lesson is that interference has the potential to occur in complex team-level strategies. Our model would need to take into account such interference effects by not assuming a constant $EQ_H$.

### Table 2. Task conflicts for subject 2.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>4 agents</th>
<th>6 agents</th>
<th>10 agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AH$</td>
<td>34</td>
<td>75</td>
<td>14</td>
</tr>
<tr>
<td>$A_T H$</td>
<td>54</td>
<td>231</td>
<td>47</td>
</tr>
</tbody>
</table>

The second aspect of our evaluation was to explore the benefits of the Navigation mode (3D) in the Omni-Viewer over solely an Allocation mode (2D). We performed 2 tests on 20 subjects. All subjects were familiar with the university campus. Test 1 showed Navigation and Allocation mode screenshots of the university campus to subjects. Subjects were asked to identify a unique building on campus, while timing each response. The average time for a subject to find the building in 2D was 29.3 seconds,
whereas the 3D allowed them to find the same building in an average of 17.1 seconds. Test 2 again displayed Navigation and Allocation mode screenshots of two buildings on campus that had just caught fire. In Test 2, subjects were asked first to allocate fire engines to the buildings using only the Allocation mode. Then subjects were shown the Navigation mode of the same scene. 90 percent of the subjects actually chose to change their initial allocation, given the extra information that the Navigation mode provided.

5 Related Work and Summary

We have discussed related work throughout this paper, however, we now provide comparisons with key previous agent software prototypes and research. Given our application domains, Scerri et al’s work on robot-agent-person (RAP) teams for disaster rescue is likely the most closely related [13]. Our work takes a significant step forward in comparison. First, the omni-viewer enables navigational capabilities improving human situational awareness not present in previous work. Second, we provide a mathematical model based on strategies, which we experimentally verify, absent in that work. Third, we provide extensive experimentation, and illustrate that some of the conclusions reached in [13] were indeed preliminary, e.g., they conclude that human involvement is always beneficial to agent team performance, while our more extensive results indicate that sometimes agent teams are better off excluding humans from the loop. Human interactions in agent teams is also investigated in [15,2], and there is significant research on human interactions with robot-teams [5,3]. However they do not use flexible AA strategies and/or team-level AA strategies. Furthermore, our experimental results may assist these researchers in recognizing the potential for harm that humans may cause to agent or robot team performance. Significant attention has been paid in the context of adjustable autonomy and mixed-initiative in single-agent single-human interactions [7,1]. However, this paper focuses on new phenomena that arise in human interactions with agent teams.

This paper addresses the issue of safety in multi-agent systems interpreted in the way that a team of agents ensures the safety of civilians or buildings in case of an emergency situation. To this end, we present a large-scale prototype, DEFACTO, that is based on a software proxy architecture and 3D visualization system and provides three key advances over previous work. First, DEFACTO’s Omni-Viewer enables the human to both improve situational awareness and assist agents, by providing a navigable 3D view along with a 2D global allocation view. Second, DEFACTO incorporates flexible AA strategies, even excluding humans from the loop in extreme circumstances. Third, analysis tools help predict the behavior of the agent team and choose the safest strategy for the given domain.

We performed detailed experiments using DEFACTO, leading to some surprising results. These results illustrate that an agent team must be equipped with flexible strategies for adjustable autonomy, so that they may select the safest strategy autonomously.

References


Extending the Buddy Model to Secure Variable Sized Multi Agent Communities

John Page, Arkady Zaslavsky, Maria Indrawan

School of Computer Science & Software Engineering, Monash University, Melbourne
{john.page, arkady.zaslavsky, maria.indrawan}@infotech.monash.edu.au

Abstract: This paper describes an extension of the Buddy model of agent security. The Buddy model provides a security layer that encapsulates all agents within the multi-agent community and therefore, extends a security cover to all agents of the community. While the current model is able to service several scenarios involving fixed sized agent communities, it does not consider variable sized agent communities. This paper describes an extension to the existing model that covers this aspect. The proposed extension increases the application coverage and flexibility of the Buddy model and makes it a potent security mechanism for multi-agent communities. Further, the paper establishes that agent migration does not lead to any schema operating overheads; on the contrary it contributes to maintaining an effective level of performance integrity within the schema.

Keywords: Agents, Mobility, Security, Community, Buddy

Introduction

The popularity of web-based services using portable languages such as Java [4] and scripting languages such as TCL/TK [14] encouraged the development of the Mobile Agent (MA) paradigm. To describe the motivation behind the MA paradigm, researchers and developers have given several definitions. Some of these definitions have introduced new terms causing confusion between various forms of mobile code in use. Franklin & Graesser [7] have produced an interesting taxonomy of different agent architectures that were evolving around the time that Java was establishing itself as a web-based application development solution. Their study identified and summarised the various characteristics that were seen to form a part of agent behaviour.

Using the commonality of these characteristics, [7] proposed a definition that identified an agent as an autonomous system situated within an environment on which the agent continuously acted over time, so as to effect its future sensing. While there are several agent based toolkits [1] which are being used for educational as well as commercial application development, some of the early toolkits such as Aglets [8], Odyssey [2], Grasshopper [3], and Concordia [13] are still popular as interesting
sources of research for the agent paradigm. Using these toolkits, researchers have attempted to explore the somewhat complex dynamics of agent oriented software engineering and propose theories for the further evolution of the paradigm. One much researched area of agent oriented computing is security. The need for entities to interact with a minimum level of mutual trust is a well appreciated and much sought after goal.

While the presence of mobility has made the nature of agent oriented applications highly pervasive, it has also compromised the achievable level of security within the applications. Agent oriented transactions are highly susceptible to malicious tamper attacks [15] and the integrity of the transactions taking place can sometimes be questionable. To achieve an acceptable level of security, researchers have proposed various methods, some of which have been summarised by [9]. While the many theories and methods proposed by researchers may be effective to some extent, there is no clear single solution to beat the challenges posed by the security threats to the MA operation.

This paper examines the reasons behind the lack of a secure computing infrastructure for mobile agents. Further it summarises some of the possible threats faced by MAs from malicious entities. To counter these threats, the paper describes the Buddy model schema of agent security [5] and presents the schema operation as an effective measure of assessing threat levels in a distributed multi-agent community scenario. The paper discusses how mobility, commonly regarded as a cause for an increasing level of security vulnerability, is used by the Buddy model in enforcing greater reliability. The schema also promotes operating transparency between the MA and the agent server which is an effective trust building mechanism.

This paper is organised as follows. The section “Barriers to a Security Operation” describe the various factors that have prevented proposed security measures from being effective within mobile agent systems. This section also summarises some of the possible security threats faced by MAs. Further it also describes the relevance of the Buddy model schema in providing a security mechanism in distributed systems. “Buddy Model Schema Architecture” describes the architecture and the operation of the Buddy model. “Buddy Model Schema Operation” describes the implementation of the system and presents an analytical view in discussing agent migration and its effect on the schema’s operation. “Related Work” describes related work and compares the advantages of the schema by comparing it with other proposed agent security schemas. “Conclusion and Future Work” concludes the paper with a summary of the schema operation and the results presented in this paper. The section also provides a brief indication of future work. Henceforth for the remainder of the discussion the term agent and MA shall be used synonymously.

**Barriers to a Security Schema Operation**

MAs enable the development of pervasive applications. They can migrate from one system to another carrying with them information obtained along its travels. While this feature allows the agent operation to be highly flexible and gives the MA a chance to make the most optimal decisions with respect to its future execution, it also
makes the MA susceptible to malicious attacks that target its code and data. Security proposals for the MA become dust in the wind, if the host Mobile Agent Server (MAS) is malicious. The main drawback that prevents security measures from being successful is that the agent host has control over the execution environment. This allows it to inspect the agent code prior to executing it, thus allowing it to pre-empt the security trigger from firing within the agent. Thus, the MAS can bypass the agent security mechanism allowing it to compromise the agent at will.

A security mechanism that allowed the agent to generate its security mechanism at runtime could possibly have been successful but since the agent host possesses a copy of the serialized agent code, the agent host could replay the agent in a controlled environment and study the execution of the run-time security generation. Thus, when the agent would actually execute in the ‘real’ time environment, the agent server would not allow the security trigger to fire.

While in the past, some security proposals have involved cryptographic mechanisms, the argument against them becoming the de facto solution is a strong one too. Cryptographic mechanisms can be rendered useless if the encryption key employed is compromised. A further argument against the use of cryptographic techniques is the expense in the processing involved. For example, consider an agent operation that normally would use up five processing cycles for its business function. For its cryptographic function, the agent might need to use another five processing cycles. This will destroy the viability of the agent function as far as cost vs. benefit analysis is concerned.

The use of cryptographic techniques also creates a level of uncertainty for the MAS. For example, consider the case where in the agent host is not intending a malicious action on the arriving agent and is interested in providing the agent with its requested resources. If an arriving agent carries along with it a host of cryptographic mechanisms, it becomes difficult for the agent server to gauge the intentions of the agent. In such situations, it is even possible that the server may regard the agent as a malicious entity with an unclear agenda and deny the agent execution privileges. While the prevailing scenario does not encourage a solution that empowers an agent to take control of the execution environment, as this approach would be unacceptable to the agent server, the question of maintaining the integrity of the serialized agent code and the executing instance of the agent remains largely unanswered.

Agent community operations bring in an additional level of complexity to the already vulnerable state of the agent security dimension. Communities involve collaboration, communication and in the case of mobile communities, there is migration also that needs to be considered. In order to safeguard the interests of the community, the community operations require a critical level of security to be in place at all times as a failure at any one level could compromise the entire agent operation.

The Buddy Model [5, 10, 11] proposed a security schema for the operation of mobile agents while operating as a part of a community. While [10] laid down the rules governing the architecture, [11] demonstrated the schema’s ability to counter security threats in mobile business scenarios. The ability of the schema to involve all members of the agent community in the security function enabled a cost effective solution that spread the risk of failure equally among the community members. According to the Buddy model, the security function of each MA is backed up by two other agents,
referred to as Buddies. This implies that in order to compromise any one agent within
the community, a malicious attacker has to attack and compromise at least two other
agents, which are located on different physical machines. Thus, the difficulty level of
compromising the schema is inordinately high. This factor deters malicious
programmers making the schema a viable option for the protection of mobile agent
communities.

Currently MAs are being used to develop web-based applications [1]. In such
scenarios, security becomes a critical requirement. Most mobile agent toolkits
implement a two-pronged security defence. They rely on authentication and
authorization of agents to filter out the genuine agents from the rogue agents [6]. As
discussed in [10], while this approach provides the basic level of application based
security, it does not guarantee the protection of the system from hostile action as
malicious entities can still slip through using forged or stolen credentials. Thus, there
is an ever present need for MAS to employ a security mechanism that functions well
within the inner reach of the system and continuously monitors the key points of the
system for tampering attacks. Further more, the authentication and authorization
approach aims to protect the MAS from malicious agents but it does not contribute
towards extending the security cover to MAs. This leaves MAs vulnerable to the
malicious attacks of the host servers.

The Buddy model of security counters this drawback by providing a security cover
that continuously monitors the MAs. To determine the level of security within the
system, the schema provides a Confidence Factor (CF) function. This function can at
different instances in time, report the level of security present in the system in
tangible terms. The next section describes the operation and the proposed extension to
the existing model.

**Buddy Model Schema Architecture**

While the architecture and the different configuration models that evolve from the
schema have been discussed with detail in [10], this section offers a brief recap.
Before going into the schema operation, it would be useful to understand some
terminology used in the schema operation and to briefly recapitulate some of the
rules that govern the operation of the schema and reduce the risk factor of a hostile
action on the community members. Each cycle in the security schema has two
phases. Further, the contribution of each member of the community to the security
function is equal. In return all members are covered by the security schema over the
two schemas.

Agents within the Buddy model have a role based action. The schema identifies two
roles, namely Buddy agents and Protected Agents (PAs). PAs are the agents that are
being monitored during a phase while Buddy agents are the agents that are involved in
monitoring the PAs. It may be noted that an agent in the model might essay a dual
role at the same time. In other words, a PA might be the Buddy for another PA in the
model. In previous papers, Protected Agents (PAs) have been referred to as Central
Agents (CAs) [10]. As it has been discussed in [10], the schema is currently
applicable for only an even number of MAs in the community. Each configuration of
the model is identified by two numerals. The first numeral identifies the total number of PAs in one phase of the cycle. The second number gives the total number of Buddy agents in the phase.

Figure 1 describes the various interconnections in the first phase of the 7-14 configuration model of the schema. In the 7-14 model there are 14 agents deployed in the field. In each phase 7 agents perform the role of PA and some of them also act as Buddy agents for the other agents within the community. The remaining 7 agents function as exclusive Buddy agents for that particular phase. In the second phase, roles of the Buddy and the PA are reversed. Table 1 documents the agent roles undertaken by the community agents in the 7-14 model. In the table PA denotes Protected Agents and B denotes Buddy Agents. The number attached to PA and B is just to simplify the understanding of the model as it aids in keeping track of the agent in the field. In the schema implementation agents use names with numbers attached to them. For example CommunityAgentOne, CommunityAgentTwo are valid names in the model.

As evident from figure 1 and from the role breakdown described in table 1, all agents perform either of the two roles in some phase of the schema and hence act as protectors and also receive protection. The role assignment and the monitoring of agents in the field are done by the Home Base of the agent community. The implementation model described in the next section uses a single Home Base for the entire agent community. A centralised Home Base for the entire agent community has the advantage of making available a central repository and a reporting centre where information related to the community agents can be easily received and processed.

The Buddy model schema uses a few rules that ensure that there is uniformity in its application across the entire agent community that comes under its purview. In brief these rules can be summarized to state the following. All Protected Agents (PAs) in
the community shall have at least two Buddy agents for itself. The location of the Buddy and the PA shall never be the same at any point of time when the schema is in operation and finally an agent in the community can be a Buddy of only one other agent.

Table.1. Agent role breakdown in the 7-14 Model

<table>
<thead>
<tr>
<th>PA 1st Phase</th>
<th>Buddy 1st Phase</th>
<th>PA 2nd Phase</th>
<th>Buddy 2nd Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA1</td>
<td>B6,B7</td>
<td>PA8</td>
<td>B12,B14</td>
</tr>
<tr>
<td>PA2</td>
<td>B1,B5</td>
<td>PA9</td>
<td>B8,B13</td>
</tr>
<tr>
<td>PA3</td>
<td>B2,B4</td>
<td>PA10</td>
<td>B7,B10</td>
</tr>
<tr>
<td>PA4</td>
<td>B8,B9</td>
<td>PA11</td>
<td>B1,B6</td>
</tr>
<tr>
<td>PA5</td>
<td>B3,B10</td>
<td>PA12</td>
<td>B10,B11</td>
</tr>
<tr>
<td>PA6</td>
<td>B11,B12</td>
<td>PA13</td>
<td>B4,B5</td>
</tr>
<tr>
<td>PA7</td>
<td>B13,B14</td>
<td>PA14</td>
<td>B2,B3</td>
</tr>
</tbody>
</table>

Implementation of these rules is carried out by the static agents located at the Home Base of the agent community. After the agent community has been set up and the agent roles have been assigned, the agents are deployed according to pre-decided itineraries. At this point the Buddy model schema is initiated and the first phase is triggered.

In the first phase of the schema, Buddy agents attempt to contact the PA assigned to them. As mentioned before the PA will be located on another agent server, different from the location of the Buddy agent. On making contact, the Buddy agent will attempt to *sense* the PA. The *sense* procedure has been described in detail in [10] and hence is not reproduced here. Briefly recapitulating, the Buddy agent in *sensing* the PA makes contact with it and performs certain checks from the remote location to verify the presence and the integrity of the PA. If the Buddy agent is able to successfully contact the PA and carry out the prescribed checks, it sends a message back to the Home Base with that information. If the Buddy agent is unable to contact the PA, no message is sent back.

Using the reports received at the Home Base, the Confidence Factor (CF) report is generated which gives an indication of the threat perception level in the agent community [10]. The Confidence Factor (CF) function is given as

\[ CF = \frac{\sum x + \left(\sum y\right)\left(-1\right) - \left(R-r\right)}{T} \]

Where,
- \( T \) = Total number of agents within the community
- \( \sum x \) = Number of agents operating on Trusted Servers
- \( \sum y \) = Number of agents operating on Non-Trusted Servers
- \( R \) = Number of Buddy Reports expected per cycle
- \( r \) = Number of Buddy Reports received per cycle.

In the event that a PA has been killed, the Home Base will receive two messages less than expected and accordingly infer that the PA is missing in the field. The second
phase is similar to the first phase, except that agent roles are reversed. Buddy agents assume a PA role which implies that apart from sensing other agents, they will also be sensed. It is important to note that the agents to be sensed by the PA are different from those allocated to it in the first phase. On conclusion of the second phase, agents that were involved in the sensing operation, send out a report to the Home Base. The reports are used to generate the CF report as done in the first phase. The CF reporting function is an important part of the schema operation as it allows the Home Base to gauge the current threat perception faced by its agents in the field. Since the agent community is mobile and the location of the agents within the community is a dynamic aspect, it is very necessary to have a reliable tool to monitor the progress of the agents in the field. Since the CF is composed of continuously dynamic parameters, it is capable in reflecting the agent community spread. Another advantage of the CF is that it does not require many parameters for its computation and hence is not overly dependent on any aspect of the community operation. An algorithmic description of the schema has been given in [10].

By employing the Buddy model, the multi-agent community implements a dynamic security layer that alerts the Home Base for possible malicious activity within the community members. However, the schema imposes some stringent demands on the community for it to be effective. The major demand, as imposed by the schema is the requirement of fixed sized agent communities. In other words, at any time for the schema to be effective the community has to adopt a particular configuration, such as the 7-14 model described earlier in the section. Adopting this configuration implies that at any point of time, there should be 14 members within the agent community. While this case is a valid assumption for a fixed size community, it does not consider agents that may wish to terminate themselves on conclusion of their business functionality. In other words, agents that have completed their business functionality are compelled to prolong their existence until other agents of the community have completed their business functions.

To counter this drawback, the paper proposes an extension to the existing model by making a clear demarcation between the business function and the security protection function in the agent community operation. The paper proposes dividing the agent community into two groups. The first group includes all those agents that are involved in carrying out a business task for the agent community. This group shall be referred to the Worker Agent Group (WAG). The second group includes all those agents that implement a security layer for the agent community. This group shall be referred to as the Security Protection Group (SPG). Agents within the SPG do not implement any business function. The sole purpose of their existence within the agent community is to provide a security cover to the WAG. Figure 2 describes the relationship between these two groups within the community.

In figure 2, the agent community boundary is depicted by the dotted outer circle. Agents within the central inner circle that are represented by the dark stars, belong to the SPG. All other agents that do not belong to the SPG circle belong to the WAG. Agents within the WAG are distributed into various clusters. The number of agents within each cluster is dictated by the business functionality of the agent application. In the figure, these agent clusters are shown as circles. Each cluster holds four agents. Since the size of the WAG is variable and agents can leave as well as join the community. To keep the size of agents constant within a cluster there is a regular
distribution of the WAG agents. For example, if 3 agents from one cluster group and 1 agent from another cluster group leave the WAG, the two clusters are merged as one. On the other hand, the size and structure of the SPG remains fixed and is defined by the configuration of the Buddy model adopted. Thus, this extension of the Buddy model defines an additional security relationship.

The first security relationship within the SPG covers the agents within the group. The scope of this relationship is governed by the rules defined for the Buddy model schema operation [10]. The second security relationship in the model bridges the WAG and the SPG and extends the security cover generated by the SPG to the cluster agents of the WAG. Each agent in the SPG is assigned to 1 cluster group of the WAG. Further, each of the assigned SPG agents has a one to one relationship with each of the agents in cluster group. This implies that each SPG agent directly monitors an agent in the assigned WAG group.

Since the WAG is a dynamic entity, relationships between the WAG and the SPG are formed and are terminated. This factor is governed by the birth and death of the WAG agents within the agent community. Each agent within the SPG is assigned a set of WAG agents to monitor. The number of agents in this set is variable and is fixed by the Home Base. When any agent belonging to the WAG is terminated, its assigned SPG agent notifies the Home Base. The Home Base can then cross check its agent layout plan and decide if the agent termination was a planned occurrence or the result of a possible malicious attack. If the agent termination was an expected event, it marks the particular cluster for new agent allocation. When a new agent joins the community, the Home Base allocates it to the cluster that is one agent short. The SPG agent monitoring that particular agent cluster is accordingly informed. The next
section gives the schema implementation details and analyses the agent migration of the SPG agents using performance graphs.

**Buddy Model Schema Operation**

The Buddy model schema is implemented using Grasshopper 2.2.4b mobile agent systems, running on INTEL P4 machines with 512MB RAM, using Microsoft Windows XP Professional 2002 and Microsoft Windows 2000.500.2195. The Java used is Sun’s JDK version 14.2_02. Some of the terms used in this section are native to Grasshopper MAS terminology and their explanation can be found in [3].

**Implementation Details**

The Grasshopper agent system screenshot of figure 3 describes the SeeTheWorldAgency region which is the central region in the implemented model of the schema. The scenario described in this section is a news agency scenario. In this scenario, Agencies register themselves with a particular Region. MAs from different users and news vendors wishing to purchase or sell news arrive at these registered agencies and attempt to transact with the agent systems (referred to as Agencies) directly or with the agents that may be representatives of the agencies. The 7-14 configuration Buddy model deploys 14 agents over 4 different agencies. These agents are launched from the Home Base agency of the community. Apart from the agent identification number agents are identified by their names. In the described scenario agents are named as CommunityAgentOne, CommunityAgentTwo and so on unto to CommunityAgentFourteen. In this schema implementation the participating agents are referred to by their name rather than their identification number just to make it easier for the readers to trace the agent operation within the schema. It may be mentioned that the agent identification number is assigned by the agency that creates the agent and is not an optional parameter while the agent name is assigned a default value if the agent creator doesn’t assign a name.

The region registering the agencies is identified as SeeTheWorldAgency. Including the HomeBase-NewsAgency, there are five different agencies that are registered to the SeeTheWorldAgency region. They are:

- NorthWestNewsAgency
- TransWorldNewsAgency
- TomorrowNeverDiesNewsAgency
- GlobalNewsAgency

The deployed agents are seen docked to their respective agencies in figure 3. Prior to an attempt to sense, the Buddy agent has to search for the PA at the designated location. After locating the PA the Buddy agent would sense the agent. After the sense operation is successfully completed, the Buddy agent contacts the Home Base reporting a successful contact. This contact-sense phase is repeated after reversing the roles between the Buddy and the PA.

The Buddy agent’s reports received at the Home Base are used to prepare the Confidence Factor (CF) report. This function takes into account the operation of all
agents within the community. It uses previous performance and travel histories of agents to differentiate between trusted and non-trusted servers. For example, servers which have Service Level Agreements (SLAs) and other business agreements with the Home Base of the agent community are regarded as trusted servers while all other servers are regarded as non-trusted servers. The CF values can be positive as well as negative. A positive CF value indicates a relatively secure level of security for the entire agent community as compared to a negative CF value.

Since the agents are located on different physical machines, it makes it difficult for a malicious action to be carried out on the community agents. Further, since the Buddy agent does not have to move to the same physical location as the PA to sense it, there is no possibility of it being threatened. While some might argue that there is no way to verify the integrity of the results received by the Buddy agent, there remains the fact...
that the PA will have to perform the role of the Buddy agent in the next cycle, from a different location and so if the PA that was sensed has been compromised, it will be unable to participate in the next phase of the schema thereby alerting the sentinels in the Home Base.

Thus changing the location of the agents during the schema operation contributes to the efficiency of the security schema. The next section analyses, the migration factor of the MAs and examines if it has any negative repercussions on the schema operation.

Agent Migration Analysis

The experiments described in this section were undertaken with a view to analyse the effect of the agent migration on the performance of the Buddy model. They attempted to analyse if mobility of agents between phases led to a performance degradation of the Buddy model security schema. Analysis of the mobility context is important as mobility is a central factor in the operation of the Buddy schema. Mobility contributes to the effectiveness and the reliability of the schema since each community agent has an equal role to play in the schema operation. After the execution of its role, the MAs move to a different location and participate in the schema by playing another role. If an agent has been compromised by a malicious server and it is not allowed to migrate, there is an immediate vacuum in the system that is felt by the other agents of the community. In another scenario if the agent is compromised but is allowed to migrate to its next port of call, it will not be able to participate in the schema as it will need to undergo a role play and perform allocated functions. A compromised agent will be unable to perform this operation and again the system will detect a change and the other agent community members will trace the change to its source.

To collect data for the experiments, the agents were supplied with a function that calculated the elapsed time between the time when the Buddy agent starts searching for the assigned PA and the time when the agent is located. The calculated time (in milliseconds) are written into a file, along with the Buddy and the PA details by the Buddy agent. This file is located at the Home Base of the agent community. The data recorded in this file is then used as an input for the Confidence Factor function that indicates the existing level of security within the schema for that particular cycle of the Buddy schema operation.

The graphs of figure 6 describe the performance of the Buddy model under different conditions. The first two graphs in the figure represent the performance of the model when the Buddy agents do not change their locations after the first phase. In other words they carry out both phases of the schema cycle from the same location. The second set of graphs shown as 3 and 4 in the figure, describe the performance of the schema when the Buddy agents change their location after the completion of the first phase. In both the cases, each set of experiments was carried out thrice and are represented by a series each. The first two graphs in the figure show the state in which the Buddy agents did not migrate after contacting the PAs, while in the graphs 3 and 4 they changed their Place location after the completion of the phase.

The graphs describe a relatively uniform performance of the schema before and after the implementation of the changed location criteria.
Figure 6. Performance Runs of the Buddy Model
A few variations are observed but since most of them occur during the performance run indicated by series 2, it can be attributed to non-availability of CPU cycles while that particular run was underway. Thus from the agent execution runs described by the graphs in figure 6, it can be inferred that the migration of the Buddy agents has no adverse effect on the performance of the schema. Further this approach of methodically changing the location of the Buddy and PA after each scan would be beneficial in maintaining the integrity of the community agents. The next section enumerates the advantages of the Buddy model by comparing it to other existing agent security schemas.

Related Work

Cryptographic Security Proposals

As summarised in [9] security techniques that aimed at protecting the agent from a malicious server used a preventive approach depending on cryptography. This section describes two of the earliest proposals that used such an approach. The first one proposed computing with encrypted functions [17]. This proposal intended to protect the agent from a malicious agent server. The proposal relied on the agent being able to perform computations with encrypted functions at the agent server. These encrypted functions contained an embedded key that allowed the agent to sign the results computed and shift them back to the Home Base, where the results could be decrypted and analysed. Closely following this schema was another schema with a cryptographic base. This schema proposed the use of Obfuscating algorithms and a Time based Black Box approach [16] to protect the agent code. This schema relied on an obfuscating function that would make the agent code indecipherable and discourage the agent server from inspecting the code.

While these two techniques appeared to be strong proposals in securing the agent code from possible tamper attempts, like all other solutions they are dependent on the host platform for execution. This limitation is the biggest contradiction in its success. In the first case, a malicious agent server might reverse engineer the agent code and prevent the security function from triggering at run time. Further the agent server can execute the security mechanism several times in a controlled environment and learn how to invalidate the agent security at run-time in a live environment. Secondly both security proposals rely heavily on an efficient tamper-resistant cryptographic schema to be successful. Such a cryptographic schema would have to be robust enough to prevent hacking by reversible engineering. Thirdly, a cryptographic approach appears to be an expensive proposition for an agent that may require only a few milliseconds to execute its business function. Burdening an agent with a security function that doubles or triples its execution turnaround time and increases dependency on the agent server for resources is a non-viable option and that is where the Buddy model presents a reliable and cost-effective technique for the security of MA communities.

Non-Cryptographic Security Proposals
A non-cryptographic approach to agent security usually focuses on detection rather than prevention. While this may appear to be a relatively weaker approach as compared to the previous, it is definitely a cost effective technique as well as a reliable technique of agent security.

Execution tracing [18] was a proposal that required the host agent server to maintain a log of the visiting agent’s execution trace summary, sign the trace with its own signature and using a secure channel transmit it back to the Home Base. While the proposal cannot be called a pure non-cryptographic approach, the core of the proposal did not require any cryptography for its execution. The obvious disadvantage with this solution was its overt dependence on the agent server for its execution and subsequent completion. A malicious agent server will never participate in sending an information log that exposes itself to the Home Base. Secondly, there is no guarantee that the execution log being sent by the host agent server is not a tampered version reflecting the agent’s expected performance. These difficulties make the reliability of the proposal somewhat dubious.

Mutual itinerary recording of agents [12] advocated the use of cooperating agents that recorded the visits of each other. However, this proposal was not robust enough to detect the perpetrator of a malicious action in the event that an agent was terminated. Further, it required two agent servers acting in conjunction could render the schema ineffective. Thirdly, for each agent that was dispatched there was a need for a peer to accompany and to follow its progress and vice versa. This approach contributed to the overheads involved in executing the schema.

The Buddy schema builds over all these documented drawbacks in proposing a solution that involves only the members of the agent community with no overheads involved. The next section documents the advantages Buddy model schema.

**Advantages of the Buddy Model Schema**

1. The schema does not require the host server’s participation in preparing any execution logs or sending any information back to the Home Base. This makes the schema, a comparatively self-reliant solution.
2. The schema involves every member of the agent community in the security function. This approach distributes the workload as well as the risk of the schema being compromised by a hostile action.
3. The schema does not require a cryptographic base making the operation of the schema cost effective.
4. The schema promotes transparency in its operation. This helps in building trust between the visiting agent and the host agent server.
5. The schema presents a reliable approach in enforcing security in agent communities.
6. While the mobility factor in most agent application scenarios is seen to be the cause of increased vulnerability. The schema uses mobility to its advantage by executing the security function from changing locations. This allows the quick detection of a hostile action carried out by a malicious server.
7. The mobility factor of the community agents also makes it possible for the agents to detect the perpetrator of a hostile action while operating in the field.
detection was not possible in other schemas until the MAs returned to their Home Base and were analysed.

8. The CF report enables the community agents to gauge the security level of the community operation in tangible terms. This is a very useful and important feature in a multi-agent based community operation.

9. The creation of a WAG and SPG groups extends the operation of the Buddy schema to variable sized communities. It allows the creation of variable sized groups within the agent community. The existence of these groups is dictated by business functionality requirements only.

Conclusion and Future Work

Security in mobile agent communities is a critical aspect for the success of the agent operation. Unfortunately maintaining a secure environment in a dynamic environment such as those created by MAs requires countering certain limitations and loopholes. The main limitation in proposing a security schema that can protect MAs from malicious host servers is the complete control over the execution environment that the servers hold. This enables malicious servers from inspecting the MA code prior to execution and introducing systems that by-pass the security functions of the MAs.

The Buddy model is a security schema that acts at the agent community level and uses the strength in numbers to counter malicious agent servers. This paper described an extension of the Buddy model schema of security for multi-agent mobile communities. The Buddy model schema is a novel approach to the agent security problem. It focuses on detecting malicious actions against MAs perpetuated by hostile agent servers. The transparency of the schema makes the hosting agent server aware of the security cover. This feature acts as a deterrent and prevents the agent server from launching a malicious attack against the MA.

The paper described an extension of the schema that gives the model flexibility to deal with variable sized communities. It proposes the separation of business functionality from the security function within the agent operation. This approach allows two layers to be formed within the agent community, the Worker Agent Group (WAG) and the Security Protection Group (SPG). The operation objective of both these groups is independent of each other. However, these groups co-exist in terms of a one-to-many relationship that is exerted by the SPG agents on the WAG agents. The new approach makes the Buddy model schema operation relatively flexible and enables broader security coverage within the agent community.

Further, the paper argued that the Buddy model schema was not tied down by the migratory nature of the community agents. To prove this, a set of experiments were carried out on the agent community. In the first set of experiments the location parameter remained unchanged in both phases, while in the second set of experiments; the location of the agents was changed between the two phases. The performance results obtained from the two set of experiments demonstrated a stable and a relatively unchanged performance of the schema operation when migration of agents was introduced. Thus, the mobility factor in the Buddy model schema does not affect the operation of the MAs. On the contrary as described in the
paper, mobility allows the detection of a malicious action on a MA while they are still operating in the field. The paper also compared the Buddy model schema against some existing security proposals and enumerated its advantages. Future work will focus on extracting other context parameters that influence the operation of the Buddy model between the two groups and studying their effects on the schema.

**References**


Fault-Tolerant Information Sharing Networks

Jisun Park, K. Suzanne Barber
The Laboratory for Intelligent Processes and Systems
Electrical and Computer Engineering Department
The University of Texas at Austin
{jisun, barber}@lips.utexas.edu

Abstract. Agents are goal-driven entities, and the goal achievement is affected by the robustness of information supply required to achieve the goal. This research addresses fault tolerance in information sharing networks, and shows that the construction of information sharing networks, in the way that an agent can tolerate \( k \) concurrent information source failures, can be expressed as a constraint satisfaction problem. By solving the resulting constraint satisfaction problem, an agent can efficiently decide on the information sources from which to request the required information, so that the agent can achieve the goals even if a subset of the information sources fail.

1. Introduction

Agents are goal-driven entities. When the goals are dependent on information, the goal achievability is dependent on how robust the information acquisition is. From the traditional AI planning perspective (e.g., [3],[7],[8],[11],[10]), a goal of an agent can be divided into a set of sub-goals. Each sub-goal of the agent may require a set of information. When an agent fails to acquire the required information for any sub-goal, the sub-goal cannot be satisfied thus the goal achievement fails. Therefore, it is essential for an agent in Multi-Agent Systems (MAS) to find the appropriate information sources which satisfy the information requirements (which is the collection of required information elements for all sub-goals of an agent) [1].

Park and Barber [5] proposed Goal Coverage as one of the metrics an agent must maximize to increase the goal achievability. An agent’s Goal Coverage represents the portion of the agent’s information requirements covered by a set of selected information sources (i.e., the portion of goals covered by a set of selected information sources). While it is desirable that Goal Coverage is complete, how individual information sources contribute to the complete goal coverage has been left as a design decision. For example, an agent can prefer requesting required information from a maximum number of information sources so that an agent’s dependence on each information source is minimized. This research is motivated by the observation decisions regarding how an agent decides the distribution of information sources’ contribution (i.e., the distribution of dependence on information sources) is closely related to the fault tolerance of the information sharing networks in which agents exchange information by requesting and providing the information. Even if an agent can find the appropriate information sources, unexpected faults of the information
sources can cause the information supply interruption, which eventually cause the failure of goal achievement. Therefore, guaranteeing the information supply in the presence of information source faults is significant. The unexpected crash of information sources which can cause the information supply interruption is of particular interest in this research. The overall objective of this research is to model the fault-tolerance problem in information sharing networks as a constraint satisfaction problem (CSP) so that the agent can use existing techniques for efficiently solving the posed CSP. The CSP modeling of information source fault tolerance can also be extended to a constraint optimization problem (COP) by considering the message cost of guaranteeing the fault-tolerance.

Section 2 describes the fault tolerance in information sharing networks in the context of $k$-fault tolerance. Section 3 introduces an information dependence matrix which is used in expressing the fault tolerant information sharing networks as a constraint satisfaction problem in Section 4. Section 5 shows the conversion of the fault tolerance problem into a constraint optimization problem by considering the message cost. Section 6 provides some examples and applicable solutions for the examples, and section 7 summarizes this paper.

2. $k$-Fault Tolerant Information Sharing Networks

Information sharing networks (ISN) describe a system where an agent can act as an information source, an information consumer, or both at the same time. Each agent has goal(s), and the goals require a set of information. This research assumes agents must rely on others for some or all of its information requirements. In order to satisfy the information requirements, an agent needs to find a set of information sources (agents) which can provide the required information. When there are multiple information sources for a single information requirement, an agent can request the same information elements from multiple information sources, meaning that redundant information sources are allowed. Aggregating information from multiple sources into a single conclusion (about an information requirement) can be domain-dependent or agent-dependent. However, it is always required to procure at least a single information source per an information requirement to satisfy the information requirements.

A $k$-fault tolerant system is defined as a system which can mask $k$ concurrent component failures. In traditional distributed systems research, redundancy is one of the most prevalent approaches for realizing $k$-fault tolerance, and it is known that $k+1$ components can survive the $k$ component failures [9]. In a similar context, $k$-fault tolerant information sharing networks are defined as the information sharing networks where an agent can satisfy the information requirements despite $k$ information source failures. Therefore, procuring multiple information sources is not just allowed but encouraged for fault-tolerance. Fig. 2 illustrates a simple example of this idea, and Fig. 1 summarizes the notation used in this paper.
Assuming $a_0$ needs three information elements $\{r_1, r_2, r_3\}$ ($R(a_0) = \{r_1, r_2, r_3\}$), Fig. 2 (a) depicts the potential information sources and corresponding information that each respective information source can provide from $a_0$'s perspective ($a_1$, $a_2$, $a_3$ each can provide $\{r_1, r_2, r_3\}$). In this situation, if $a_0$ requests information as in Fig. 2 (b) (i.e., requesting $\{r_1, r_2\}$ from $a_1$, $\{r_2\}$ from $a_2$, and $\{r_j\}$ from $a_3$), $a_0$ cannot satisfy its information requirements when $a_1$ or $a_3$ fails to provide the requested information (i.e., $r_j$ cannot be provided when $a_1$ fails and $r_j$ cannot be provided when $a_3$ fails). On the other hand, in Fig. 2 (c) (i.e., $a_0$ requesting $\{r_1, r_2\}$ from $a_1$, $\{r_2, r_3\}$ from $a_2$, and $\{r_i, r_j\}$ from $a_3$) $a_0$ constructs a 1-fault tolerant information sharing network with the information sources, meaning $a_0$'s information requirements can be satisfied even when a single information source failure occurs. For example, even if $a_j$ fails to provide requested information $\{r_i, r_j\}$, $a_0$ can still be provided $\{r_i, r_j, r_k\}$ from $a_j$ and $a_3$.

To show the practical implication of the proposed approach, the following scenario can be mapped into Fig. 2.

The goal of $a_0$ is to track the location of a moving object in a three-dimensional Cartesian coordinate $(x, y, z)$, and each coordinate value can be estimated from the observed information $(r_1, r_2, r_3)$ about the object (i.e., $x=f(x(r_1), y=f(y(r_2), z=f(z(r_3)))$.

Three information sources ($a_1, a_2, a_3$) can provide the information about the location, and $a_0$ needs to decide from which information source to request the information to achieve its goal.

Assuming $a_0$ needs three information elements $\{r_1, r_2, r_3\}$ ($R(a_0) = \{r_1, r_2, r_3\}$).}

### Notation

- $A = \{a_0, ..., a_{n-1}\}$: agents in a system. ($|A| = N$)
- $R(a_j) = \{r_1, ..., r_{n_j}\}$: agent $a_j$'s information requirement ($|R(a_j)| = M$)
- $\text{PROV}(a_j)$: a set of information provided by $a_j$
- $S(a_j)$: agent $a_j$’s potential information sources
- $\text{req}(R_{\text{req}}(a_j), a_j)$: agent $a_j$'s request of information from $a_j$

where $(R_{\text{req}}(a_j) \subseteq R(a_j)) \land (R_{\text{req}}(a_j) \subseteq \text{PROV}(a_j))$

### Fig. 1. Notation

### Fig. 2. Information Sources and 1-Fault Tolerant Information Sharing Network: (a) Information Sources and a Consumer (b) Non Fault Tolerant ISN (c) 1-Fault Tolerant ISN
Therefore, in order for an agent to construct a $k$-fault tolerant information sharing networks, the agent needs to secure at least $k+1$ information sources per each information requirement. The difficulty is that securing $k+1$ information sources per each information requirement necessitates a combinatorial search where solution time grows exponentially. In addition, since the information sources often do not provide the same set of information (as opposed to Fig. 2 (a)) and the number of information types provided by each information source is different, the search becomes more complicated with those constraints.

Without considering the message cost, the easiest way to realize the $k$-fault tolerance is to request all the required information from all the potential sources. For example, in Fig. 2 (a), $a_0$ can simply request \{r_1, r_2, r_3\} from the respective information sources $a_1$, $a_2$, $a_3$. However, since the information request is not a one-time event, message cost can affect the agent performance over a period of time. This research addresses $k$-fault tolerant information sharing network either considering the message cost or without considering the message cost.

$k$-fault tolerant information sharing networks can be mapped into a constraint satisfaction problem (CSP), and the advantage of converting the problem into a CSP is that an agent can then use existing CSP solvers.

In the following section, the Information Dependence Matrix (IDM) is defined to visually describe the relationship between the information requirements and the information sources. IDM also aids to establish a proper set of constraints on the information requirement and information sources to realize the $k$-fault tolerant information sharing networks.

### 3. Information Dependence Matrix

The Information Dependence Matrix (IDM) depicts which information requirement is dependent on which information sources. The row of the matrix represents the information requirements ($r_i$) and the column represents the information sources ($a_j$). An element in the matrix ($d_{ij}$) denotes the dependence of an information requirement ($r_i$) on the corresponding information source ($a_j$). The elements are binary values ($\{0, 1\}$), where 1 means the existence of dependence and 0 means no dependence. Dependence between an information requirement and an information source exists when the information requirement is requested from the corresponding information source. Fig. 3(a) shows the $3 \times 3$ IDM, and (b), (c) show the IDM for Fig. 2 (b) and (c).

\[
\begin{array}{ccc}
   & a_1 & a_2 & a_3 \\
 r_1 & d_{11} & d_{12} & d_{13} \\
 r_2 & d_{21} & d_{22} & d_{23} \\
 r_3 & d_{31} & d_{32} & d_{33} \\
\end{array}
\]

(a) $3 \times 3$ Information Dependence Matrix
By representing the dependence of information requirements on the information sources, \( k \)-fault tolerance can be diagnosed simply by summing up each row and checking if the value is equal to or greater than \( k+1 \). If there exists any row which sums up to less than \( k+1 \), then the information sharing network is not \( k \)-fault tolerant. In Fig. 3 (b), the first row and the third row sum up to 1 respectively, so the agent cannot mask the faults in \( a_1 \) or \( a_3 \). On the other hand, in Fig. 3 (c), all rows sum up to 2, so the constructed information sharing network is 1-fault tolerant.

In the next section, the formal representation of the constraints using the IDM will be presented.

4. \( k \)-Fault Tolerance as a Constraint Satisfaction Problem

A constraint satisfaction problem (CSP) is defined by a tuple \(<X, \mathcal{C}>\), where \( X \) is a set of variables \( \{x_1, x_2, \ldots, x_n\} \), and \( \mathcal{C} \) is a set of constraints \( \{c_1, c_2, \ldots, c_m\} \). Each variable \( x_i \) has a nonempty domain \( D_i \) which contains possible values of the variable, and each constraint \( c_j \) involves a subset of \( X \) specifying the allowable conditions of the variables. The solution to a CSP is a complete assignment of values to all variables which do not violate the constraints [6].

\( k \)-fault tolerant information sharing networks can be expressed in a Boolean CSP, where the variables are dependences \( (d_{ij}) \) and the domain of the variables consists of binary values \( \{0, 1\} \). The constraints can be built from (1) \( k \)-fault tolerance conditions (i.e., at least \( k+1 \) information sources per information requirement), (2) the
set of information each information source can provides. The first set of constraints
from the $k$-fault tolerance conditions can be stated as Eq. (1).

$$\forall i, \sum_j d_{ij} \geq k + 1$$  \hspace{1cm} (1)$$

Therefore, Eq. (1) can be applied to Fig. 3 (a) in the case of 3×3 IDM:

$$d_{11} + d_{12} + d_{13} \geq k + 1
\quad d_{21} + d_{22} + d_{23} \geq k + 1
\quad d_{31} + d_{32} + d_{33} \geq k + 1$$

Other sets of constraints come from each information source’s capability (i.e., the
set of information each information source can provide). In the case of Fig. 2 (a)
where each information source can provide all the required information, there are no
additional constraints. However, if each information source can provide only a subset
of the information requirements, the information elements each information source
can provide need to be described as constraints. For example, assuming that $a_3$ can
provide \{r_1, r_2\} instead of \{r_1, r_2, r_3\}, then the additional constraint $d_{33} = 0$ is included.

Satisfying the constraints, by assigning appropriate values to the variables,
guarantees $k$-fault tolerance of an information sharing network. However, requesting
ting all the information that can be provided by all the potential information sources can
achieve the maximum possible fault tolerance for information source crash failures.
As an example, when each information source ($a_1, a_2, a_3$) can provide the information
elements \{r_1, r_2, r_3\} (as in Fig. 2 (a)), requesting \{r_1, r_2, r_3\} from each information
source guarantees up to a 2-fault tolerance (i.e., each information requirement is
provided by three information sources). The problem is that such a naïve approach
requires more message exchanges increasing the message complexity of the system.
Moreover, more computation is required for aggregating the received information into
a single conclusion. In order to overcome those problems, another set of constraint
can be added so that $k$-fault tolerance can be achieved with minimum message cost.

5. $k$-Fault Tolerance as a Constraint Optimization Problem

In a constraint optimization problem (COP), a subset of the constraints is represented
as an objective function to be maximized (or minimized). Considering the $k$-fault
tolerant information sharing networks, the number of messages (message cost) can be
represented as an objective function to be minimized. In an information sharing
network, information is provided once an agent sends a request for the information.
Therefore, 2 messages are required per a single instance of information.

An element in an IDM (i.e., dependence $d_{ij}$) can represent the number of message
exchanges along with the existence of dependence without changing the domain of $d_{ij}$.
The domain of dependence consists of binary values (\{0, 1\}). 1 represents the
existence of dependence and 1 round of message exchange thus 2 messages (request-
reply). 0 represents no dependence thus no message. Having the implication of dependence extended, the abstracted message cost in an information sharing network is simply the sum of all dependence values, and the message cost is an objective function to be minimized. The message cost can be represented as follows (Eq. (2)).

\[
\text{message cost} = \sum_i \sum_j d_{ij}
\]

(2)

Therefore, the additional constraint is described as follows (Eq. (3)):

\[
\text{minimize} (\text{message cost}) = \text{minimize}(\sum_i \sum_j d_{ij})
\]

(3)

In summary, the \( k \)-fault tolerant information sharing networks with minimum message cost can be expressed as a constraint optimization problem, where the constraints are Eq. (1), Eq. (3), and the constraints on the information sources’ capability. As an example, considering Fig. 2 (a) and Fig. 3 (a) where \( a_0 \) requires \{\( r_1 \), \( r_2 \), \( r_3 \)\}, and \( a_1, a_2, a_3 \) are the potential information sources, building a \( k \)-fault tolerant information sharing network is to solve the following COP:

\[
\begin{align*}
d_{11} + d_{12} + d_{13} &\geq k+1 \\
d_{21} + d_{22} + d_{23} &\geq k+1 \\
d_{31} + d_{32} + d_{33} &\geq k+1 \\
\text{minimize}(d_{11} + d_{12} + d_{13} + d_{21} + d_{22} + d_{23} + d_{31} + d_{32} + d_{33})
\end{align*}
\]

In this example, this COP can be converted to a CSP. In order to minimize the message cost (last constraint), the sum of dependence values for a single information requirement must be equal to \( k+1 \). So the constraints are reduced to be as follows:

\[
\begin{align*}
d_{11} + d_{12} + d_{13} &= k+1 \\
d_{21} + d_{22} + d_{23} &= k+1 \\
d_{31} + d_{32} + d_{33} &= k+1
\end{align*}
\]

The reduction of constraints thus the reduction of the problem to a CSP is useful because a COP usually involves a computationally expensive search procedure. In the following section, the discussion about solving a CSP problem in \( k \)-fault tolerant information sharing networks will be presented.

6. Solving CSP: A Case Study

Any search algorithm can solve a CSP because the CSP is basically finding an assignment of values to the variables meeting the constraints. However, backtracking search [2], which is a depth first search in essence, is a popular method since it is
more efficient, for CSP, than breadth first search methods. Backtracking is to choose a value for one variable at a time and backtrack when a node cannot be assigned a value due to the constraint violation. The performance of backtracking search is dependent on various heuristics (e.g., the order of variable assignment). Although there are some applicable heuristics (e.g., minimum remaining values, least constraining value, etc), those heuristics do not always enhance the performance of the search.

Constraint propagation often helps to reduce the search space. Constraint propagation is propagating the constraints on some variables to other variables, so that the search space can be reduced before or in the early stages of the search.

Arc-consistency [4] provides a way of propagating the constraints. A constraint satisfaction problem can be depicted as a constraint graph. Each node \( V_i \) in the graph represents a variable \( x_i \) in the CSP, and an edge (called an “arc” in a constraint graph) is drawn between the nodes which are related by any constraints. A unary constraint (e.g., \( d_{ij} = 0 \)) is depicted as an arc originating and terminating at the same node. An arc \( (V_i, V_j) \) between \( V_i \) and \( V_j \) is arc-consistent if for every value of \( x_i \) in \( D_i \) there exists some value of \( x_j \) in \( D_j \) which is consistent with \( x_i \). Arc-consistency is directional, meaning that the arc \((V_i, V_j)\) can be arc-consistent when \((V_j, V_i)\) is not. By making each arc arc-consistent, inconsistent values can be eliminated from the domain of a variable, and the search space size can be reduced.

In the next subsections, example scenarios in \( k \)-fault tolerant information sharing networks are illustrated to show how the search and constraint propagation are performed.

6.1 Example Setup

The following setup is used for the illustration of search and constraint propagation.

\[
\begin{align*}
A = \{a_0, a_1, a_2, a_3\} & : a_0, a_1, a_2, a_3 \text{ exist in a system} \\
R(a_0) = \{r_1, r_2, r_3\} & : a_0 \text{ requires } \{r_1, r_2, r_3\} \\
S(a_0) = \{a_1, a_2, a_3\} & : a_1, a_2, a_3 \text{ are potential information sources of } a_0 \\
PROV(a_0) = \{r_1, r_2\} & : a_1 \text{ can provide } r_1, r_2 \\
PROV(a_1) = \{r_3\} & : a_2 \text{ can provide } r_3 \\
PROV(a_2) = \{r_1, r_2, r_3\} & : a_3 \text{ can provide } r_1, r_2, r_3
\end{align*}
\]

There are four agents and \( a_0 \) requires \( \{r_1, r_2, r_3\} \). \( a_1, a_2, a_3 \) can provide \( \{r_1, r_2\}, \{r_2, r_3\}, \{r_1, r_2, r_3\} \) respectively to \( a_0 \) (Fig. 4).

![Fig. 4. An Example Setup](image-url)
In this setup, the objective of \( a_0 \) is to construct a 1-fault tolerant information sharing network with minimum message cost. A 3×3 Information Dependence Matrix can be built from this setup and the 1-fault tolerant conditions can be described as follows:

\[
\begin{align*}
&d_{11} + d_{12} + d_{13} = 2 \\
&d_{21} + d_{22} + d_{23} = 2 \\
&d_{31} + d_{32} + d_{33} = 2
\end{align*}
\]

Also, two unary constraints exist from the set of information each information source can provides as follows:

\[
\begin{align*}
&d_{12} = 0 \\
&d_{31} = 0
\end{align*}
\]

The first scenario described in section 6.2 uses the original setup without any additional constraints, while the second scenario in section 6.3 adds more constraints to simulate a more complicated environment.

### 6.2 Applying a Backtracking Algorithm for Solving a CSP

The setup introduced in the previous section can be converted into a CSP as follows:

\[
\begin{align*}
&d_{11} + d_{13} = 2 \\
&d_{21} + d_{22} + d_{23} = 2 \\
&d_{12} + d_{31} = 2
\end{align*}
\]

This set of constraints is a reduced version of the original constraints presented in section 6.1 by using the unary constraints on \( d_{12} \) and \( d_{31} \) \((d_{12}=0, d_{31}=0)\). The constraint graph is provided in Fig. 5. Since \( d_{11} \) and \( d_{13} \) are related in the first constraint, an arc exists between \( d_{11} \) and \( d_{13} \). The second constraint results in the arcs among \( d_{21}, d_{22}, \) and \( d_{23} \). The arc between \( d_{12} \) and \( d_{31} \) is from the third constraint. It can be seen that this problem is divided into three separate sub-graphs (labeled as \( c_1, c_2, \) and \( c_3 \)). It is trivial to solve the graphs \( c_1 \) and \( c_3 \) although they are not arc-consistent. Considering the graph \( c_2 \) where all arcs are arc-consistent, backtracking can be applied as in Fig. 6. It can be observed that the variable and value ordering affect the search performance of backtracking (i.e., if search starts from the right-hand side of the tree, the first leaf node can be taken as a solution). The resulting solution from this example is \(<d_{11}=1, d_{12}=0, d_{13}=1, d_{21}=0, d_{22}=1, d_{23}=1, d_{31}=0, d_{32}=1, d_{33}=1>\)
6.3 Constraint Propagation by Arc-Consistency

Additional constraints can increase the complexity of search. The preprocessing on the constraints by constraint propagation can help to reduce the search space. To illustrate the constraint propagation by an example, this section adds one more constraint that $a_i$ can provide either $r_j$ or $r_k$ exclusively. The additional constraint can be expressed as $d_{ij}+d_{ik}=1$. The constraint graph with this additional constraint is in Fig. 7 (a). In this figure, the domain for each variable is also annotated on the right-side of each node.
The graph is divided into two independent sub-graphs (labeled $c_4$, $c_5$). In order to reduce the search space by eliminating the inconsistent values in the domain, arc-consistency is checked for each arc. In graph $c_4$, the arc $(d_{13}, d_{11})$ is inconsistent because no value of $d_{11}$ can satisfy the constraint $d_{11} + d_{13} = 2$ when $d_{13} = 0$. Therefore, $\{0\}$ can be eliminated from the domain of $d_{13}$. In addition, the arc $(d_{11}, d_{13})$ is inconsistent due to the reduced domain of $d_{13}$, so the domain of $d_{11}$ is reduced to $\{1\}$. The arc $(d_{11}, d_{21})$ is arc-consistent, so no domain reduction is available. The arcs $(d_{21}, d_{32})$, $(d_{21}, d_{33})$, and $(d_{22}, d_{32})$ are also all arc-consistent. In the graph $c_5$, the arcs $(d_{32}, d_{33})$ and $(d_{32}, d_{33})$ are not consistent. In order to make those arcs arc-consistent, the domains of $d_{32}$ and $d_{33}$ are reduced from $\{0, 1\}$ to $\{1\}$. The resulting graph with reduced domains is depicted in Fig. 7 (b). The constraint propagation leverages the search efficiency by reducing search space size, and there are numerous algorithms which make the arcs in a constraint graph arc-consistent (e.g., [4]). With this idea of arc-consistency, the search becomes more efficient and the arc-consistency itself can even generate the solution in some cases without additional search. The numerous
existing CSP solvers also adopt a similar approach. One of the resulting solutions from this example can be \( <d_{11}=1, d_{12}=0, d_{13}=1, d_{21}=0, d_{22}=1, d_{23}=1, d_{31}=0, d_{32}=1, d_{33}=1> \).

7. Conclusion

Agents are goal-driven entities. When the goals are dependent on information, the goal achievability is dependent on how robust the information acquisition is. When an agent’s goals require a set of information, the agent must acquire the information from others if it does not own the information. If any information supply interruption occurs, the agent cannot achieve its goals. However, unexpected faults of information sources can occur thus can prevent an agent’s goal achievement. This research addresses the issue of fault tolerance when agents share their information with others. Of particular interest, the research investigates the crash failure of information sources or agents. In order for an agent to tolerate \( k \) concurrent failures of information sources, the agent needs to have at least \( k+1 \) information sources per each required information. The system which can tolerate \( k \) concurrent components failures is called \( k \)-fault tolerant. \( k \)-fault tolerant information sharing networks can mask \( k \) concurrent failures of information sources, meaning that an agent is guaranteed to be provided all the required information despite \( k \) information source failures. This research shows that constructing \( k \)-fault tolerant information sharing networks can be expressed as a constraint satisfaction problem (CSP). By expressing the problem as a CSP, an agent can make use of the existing CSP solvers. \( k \)-fault tolerant information sharing networks in terms of a CSP offers agents’ efficient decision-making method to determine the distribution of dependence upon information sources, so that the agent’s goal achievement is guaranteed in the presence of information sources failure.

8. Acknowledgment

This research is sponsored in part by the Defense Advanced Research Project Agency (DARPA) Taskable Agent Software Kit (TASK) program, F30602-00-2-0588. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Project Agency.
References

Safety in Multiagent Systems by Policy Randomization

Praveen Paruchuri¹, Don M Dini², Milind Tambe³, Fernando Ordonez⁴, and Sarit Kraus⁵

¹ University of Southern California
Los Angeles, CA 90089
paruchur@usc.edu

² University of Southern California
Los Angeles, CA 90089
ddini@usc.edu

³ University of Southern California
Los Angeles, CA 90089
tambe@usc.edu

⁴ University of Southern California
Los Angeles, CA 90089
fordon@usc.edu

⁵ Bar-Ilan University
Ramat-Gan 52900, Israel sarit@macs.biu.ac.il

Abstract. In adversarial settings, action randomization can effectively deteriorate an opponent’s capability to predict and counteract an agent’s or an agent team’s policies. Unfortunately, little attention has been paid to intentional randomization of agents’ policies in (PO)MDPs (without sacrificing rewards). Indeed, the potential for miscoordination exacerbates the difficulty of policy randomization in teams based on decentralized POMDPs. This paper provides two key contributions to remedy this situation. First, it provides novel algorithms, one based on a non-linear program and one on a linear program (LP), to randomize single-agent policies, while attaining a certain level of reward. Second, it provides RDR, a new algorithm that efficiently generates randomized policies for decentralized POMDPs via the single-agent LP method.

1 Introduction

When agents and agent teams based on single-agent or decentralized (PO)MDPs face intelligent adversaries, policy randomization can effectively reduce an opponent’s capability to predict and exploit an individual or team’s action predictability. For example, when individual or teams of UAVs (Unmanned Aerial Vehicles) or robot-sentries monitor sensitive areas [1], randomized policies can degrade opponent’s ability to predict the UAV or robot actions.

This research on policy randomization is motivated by domains where we are unable to explicitly model our adversary’s actions or capabilities or its payoffs, but the adversary observes our agents actions and exploits any predictability in such actions in some unknown fashion. For instance, consider agents scheduling security crew rotation, maintenance, refueling etc at nerve centers such as ports and airports. Adversaries may
be unobserved terrorists with unknown capabilities and actions. Yet these adversaries easily observe all scheduled actions and may exploit schedule predictability to cause tremendous unanticipated harm. Alternatively, consider a team of UAVs surveilling a region wracked with a humanitarian crisis. Adversaries may be humans intent on causing some significant unanticipated harm — e.g., disrupting food convoys or harming refugees or shooting down the UAVs — the adversaries capabilities, actions or payoffs are unknown and difficult to model explicitly. However, the adversaries can observe the UAVs and exploit any predictability in UAV surveillance routes, e.g. engage in unknown harmful actions by exactly avoiding the UAVs route.

In other words, the adversary observes or estimates the agent’s current state and if the agent’s actions are predictable (based on past observations and learning) it exploits the prediction. In fully observable domains, the opponent estimates the agent’s state to be the current world state which both can observe fully. Even if the domain is partially observable, we assume that the adversary estimates the agent’s belief state, because: (i) the adversary eavesdrops or spies on the agent’s sensors such as sonars or radars (e.g., UAV/robot domains); (ii) or the adversary estimates the most likely observations based on its model of the agent’s sensors; (iii) or the adversary is co-located and equipped with similar sensors. Thus, we make a worst-case assumption about the adversary’s estimation abilities, but this is appropriate given the potential for tremendous harm. Thus, our work maximizes policy randomization to thwart the opponent’s prediction of the agent’s actions based on an agent’s state.

Unfortunately, while randomized policies are created as a side effect [2] and turn out to be optimal in some stochastic games [3], little attention has been paid to intentionally maximizing randomization of agents’ policies even for single-agents. Obviously, simply randomizing an MDP/POMDP policy can degrade an agent’s rewards, and thus we face a multi-criteria problem: how to randomize policies while ensuring that agents do not sacrifice their rewards. Casting the problem as a multi-criteria one enables explicit understanding of the trade offs in rewards vs. action unpredictability. However, generating policies under multi-criteria objectives is difficult, a difficulty that is exacerbated in agent teams based on decentralized POMDPs, as randomization may create miscoordination.

This paper provides two key contributions to remedy this situation. First, we provide novel techniques that enable policy randomization in single agents, while attaining a certain reward threshold. While randomization is measured via an entropy-based metric, our techniques are not dependent on that metric. In particular, we illustrate that simply maximizing entropy-based metrics introduces a non-linear program that does not guarantee polynomial run-time. Hence, we introduce our BRLP (Binary search for Randomization) linear programming technique that randomizes policies more efficiently. The second part of the paper provides a new algorithm, RDR (Rolling Down Randomization), for generating randomized policies for decentralized POMDPs. RDR starts from a locally optimal policy for decentralized POMDPs, then iterates, randomizing policies for agents turn-by-turn, keeping policies of all other agents fixed. One key insight in RDR is that given fixed randomized policies for other agents, the problem of generating a randomized policy can be solved via BRLP, i.e., our single agent method.
2 Randomizing Single Agent Policies

Before considering agent teams, we focus on randomization of single agent MDP policies. For example, suppose a single MDP-based UAV agent is surveilling a troubled region. The UAV gets rewards for surveying different areas of the region, but as discussed above, it must randomize its surveilling strategies to disrupt adversary’s plans.

An MDP is denoted as a tuple: \((S, A, P, R)\) where \(S\) is a set of world states \(\{s_1, \ldots, s_m\}\), \(A\) the set of actions \(\{a_1, \ldots, a_k\}\), \(p(s, a, j)\) the transition function, and \(r(s, a)\) the immediate reward. If \(x(s, a)\) represents the number of times the MDP visits state \(s\) and takes action \(a\) and \(\alpha_j\) represent the initial probability distribution over states \(j \in S\), then the optimal policy, maximizing expected reward, is derived via the following linear program [4]:

\[
\begin{align*}
\text{max} & \quad \sum_{s \in S} \sum_{a \in A} r(s, a) x(s, a) \\
\text{s.t.} & \quad \sum_{a \in A} x(j, a) - \sum_{s \in S} \sum_{a \in A} p(s, a, j) x(s, a) = \alpha_j \quad \forall j \in S \\
& \quad x(s, a) \geq 0 \quad \forall s \in S, a \in A
\end{align*}
\]

If \(x\) is the optimal solution to (1), the optimal policy \(\pi\) is given by (2) below. It turns out that \(\pi\) is a deterministic policy and uniformly optimal regardless of the initial distribution \(\{\alpha_j\}_{j \in S}\) [4]. However, such deterministic policies are undesirable in domains such as our UAV example.

\[
\pi(s, a) = \frac{x(s, a)}{\sum_{\hat{a} \in A} x(s, \hat{a})}.
\]

2.1 Randomness of a policy

We borrow from information theory the concept of entropy of a set of probabilities to quantify the randomness, or information content, in a policy of the MDP. For probabilities \(p_1, \ldots, p_n\) the only function, up to a multiplicative constant, that captures the information content is the entropy, defined by the formula \(H = -\sum_{i=1}^{n} p_i \log p_i\) [5]. We consider two entropy-based functions to measure randomness of a policy \(\pi\) for a MDP. We express these entropies as functions of the underlying frequency \(x\). The first is obtained by adding the entropy of policy \(\pi\) out of every state

\[
H_A(x) = \sum_{s \in S} \left( -\sum_{a \in A} \pi(s, a) \log \pi(s, a) \right)
= -\sum_{s \in S} \sum_{a \in A} \frac{x(s, a)}{\sum_{\hat{a} \in A} x(s, \hat{a})} \log \left( \frac{x(s, a)}{\sum_{\hat{a} \in A} x(s, \hat{a})} \right) .
\]

This additive entropy ignores dependencies between states in the MDP. Our second entropy-based measure, the weighted entropy, is obtained by averaging the entropy of
the policy out of each state, $\pi(s, \cdot)$ with the fraction of flow that reaches that state, in other words:

\[
H_W(x) = - \sum_{s \in S} \sum_{a \in A} \left( \frac{x(s, a)}{\sum_{\hat{a} \in A} x(s, \hat{a})} \right) \pi(s, a) \log \pi(s, a) \\
= - \frac{1}{\sum_{j \in S} \alpha_j} \sum_{s \in S} \sum_{a \in A} x(s, a) \log \left( \frac{x(s, a)}{\sum_{\hat{a} \in A} x(s, \hat{a})} \right).
\]

### 2.2 Trade-off between entropy and reward

We present below general methods for constructing randomized policies using $H_A$ and $H_W$ (the methods are not dependent on specific measures of entropy). We can obtain policies with maximal entropy but small reward by replacing the objective of Problem (1) with the definition of either the additive entropy $H_A(x)$ or weighted entropy $H_W(x)$. The reduction in reward can be controlled by limiting the feasible solutions to those policies that achieve at least a certain reward $R_{\text{min}}$. The following problem, for example, maximizes the weighted entropy while maintaining the reward above $R_{\text{min}}$:

\[
\begin{align*}
\max & \quad - \frac{1}{\sum_{j \in S} \alpha_j} \sum_{s \in S} \sum_{a \in A} x(s, a) \log \left( \frac{x(s, a)}{\sum_{\hat{a} \in A} x(s, \hat{a})} \right) \\
\text{s.t.} & \quad \sum_{a \in A} x(j, a) - \sum_{s \in S} \sum_{a \in A} p(s, a, j) x(s, a) = \alpha_j \\
& \quad \forall j \in S \\
& \quad \sum_{s \in S} x(s, a) \geq R_{\text{min}} \\
& \quad x(s, a) \geq 0 \quad \forall s \in S, a \in A
\end{align*}
\]

(3)

Let $R^*$ be the maximum reward value (from (1)). By varying the reward threshold $R_{\text{min}} \in [0, R^*]$ we can explore the trade-off between the achievable reward and entropy. Note that for $R_{\text{min}} = 0$ the above problem finds the maximum weighted entropy policy, and for $R_{\text{min}} = R^*$, Problem (3) returns the maximum reward policy with largest entropy. Unfortunately, functions $H_A(x)$ and $H_W(x)$ are neither convex nor concave in $x$, hence there are no complexity guarantees to solving Problem (3), even for a local optimal.

Due to this negative complexity result we explore a LP solvable in polynomial time. Given $\beta \in [0, 1]$ and a solution $\bar{x}$ with desired level of entropy, we return a maximum reward solution with an uncertainty structure (i.e., probability distribution over actions) similar to $\bar{x}$ by solving (4).

\[
\begin{align*}
\max & \quad \sum_{s \in S} \sum_{a \in A} r(s, a) x(s, a) \\
\text{s.t.} & \quad \sum_{a \in A} x(j, a) - \sum_{s \in S} \sum_{a \in A} p(s, a, j) x(s, a) = \alpha_j \\
& \quad \forall j \in S \\
& \quad x(s, a) \geq \beta \bar{x}(s, a) \quad \forall s \in S, a \in A \\
& \quad x(s, a) \geq 0 \quad \forall s \in S, a \in A \\
\end{align*}
\]

(4)
\( \bar{x} \) can be any solution with interesting uncertainty structure, e.g. uniform uncertainty where \( x(s,a) = 1/|A| \) (used below) or the maximum weighted entropy solution \( x^* \). By varying \( \beta \in [0,1] \) we gradually require a larger part of this uncertainty structure in the high reward solution. For \( \beta = 0 \) the above problem reduces to (1) returning the maximum reward solution \( R^* \); and for \( \beta = 1 \) the problem maximizes reward out of all solutions (henceforth this reward denoted \( \bar{R} \)) with at least as much uncertainty as \( \bar{x} \).

We now present two algorithms to obtain random solutions that maintain a reward of at least \( R_{\text{min}} \) (\( R_{\text{min}} \) set as a certain fraction of \( R^* \) obtained via 1). The first algorithm is based on solving Problem (3). Algorithm 2 (BRLP), uses Problem (4) to zoom-in on a solution that achieves a specified reward value while requiring an uncertainty structure from a given \( \bar{x} \). This algorithm approaches reward values \( R_{\text{min}} \in [\bar{R}, R^*] \), by increasing and decreasing the uncertainty requirement. Both algorithms obtain a solution with reward \( R_{\text{min}} \), but differ substantially. The solution for Algorithm 1 maximizes entropy, either \( H_A(x) \) or \( H_W(x) \), but due to lack of convexity property, does not run in polynomial time (even for a local optimal solution). BRLP on the other hand obtains a solution with just enough randomness to have its maximum reward solution meet the reward objective. Given the input \( \bar{x} \), this algorithm runs in polynomial time, since at each iteration it solves an LP and for a tolerance of \( \epsilon \), it takes at most \( O \left( \frac{R(0) - R(1)}{\epsilon} \right) \) iterations to converge (\( R(0) \) and \( R(1) \) are rewards corresponding to 0 and 1 values of \( \beta \)). The result of the two algorithms would be policies that, in domains like our UAV one, enable an agent to get a sufficiently high reward, e.g. surveying enough area, using randomized flying patterns.

**Algorithm 1** MAX-ENTROPY(\( R_{\text{min}} \))

1: Solve Problem (3) with \( R_{\text{min}} \), let \( x_{R_{\text{min}}} \) be optimal solution
2: return \( x_{R_{\text{min}}} \) (maximal entropy, reward \( \geq R_{\text{min}} \))

**Algorithm 2** BRLP(\( R_{\text{min}}, \bar{x} \))

1: Set \( \beta_1 = 0 \), \( \beta_2 = 1 \), and \( \beta = 1/2 \).
2: Solve Problem (4), let \( x_{\beta} \) and \( R(\beta) \) be the optimal solution and reward value returned
3: while \( |R(\beta) - R_{\text{min}}| > \epsilon \) do
4: if \( R(\beta) > R_{\text{min}} \) then
5: Set \( \beta_1 = \beta \)
6: else
7: Set \( \beta_2 = \beta \)
8: \( \beta = \frac{\beta_1 + \beta_2}{2} \)
9: Solve Problem (4), let \( x_{\beta} \) and \( R(\beta) \) be the optimal solution and reward value returned
10: return \( x_{\beta} \) (reward \( = R_{\text{min}} \pm \epsilon \), entropy related to \( \beta \bar{x} \))

Section 5 experimentally compares the two single-agent algorithms. Before turning to agent teams next, we quickly sketch approaches to applying these algorithms in single agent POMDPs. For single-agent finite-horizon POMDPs with known starting belief states, we convert the POMDP to a (finite horizon) belief MDP, which allows our
randomization algorithms such as BRLP to be applied, and return a randomized policy. However, addressing unknown starting belief states remains a key issue for future work.

3 From Single Agent to Agent Teams

The problem of policy generation for multiagent MDPs, i.e., where a centralized planner provides policies to multiple agents in a fully observable setting, is equivalent to generating policies for single agent MDPs for deterministic policies [6]. However, randomized policies in multiagent MDPs lead to miscoordination [10]. By giving each agent an identical pseudo-random number generator and setting the same seed or by sharing the random numbers in the fully observable domain, techniques from the prior section for single agent MDPs can generate randomized policies for multiagent MDPs avoiding miscoordination. Thus, this section focuses exclusively on decentralized POMDPs, that are not equivalent to single agent POMDPs [7]. Since decentralized POMDP models and algorithms have been developed recently, we discuss an illustrative multiagent domain to demonstrate key concepts in our algorithms.

3.1 MTDP: A Decentralized POMDP model

We use notation from MTDP [6] for our decentralized POMDP model; other models are equivalent [7]. Given a team of $n$ agents, an MTDP is defined as a tuple: $(S, A, P, \Omega, O, R)$. $S$ is a finite set of world states $\{s_1, \ldots, s_m\}$. $A = \times_{1 \leq i \leq n} A_i$, where $A_1, \ldots, A_n$ are the sets of action for agents 1 to $n$. A joint action is represented as $(a_1, \ldots, a_n)$. $P(s_t, (a_1, \ldots, a_n), s_f)$, the transition function, represents the probability that the current state is $s_f$, if the previous state is $s_t$ and the previous joint action is $(a_1, \ldots, a_n)$. $\Omega = \times_{1 \leq i \leq n} \Omega_i$ is the set of joint observations where $\Omega_i$ is the set of observations for agents $i$. $O(s, (a_1, \ldots, a_n), \omega)$, the observation function, represents the probability of joint observation $\omega \in \Omega$, if the current state is $s$ and the previous joint action is $(a_1, \ldots, a_n)$. We assume that observations of each agent is independent of each other’s observations i.e the observation function can be expressed as $O(s, (a_1, \ldots, a_n), \omega) = O_1(s, (a_1, \ldots, a_n), \omega_1) \cdots O_n(s, (a_1, \ldots, a_n), \omega_n)$. The agents receive a single, immediate joint reward $R(s, (a_1, \ldots, a_n))$.

For deterministic policies, each agent $i$ chooses its actions based on its local policy, $\Pi_i$, which is a mapping of its observation history to actions. Thus, at time $t$, agent $i$ will perform action $\Pi_i(\omega_t^i)$ where $\omega_t^i = \omega_t^1, \ldots, \omega_t^n$. $\Pi = \langle \Pi_1, \ldots, \Pi_n \rangle$ refers to the joint policy of the team of agents. However if the policies are randomized, agents obtain a probability distribution over a set of actions rather than a single action. Furthermore, this probability distribution is indexed by a sequence of action-observation tuples rather than just observations, since observations do not map to unique actions. Thus in MTDP, a randomized policy maps $\Psi_t^i$ to a probability distribution over actions, where $\Psi_t^i = \{\psi_t^1, \ldots, \psi_t^n\}$ and $\psi_t^i = \langle \alpha_{t-1}^i, \omega_t^i \rangle$. Thus, at time $t$, agent $i$ will perform an action selected randomly based on the probability distribution returned by $\Pi_i(\Psi_t^i)$. Furthermore we denote the probability of an individual action under policy $\Pi_i$ given $\Psi_t^i$ as $P_{\Pi_i}(a_t^i | \Psi_t^i)$. In this model, execution is distributed but planning is centralized; and agents don’t know each other’s observations and actions at run time.
3.2 Illustrative UAV team Domain

Our simple illustrative UAV domain is analogous to the illustrative multiagent tiger domain [8] except for the enemy — indeed, to enable replicable experiments, rewards, transition and observation probabilities from [8] are used. Consider two armed UAVs on a humanitarian mission hovering above two regions: Left and Right. One region has landmines which the UAVs must destroy to get a high positive reward and the other has fertile land where destruction creates high negative reward; but the UAV team is unaware of which region has the landmines. The UAVs can perform three actions Shoot-left, Sense and Shoot-right. Both UAVs may be observed by an enemy with equal probability, and our worst case assumption about the enemy is that it can eavesdrop or estimate the UAV observations. The UAV team must maximize the enemy’s uncertainty while ensuring reward above a certain threshold.

Action Sense leaves the state unchanged, but provides a noisy observation OR or OL, to indicate whether the landmine is to the left or right. The Shoot-left and Shoot-right actions are used to destroy a landmine, but the landmine is destroyed only if both UAVs simultaneously take either Shoot-left or Shoot-right actions. Unfortunately, if one takes a Shoot-left and the other Shoot-right they incur a very high negative reward as the landmine becomes highly activated, exploding at the slightest vibration. Following the tradition of the multiagent tiger domain, the problem is restarted once any Shoot action occurs to ensure an infinite horizon setting.

4 Rolling Down Randomization in MTDP

Let $p_i$ be the probability with which the enemy targets agent $i$, and Entropy$(i)$ the entropy for agent $i$’s policy. We design an algorithm that maximizes the multiagent weighted entropy, given by $\sum_{i=1}^{n} p_i \text{Entropy}(i)$, in MDPs while maintaining team reward above a threshold. Unfortunately, generating optimal policies for decentralized POMDPs is of higher complexity (NEXP-complete) than single agent MDPs and POMDPs [7], i.e., MTDP presents a fundamentally different class where we cannot directly use the single agent randomization techniques from Section 2.

To exploit efficiency of algorithms like BRLP, we convert the MTDP into a single agent POMDP. To this end, our new iterative algorithm called RDR (Rolling Down Randomization) iterates through finding the best randomized policy for one agent while fixing the policies for all the other agents — we show that such iteration of fixing the randomized policies of all but one agent in the MTDP leads to a single agent problem being solved at each step. Thus, each iteration can be solved via BRLP. For a two agent case, we fix the policy of agent $i$ and generate best randomized policy for agent $j$ and then iterate with agent $j$’s policy fixed. RDR thus builds on and significantly extends the iterative JESP algorithm [8]. RDR finds randomized policies, as opposed to JESP’s deterministic policies, leading to key representational and algorithmic differences in RDR (see below).

Overall RDR starts with a locally optimal set of joint deterministic policies, and then rolls down the reward, randomizing policies turn-by-turn for each agent. Rolling down from a local optimal point allows control of the the amount of reward loss in service of
gaining entropy; RDR starts from a local optimal because a local optimal obtained with restarts can provide a high quality solution at a lower cost than a global optimal [8]. The amount the reward can be rolled down is input to RDR. RDR then achieves the rolldown in 1/d steps where d is an input parameter.

**RDR Details:** For expository purposes, we use a two agent domain, but can be easily generalized to n agents. When constructing deterministic policies in decentralized POMDPs, fixing policy of one agent (say agent 2) enables us to create a single agent POMDP, if agent 1 uses an extended state, i.e at each time t, agent 1 uses an extended state \( \langle S^t, \omega^t_2 \rangle \) [8]. Here \( S^t \) is the current state and \( \omega^t_2 \) is the observation history of agent 2. However, given that RDR focuses on randomized policies, at each time t, we define an extended state in RDR for an agent, say agent 1, to be \( e^t_1 = \langle S^t, \Psi^t_2 \rangle \) where \( \Psi^t_2 \) is as introduced in previous section. By using \( e^t_1 \) as agent 1’s state at time t, given fixed policy of agent 2, we can define a single-agent POMDP for agent 1 with transition and observation function as follows.

\[
P'(e^t_1, a^t_1, e^{t+1}_1) = P((S^{t+1}, \Psi^{t+1}_2)|(S^t, \Psi^t_2), a^t_1)
\]

\[
= P(\omega^{t+1}_2|S^{t+1}, a^t_1, \Psi^t_2) \cdot P(S^{t+1}|S^t, a^t_2, \Psi^t_2, a^t_1) \cdot P(a^t_2|S^t, \Psi^t_2, a^t_1) \]

\[\quad \cdot O_2(S^{t+1}, a^t_2, a^t_1, \omega^{t+1}_2) \]

\[\quad \cdot O_1(S^{t+1}, a^t_2, a^t_1, \omega^{t+1}_2) \] \hspace{1cm} (5)

\[
O'(e^{t+1}_1, a^t_1, \omega^{t+1}_1) = Pr(\omega^{t+1}_1|e^{t+1}_1, a^t_1)
\]

\[\quad = O_1(S^{t+1}, a^t_2, a^t_1, \omega^{t+1}_2) \] \hspace{1cm} (6)

**Fig. 1.** Trace of UAV scenario

Thus, we can create a belief state for an agent i in context of j’s fixed policy by maintaining a distribution over \( e^t_i = \langle S^t, \Psi^t_i \rangle \). Figure 1 shows three belief states for agent 1 in the UAV domain. For instance \( B^2 \) shows probability distributions over \( e^t_2 \). In \( e^t_2 = \langle \text{Left}(SL, OL) \rangle \), Left is the current state, SL (Shoot Left) is the agent 2’s action at time 1, OL (Observe Left) is agent 2’s observation at time 2. The belief update rule derived from the transition and observation functions is given in (7), where denominator
provides transition probability. Immediate rewards for the belief states are assigned using (8).

\[
B_{i+1}^{e_{i+1}} = \sum B_i^e \cdot P(S_i^t, (a_1^i, a_2^i), S_{i+1}^{e_{i+1}}) \\
\cdot O_1(S_i^{t+1}, (a_1^i, a_2^i), \omega_{i+1}^{e_{i+1}}) \cdot \frac{P^{\Pi_2} \cdot \omega_{i+1}^{e_{i+1}}}{P\cdot \omega_i} \\
\cdot \sum R(S_i^t, (a_1^i, a_2^i)) \cdot P^{\Pi_2}(a_2^t | \omega_i) \\
\mathbb{R}(a_1^t, B_i^e) = \sum B_i^e \cdot \sum a_2^i R(S_i^t, (a_1^i, a_2^i)) \cdot P^{\Pi_2}(a_2^t | \omega_i) 
\]

(7)

Thus, RDR’s policy generation implicitly coordinates the two agents, without explicit communication. Randomized actions of one agent are planned taking into account the impact of possible randomized actions of its teammate on the joint reward. Algorithm 3 presents the pseudo-code for RDR, and shows how we can use beliefs over extended states $e_i$ to construct LPs that maximize entropy while maintaining a certain reward threshold. We first generate a locally optimal distributed POMDP policy using the JESP algorithm [8]. The variable \texttt{percentdec} varies between 0 to 1 denoting the percentage of reward that can be sacrificed by the joint policy. The step size denotes the amount of reward sacrificed during each iteration. RDR then iterates, with the number of iterations equal to 1/d.

The function \texttt{GenerateMDP} generates all reachable belief states (on lines 2 through 7) from a given starting belief state $b$ and hence a belief MDP is generated. The number of such reachable belief states is $O(|A_1|\cdot|Q_1|)^T$ where $T$ is the number of time steps. The number of extended states in each belief $B$ increases by a factor of $|A_2|\cdot|Q_2|$ with increasing time horizon $T$. Thus the time to calculate $B(b)$ for all extended states $b$, for all belief states $B$ in agent 1’s belief MDP is $O(|S|\cdot|A_1|\cdot|Q_1|\cdot|Q_2|)^T$. Lines 12 through 14 compute the reward for each belief state. The total computations to calculate the reward is $O(|S|\cdot|A_1|\cdot|A_2|\cdot|Q_1|\cdot|Q_2|)^T$. The belief MDP generated is denoted by the tuple $\langle B, A, \texttt{trans}, R \rangle$. We reformulate the MDP obtained to problem 4 in Section 2.2 and use the BRLP procedure to solve it. As pointed out in Section 2.2, BRLP uses an LP fixed number of times and hence has an upper bound polynomial time provided by the interior point methods. Thus, both the complexity of generating the MDP and the running of LP imply that RDR complexity favorably compares with that of JESP [8]. It is at worst within a polynomial factor of JESP.

5 Experimental results

Figures 2a and 2b show the results for single agent problems based on generation of MDP policies. The results show averages over 10 MDPs where each MDP represents a flight of a UAV, with state space of 28-40 states. These experiments compare the performance of three methods of randomization for single agent policies. In the figures, \texttt{BRLP} refers to BRLP in Section 2.2; whereas $H_W(x)$ and $H_A(x)$ refer to Algorithm 1 with these maximization functions. Figure 2a shows the average weighted entropy on the y-axis and reward threshold percent on the x-axis. The average maximally obtainable entropy for these MDPs is 8.89 and all three methods attain it at about 50%
threshold i.e an agent can attain maximum entropy if it is satisfied with 50% of the reward. However, if no reward can be sacrificed (100% threshold) the policy returned is deterministic. Figure 2b shows the run-times, plotting the execution time in seconds on the y-axis, and the reward threshold percent on the x-axis. Thus, if an agent is satisfied with obtaining 80% of maximum possible reward to maximize entropy, the run-time of $H_W(x)$ method is 15 secs.

![Comparison of Algorithms 1 and 2](image)

**Fig. 2.** Comparison of Algorithms 1 and 2

We conclude the following from Figure 2: (i) BRLP is significantly faster than the other methods, providing 7-fold speedup on average over the 10 MDPs over the entire range of thresholds. (ii) Algorithm 1 with $H_W(x)$ provides the maximum entropy, but the average gain in entropy is only 10% over the value obtained using BRLP for identical threshold rewards. Thus, BRLP provides a very favorable trade off of run-time to entropy, and we use this method in the multiagent case.

Table 1 shows the runtime results and entropy (in parenthesis) for $d$ varying from 1 to 0.125 versus two values of percentage threshold reward (10% and 50%) with $T$ set to 2. We conclude that as $d$ decreases, the run-time increases, but the entropy remains fairly constant for $d \leq .5$. Thus, in our next set of experiments, we fix $d$ to 0.5, as that provides the most favorable tradeoff. The graph in Figure 3a shows the experimental results for Algorithm RDR using averages over 5 examples. It plots the reward threshold percent on the x-axis and weighted entropy on the y-axis. Thus, if the team is willing to obtain only 90% of maximum reward with a time horizon $T = 3$, it gets a weighted entropy of 2.58 as opposed to 4.37 if it can sacrifice 50% under the same set of conditions for $d$ and $T$. Figure 3b studies the effect of miscoordination cost on entropy for five different variations of our UAV team domain. We plot multiplicative reduction in miscoordination cost on x-axis and entropy on y-axis, with reward threshold of 70% and time horizon of 2. For instance, for example 1, the original miscoordination cost provided an entropy of 1.6, but as the miscoordination cost is scaled down by a factor of 12, the entropy increases to 1.75.

Based on these experiments, we conclude that: (i) Greater tolerance of reward loss allows higher entropy. However reaching the maximum possible entropy is more difficult in multiagent domains. (ii) Varying $d$ produces only a slight change in entropy; thus we can use $d$ as high as 0.5 to cut down runtimes. (iii) RDR is time efficient because of
Table 1. Average run time (sec)/Entropy for RDR, $T = 2$

<table>
<thead>
<tr>
<th>Reward Loss</th>
<th>1</th>
<th>.5</th>
<th>.25</th>
<th>.125</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>.6 (.61752)</td>
<td>1.8 (.7326)</td>
<td>3.4 (.7366)</td>
<td>6.6 (.7365)</td>
</tr>
<tr>
<td>50%</td>
<td>.8 (1.52818)</td>
<td>1.4 (2.52)</td>
<td>3.2 (2.6)</td>
<td>7.2 (2.62624)</td>
</tr>
</tbody>
</table>

the underlying BRLP algorithm (iv) The lower the miscoordination cost, the higher the entropy.

6 Summary and Related Work

Optimizing policy randomization in single-agent and decentralized MDPs/POMDPs can be highly beneficial in settings where adversary actions, capabilities are unknown, but the adversary can exploit predictability in agent policies. To this end, this paper provides two key contributions: (i) it provides novel algorithms, in particular the polynomial-time BRLP algorithm, to randomize single-agent MDP and POMDP policies, while attaining a certain level of expected reward; (ii) it provides RDR, a new algorithm for generating randomized policies for decentralized POMDPs. RDR exploits BRLP, and thus is able to efficiently provide randomized policies.

Randomization as a goal has received little attention in literature, and it is seen as a means or a side-effect in attaining other objectives, e.g.; in resource-constrained MDPs [2] or memoryless POMDP policy generators [9] (for breaking loops). The problem of coordinating randomized policies has not been widely discussed. [10] do discuss coordinating multiple agents executing randomized policies in a distributed MDP setting. However in their work randomization occurs as a side-effect of resource constraints; furthermore, agents communicate to resolve the resulting miscoordination in agent teams. In contrast, our work explicitly emphasizes techniques to maximize entropy in policies, and attains implicit coordination (without communication) over randomized policies.
in decentralized POMDPs. Significant attention has been paid to learning in stochastic games, where agents must learn dominant strategies against explicitly modeled adversaries [3, 11]. Such dominant strategies may lead to randomization, but randomization itself is not the goal of the work. Our work in contrast does not explicitly model adversary payoffs, and instead explicitly attempts to maximize weighted entropy leading to nonlinear programs and hence the need to develop lower complexity BRLP. Furthermore, we focus on agent teams using decentralized POMDPs rather than stochastic games.

Algorithm 3 RDR(d, percentdec, x)

1: \(\pi_1, \pi_2, \text{Optimalreward} \leftarrow \text{DP}_{\text{JESP}}()\)
2: \(\text{stepsize} \leftarrow \text{percentdec} \cdot \text{Optimalreward} \cdot d\)
3: for \(i \leftarrow 1\) to \(1/d\) do
4: \(\text{MDP} \leftarrow \text{GenerateMDP}(b, \Pi_{(i+1)\mod_2} T)\)
5: \(\text{Entropy}, \Pi_i \leftarrow \text{BRLP}((\text{Optimalrew} – \text{stepsize}) \cdot i, x)\)

1: \(\text{GenerateMDP}(b, \pi_2, T)\):
2: \(\text{reachable}(0) \leftarrow \{b\}\)
3: for \(t \leftarrow 1\) to \(T\) do
4: \(\text{reachable}(t) \leftarrow \emptyset\)
5: for all \(B^t \in \text{reachable}(t – 1)\) do
6: for all \(a_1 \in A_1, \omega_1 \in \Omega_1\) do
7: \(\text{trans, reachable}(t) \leftarrow \text{UPDATE}(B^t \cdot a_1, \omega_1)\)
8: for \(t \leftarrow T\) downto 1 do
9: for all \(B^t \in \text{reachable}(t)\) do
10: for all \(a_1 \in A_1\) do
11: \(\mathcal{R}^{t+1}_1(B^t) \leftarrow 0\)
12: for all \(s \in S, \psi_2^t \leftarrow \langle a_2^{t-1}, \omega_2^t\rangle\) do (Equation 8)
13: for all \(a_2\) given \(\Psi^t_2\) do
14: \(\mathcal{R}^{t+1}_2(B^t) \leftarrow B^t(s^t, \Psi^t_2) \cdot R(s^t, \langle a_1^t, a_2^t\rangle) \cdot P(a_2^t | \Psi^t_2)\)
15: return \((B, A, \text{trans}, \mathcal{R})\)

1: \(\text{UPDATE}(B^t, a_1, \omega_1) :\)
2: for all \(s^{t+1} \in S, \psi_2^{t+1} \leftarrow \langle a_2^{t-1}, \omega_2^t\rangle\) do
3: \(B^{t+1}(s^{t+1}, \Psi_2^{t+1}) \leftarrow 0\)
4: for all \(s^t \in S\) do (Equation 7)
5: \(B^{t+1}(s^{t+1}, \Psi_2^{t+1}) \leftarrow B^t(s^t, \Psi_2^t) \cdot P(s^t, \langle a_1^t, a_2^t\rangle, s^{t+1}) \cdot O_1(s^{t+1}, \langle a_1^t, a_2^t, \omega_1^{t+1}\rangle) \cdot \)
6: \(\text{trans} \leftarrow \text{normalize}(B^{t+1}(s^{t+1}, \Psi_2^{t+1}))\)
7: return \(\text{trans}, B^{t+1}\)

References

The Use of MAS Rescue Simulation to Assess the Effectiveness of Planning for Urban Disasters: A Preliminary Study

Tomoichi Takahashi¹ and Nobuhiro Ito²

¹ Meijo University, Shiogamaguchi, Tempaku, Nagoya 468-8502, JAPAN, ttaka@ccmfs.meijo-u.ac.jp
² Nagoya Inst. of Tech., Gokisochou, Showa-ku, Nagoya, 466-8555 Japan, bobson@phaser.elcom.nitech.ac.jp

Abstract. We are thinking that multi agent system approaches is one of key technologies in disaster rescue simulations, since interactions with human activities should be implemented in them. In this paper, we show difficulties in evaluating rescue agents performances and propose an approach that aims to use MAS as a tool to check our urban lives. Form experiments of various maps, we shows that disaster rescue simulation has possibilities to be used to compare disaster simulation results.

1 Introduction

Agent based approaches have been accepted in various areas and multi agent systems (MAS) have been studied in various fields [7][2]. Putting simulation results to practical use requires involving human activities. This makes researches on MAS interesting and difficult.

We are thinking that MAS approaches are one of key technologies in disaster rescue simulations, since interactions with human activities should be implemented in them. Joining development of RoboCup rescue system (RCRS), we have recognized that a universal method for evaluating agents’ performance is necessary as well as ad hoc or task dependent analysis [4].

In this paper, we propose an approach that aims to use MAS as a tool to check our urban lives. First, we show problems in evaluating agent’s performances that we found at RoboCup Rescue competitions. Next, an automatic map generation method and results of rescue simulation using real map data are presented. We show there are some correlation between rescue simulation results and environment’s changes. In the last, we discuss possibilities of usage of MAS as tools that evaluate simulated areas are safe places for us.

2 Evaluations of MAS as a system

A step “guess → compute consequence → compare experiment” has been repeated to improve simulation models in science and engineering. In disaster fields, while it is really difficult to compare computed results with real ones, it is required to compare the computed results each other.
Fig. 1. Scores at 2004 RC games

2.1 Agent evaluation at RoboCup rescue

In RoboCup Rescue competitions, a score formula $V = (P + \frac{H}{Hint}) \times \sqrt{\frac{B}{B_{max}}}$ has been used in ranking teams [5]. $P$ is the number of living civilian agent, $H$ is Health Point (how stamina agents have) values of all agents and $B$ is the area of houses that are not burnt. Hint and Bmax are values at start, the score decrease as disasters spreads.

Fig. 1 shows relative scores of semi-final games at RoboCup 2004. Six teams (from team A to F) did rescue operations under different disaster conditions and at different areas. Vertical scales on the figure are Vs that are normalize with the high score. Table 1 shows scores of three teams at the same map with varying sensing conditions. Simulation results at column $r_1$ are the results of simulations that the seeing ability of agents are set the half of normal sensing ability $s$. Values at column $r_2$ are simulation results where the hearing abilities of agents are set half of $s$. They indicate one performance at one disaster situation does not guarantee their performances at other situations.

2.2 Evaluation as one of social systems and its uses

Jenning et al. proposed a framework called Social Level[3]. Their responsible societies are composed of the following components: members (entities solving the problem), environment (where members are situated), interaction means (ways member interact), and goals (motivations for members’ problem solving). To make clear analysis of rescue agents’ performance, we represent RCRS as $S = \{G, Ag, \Sigma, \mathcal{E}, Ac, \mathcal{C}\}$. The components correspond to the responsible society’s components in followings: $members = Ag$, environment $= \{\Sigma, \mathcal{E}\}$, interaction means $= \{Ac, \mathcal{C}\}$, goals $= G$. At RoboCup competitions, rescue agents in $Ag$ have been competed under conditions of varying $\mathcal{E}$ and $\mathcal{C}$.

In a case of RCRS, $G$ are composite of the number of human lives or extinguished fires. $Ag$ is a set of agents such as civilians, rescue agents. $\mathcal{E}$ specifies environments where disasters occur. They are GIS data, such as roads, crossings, buildings (including refuges) and the initial locations of agents. $\Sigma$ is a set of earthquake simulators, fire simulator, building & road collapse simulators and

<table>
<thead>
<tr>
<th>team</th>
<th>sensing conditions</th>
<th>normal</th>
<th>restricted</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{team X}</td>
<td>s</td>
<td>78.92</td>
<td>78.92</td>
</tr>
<tr>
<td>\text{team X}</td>
<td>$r_1$</td>
<td>79.92</td>
<td>78.91</td>
</tr>
<tr>
<td>\text{team X}</td>
<td>$r_2$</td>
<td>79.92</td>
<td>78.91</td>
</tr>
<tr>
<td>\text{team X}</td>
<td>$r_3$</td>
<td>79.92</td>
<td>78.91</td>
</tr>
<tr>
<td>\text{team Y}</td>
<td>s</td>
<td>97.69</td>
<td>35.41</td>
</tr>
<tr>
<td>\text{team Y}</td>
<td>$r_1$</td>
<td>83.49</td>
<td>90.87</td>
</tr>
<tr>
<td>\text{team Y}</td>
<td>$r_2$</td>
<td>83.49</td>
<td>90.87</td>
</tr>
<tr>
<td>\text{team Y}</td>
<td>$r_3$</td>
<td>83.49</td>
<td>90.87</td>
</tr>
<tr>
<td>\text{team Z}</td>
<td>s</td>
<td>88.24</td>
<td>83.30</td>
</tr>
<tr>
<td>\text{team Z}</td>
<td>$r_1$</td>
<td>51.45</td>
<td>45.76</td>
</tr>
<tr>
<td>\text{team Z}</td>
<td>$r_2$</td>
<td>51.45</td>
<td>45.76</td>
</tr>
<tr>
<td>\text{team Z}</td>
<td>$r_3$</td>
<td>51.45</td>
<td>45.76</td>
</tr>
</tbody>
</table>
traffic simulator. \( \mathcal{A} \) and \( \mathcal{C} \) are languages that agents in \( \mathcal{A} \) use in simulation. \( \mathcal{C} \) represents communication channel among agents and interaction between agents and \( \mathcal{E} \). This represents oral communication, telephone or wireless communication.

3 Tools to check Urban’s disaster prevention plans

3.1 Requirements for simulation

Governments plan to make towns so that people will live safely at disasters. Tools checking the effectiveness of the plans are required and following features are desired: (1) it handles various sorts of disasters (\( \Sigma \)), (2) it reflects disaster prevention plans that specifies rescue actions (\( \mathcal{A}, \mathcal{C} \)), (3) it simulates disasters at the \( \mathcal{E}_{present} \) and \( \mathcal{E}_{plan} \) that is after prevention plans.

The RCRS satisfies the first and second requirements. The third requires that the simulation results show sound tendency to \( \mathcal{E} \)'s changes. We select \( \mathcal{E} \) as parameters to check how Rescue simulations results depend on \( \mathcal{E} \).

3.2 Map generation from public GIS data

Free data for real urban road networks are provided. In Japan, digital maps that contains road networks and public buildings are available in XML format [1]. Since buildings are privately owned properties, they are not in them.

Building information is essential data for disaster simulations such as fire simulation, building collapse simulation and so on. They are created automatically using Voronoi diagram[6]. Fig. 2 shows RoboCup Rescue (RCR) Kobe map and automatically generated map. The number of generated buildings is set the same order as the RCR one. RCR map contains building data at 1995 when Kobe Awaji earthquake occurred. Maps of Nagoya are generated.

3.3 RCRS simulation results of Nagoya maps

Fig. 3 shows snapshots of RoboCup Rescue simulations. \( \times \) marked places are roads covered with debris. The colored \( \circ \) show agents. There two types of agents.
One type is civilian who moves to safe places autonomously. The other type is a rescue agent such as police, fire engines and ambulances. Black colored circles are damaged agents and other colored ones are active agents.

Table 2 show properties of maps in Nagoya where our universities are and the result of simulations. Numbers of the statistical data are area \((km^2)\) and the number of families. The network properties are the number of roads, nodes of networks and buildings. The buildings are generated be proportional to the number of families. Our rescue agents are used in simulations and the results are shown with two indexes. Surviving rates are the number of alive agents \(P\) and not burned rates are \(B/B_{max}\).

The numbers and the initial positions of agents & fire ignition are set in following ways: (1) The number of civilians is set to be proportional to real population. Real numbers of central stations are used and fire engines and ambulances are deployed proportional to the stations. Ten police agents are deployed to every ward. (2) Agents are uniformly distributed over the maps. This setting is assumed that an earthquake occurs in the daytime when people work outside. (3) Fires break out simultaneously at five points.

Damages to agents are different where they are when earthquakes happen. We simulated three different situations - a: all people are outsides, b: one half people are outside and the other half at home, and c: all people are at home-. The surviving rates change with people’s initial locations, while the non_burning rate does not change so much.

The lower rows of Table 2 show correlations coefficients between the simulation results and other features. The first two features are the number of rescue agents. Ambulance agents task is to save people and fire agents extinguish fires. The ambulance and fire agents are programmed to perform the task. In our simulation, the fire agents show good correlations. It means the fires are properly deployed as the size of map. With regard to the surviving rate, the situations b are off interpolated lines between a and c.

The other are network features of \(\mathcal{E}\). These features show correlations with not_burning factors. This indicates that the fire simulator involved in RSRS causes similar disasters over various maps, although it was tuned to the Hanshin Awaji earthquakes, and that the result can be used to compare them relatively.

4 Discussions and Conclusion

In future, MAS will deal in practical and social fields. The fields require not only simulations with more agents and real data, but also systematical analysis of results. We have not got any practical evaluation methods for social agents so far, because the social activities are composed of various tasks and are difficult to experiment in real environments.

In this paper, we showed difficulties in evaluating agent performances in rescue fields using RoboCup Rescue competitions results. Next, we applied the rescue simulations to various towns that’s environments are created based on real GIS data. We shows there some correlation between \(\mathcal{E}\)’s structure and simulation.
Table 2. Simulation results and target GIS data

<table>
<thead>
<tr>
<th>ward</th>
<th>surviving rate</th>
<th>not burned rate</th>
<th>statistics</th>
<th>network properties</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>Chikusa</td>
<td>100%</td>
<td>72%</td>
<td>39%</td>
<td>43%</td>
</tr>
<tr>
<td>Higashi</td>
<td>98%</td>
<td>77%</td>
<td>42%</td>
<td>24%</td>
</tr>
<tr>
<td>Kita</td>
<td>99%</td>
<td>78%</td>
<td>56%</td>
<td>68%</td>
</tr>
<tr>
<td>Nishi</td>
<td>100%</td>
<td>65%</td>
<td>40%</td>
<td>30%</td>
</tr>
<tr>
<td>Nakamura</td>
<td>100%</td>
<td>67%</td>
<td>31%</td>
<td>58%</td>
</tr>
<tr>
<td>Naka</td>
<td>100%</td>
<td>61%</td>
<td>26%</td>
<td>5%</td>
</tr>
<tr>
<td>Showa</td>
<td>99%</td>
<td>99%</td>
<td>27%</td>
<td>11%</td>
</tr>
<tr>
<td>Mizuho</td>
<td>99%</td>
<td>62%</td>
<td>23%</td>
<td>6%</td>
</tr>
<tr>
<td>Atsuta</td>
<td>99%</td>
<td>75%</td>
<td>25%</td>
<td>10%</td>
</tr>
<tr>
<td>Nakagawa</td>
<td>100%</td>
<td>74%</td>
<td>60%</td>
<td>60%</td>
</tr>
<tr>
<td>Minato</td>
<td>98%</td>
<td>79%</td>
<td>42%</td>
<td>67%</td>
</tr>
<tr>
<td>Minami</td>
<td>100%</td>
<td>69%</td>
<td>39%</td>
<td>58%</td>
</tr>
<tr>
<td>Moriyama</td>
<td>99%</td>
<td>79%</td>
<td>50%</td>
<td>64%</td>
</tr>
<tr>
<td>Midori</td>
<td>100%</td>
<td>75%</td>
<td>48%</td>
<td>76%</td>
</tr>
<tr>
<td>Meito</td>
<td>99%</td>
<td>70%</td>
<td>40%</td>
<td>55%</td>
</tr>
<tr>
<td>Tenpaku</td>
<td>99%</td>
<td>79%</td>
<td>52%</td>
<td>73%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fire agent</th>
<th>Ambulance agent</th>
<th>correlation between the above and followings</th>
<th>cf. the biggest RCR map’s data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>0.6% 0.7% -</td>
<td>0.6% 0.7% 0.5% 0.6%</td>
<td>1.369 1.480 1.078</td>
</tr>
<tr>
<td>Families</td>
<td>0.11 0.05 -</td>
<td>0.11 0.05 0.65 0.65</td>
<td></td>
</tr>
<tr>
<td>Buildings</td>
<td>0.43 0.06 0.72</td>
<td>0.43 0.06 0.72 0.78</td>
<td></td>
</tr>
<tr>
<td>Faces</td>
<td>0.29 0.09 0.69</td>
<td>0.29 0.09 0.78 0.78</td>
<td></td>
</tr>
</tbody>
</table>

results. This correlation indicates disaster rescue simulation has possibilities to be used to compare the disaster simulation results.

Authors appreciate RoboCup Rescue community that provides fine software environments and organizations that provide GIS data.

References

Safe Agents in Space: Preventing and Responding to Anomalies in the Autonomous Sciencecraft Experiment

Daniel Tran, Steve Chien, Gregg Rabideau, Benjamin Cichy
Jet Propulsion Laboratory, California Institute of Technology
Firstname.Lastname@jpl.nasa.gov

Abstract. This paper describes the design of the Autonomous Sciencecraft Experiment, a software agent that has been running on-board the EO-1 spacecraft since 2003. The agent recognizes science events, retargets the spacecraft to respond to the science events, and reduces data downlink to only the highest value science data. The autonomous science agent was designed using a layered architectural approach with specific redundant safeguards to reduce the risk of an agent malfunction to the EO-1 spacecraft. The agent was designed to be “safe” by first preventing anomalies, then by automatically detecting and responding to them when possible. This paper describes elements of the design that increase the safety of the agent, several of the anomalies that occurred during the experiment, and how the agent responded to these anomalies.

1 Introduction

Autonomy technologies have incredible potential to revolutionize space exploration. In the current mode of operations, space missions involve meticulous ground planning significantly in advance of actual operations. In this paradigm, rapid responses to dynamic science events can require substantial operations effort. Artificial intelligence technologies enable onboard software to detect science events, re-plan upcoming mission operations, and enable successful execution of re-planned responses. Additionally, with onboard response, the spacecraft can acquire data, analyze it onboard to estimate its science value, and only downlink the highest priority data. For example, a spacecraft could monitor active volcano sites and only downlink images when the volcano is erupting. Or a spacecraft could monitor ice shelves and downlink images when calving activities are high. Or a spacecraft could monitor river lowlands, and downlink images when flooding occurs. This onboard data selection can vastly improve the science return of the mission by improving the efficiency of the limited downlink. Thus, there is significant motivation for onboard autonomy.

However, building autonomy software for space missions has a number of key challenges and constraints; many of these issues increase the importance of building a reliable, safe, agent.

1. Limited, intermittent communications to the agent. A spacecraft in low earth orbit typically has 8 communications opportunities per day. This means that the spacecraft must be able to operate for long periods of time without supervision. For deep space missions the spacecraft may be in communications far less fre-
quent. Some deep space missions only contact the spacecraft once per week, or even once every several weeks.

2. Spacecraft are very complex. A typical spacecraft has thousands of components, each of which must be carefully engineered to survive rigors of space (extreme temperature, radiation, physical stresses). Add to this the fact that many components are one-of-a-kind and thus have behaviors that are hard to characterize.

3. Limited observability. Because processing telemetry is expensive, onboard storage is limited, and downlink bandwidth is limited, engineering telemetry is limited. Thus onboard software must be able to make decisions on limited information.

4. Limited computing power. Because of limited power onboard, spacecraft computing resources are usually very constrained. An average spacecraft CPUs offer 25 MIPS and 128 MB RAM – far less than a typical personal computer.

5. High stakes. A typical space mission costs hundreds of millions of dollars and any failure has significant economic impact. Over financial cost, many launch and/or mission opportunities are limited by planetary geometries. In these cases, if a space mission is lost it may be years before another similar mission can be launched. Additionally, a space mission can take years to plan, construct the spacecraft, and reach their targets. This delay can be catastrophic.

This paper discusses our efforts to build and operate a safe autonomous space science agent. The principal contributions of this paper are as follows:

1. We describe our layered agent architecture and how that enables additional agent safety.
2. We describe our knowledge engineering and model review process designed to enforce agent safety.
3. We describe the process the agent use to detect anomalies and how it responds to these situations.
4. We describe several of the anomalies that occurred during in-flight testing, the response of the agent, and what steps were taken to prevent its occurrence in the future.

This work has been done for the Autonomous Sciencecraft Experiment (ASE) [2], an autonomy software package currently in use on NASA’s New Millennium Earth Observer One (EO-1) [5] spacecraft.

In this paper we address a number of issues from the workshop call.

Definition of agent safety and how to build a safe agent – we define agent safety as ensuring the health and continued operation of the spacecraft. We design our agent to have redundant means to enforce all known spacecraft operations constraints. We also utilize declarative knowledge representations, whose models are extensively reviewed and tested. We use code generation technologies to automatically generate redundant checks to improve software reliability. Additionally, our experiment is also designed to fly in a series of increasing autonomous phases, to enable characterization of performance of the agent and to build confidence.
Robust to environment (unexpected) – our agent must be robust to unexpected environmental changes. Our agent uses a classic layered architecture approach to dealing with execution uncertainties.

How to constrain agents – because of strong concerns for safety, our agent architecture is designed to enable redundancy, adjustable autonomy, and fail-safe disabling of agent capabilities. The layering of the agent enables lower levels of the agent to inhibit higher-level agent behavior. For example, the task executive systems (SCL) does not allow dangerous commands from the planner to be sent on to the flight software. The flight software bridge (FSB) can be instructed to disable any commands from the autonomy software or to shutdown components of or the entire autonomy software. The EO-1 flight software also includes a fault protection function designed to inhibit potentially hazardous commands from any source (including the autonomy software, stored command loads from the ground, or real-time commands).

The remainder of this paper is organized as follows. First we describe the ASE software architecture, with an emphasis on how it enhances safe agent construction. Next we discuss the efforts made to prevent, detect, and respond to in-flight anomalies. Finally we present several of the anomalies that have occurred to date. We describe how the software responded to these anomalous situations, and the steps taken to prevent it from occurring in the future.

2 ASE

The autonomy software on EO-1 is organized as a traditional three-layer architecture [4] (See Figure 1.). At the top layer, the Continuous Activity Scheduling Planning Execution and Replanning (CASPER) system [1, 7] is responsible for mission planning functions. Operating on the tens-of-minutes timescale, CASPER responds to events that have widespread (across orbits) effects, scheduling science activities that respect spacecraft operations and resource constraints. Activities in a CASPER schedule become inputs to the Spacecraft Command Language (SCL) execution system [6].

SCL initiates a set of scripts that issue the complete sequence of commands to the flight software. Prior to issuing each command, constraints are checked again to confirm the validity of the command as well as ensure the safety of the spacecraft. After the command is sent, SCL checks for a successful initiation and completion of the command. When a full sequence for a data collection is complete, one or more image processing algorithms are performed which may result in new requests to the planner.

2.1 Mission Planning

Responsible for long-term mission planning, the ASE planner (CASPER) accepts as inputs the science and engineering goals and ensures high-level goal-oriented behavior. These goals may be provided by either the ground operators or triggered by the onboard science algorithms. The model-based planning algorithms enables rapid response to a wide range of operations scenarios based on a deep model of spacecraft
constraints, including faster recovery from spacecraft anomalies. CASPER uses repair-based techniques [1] that allow the planner to make rapid changes to the current plan to accommodate the continuously changing spacecraft state and science requests. During repair, CASPER collects a set of conflicts that represent violations of spacecraft constraints. Generic algorithms are used to select and analyze a conflict to produce a set of potential plan modifications that may resolve the conflict. Heuristics are used to select a potential modification, and the plan is updated and reevaluated for new conflicts. This process continues until no conflicts remain.

![Autonomy Software Architecture](image)

**Fig. 1. Autonomy Software Architecture**

### 2.2 Robust Execution

At the middle layer, SCL is responsible for generating and executing a detailed sequence of commands that correspond to expansions of CASPER activities. SCL also implements spacecraft constraints and flight rules. Operating on the several-second timescale, SCL responds to events that have local effects, but require immediate attention and a quick resolution. SCL performs activities using scripts and rules. The scripts link together lower level commands and routines and the rules enforce additional flight constraints.

SCL issues commands to the EO-1 flight software system (FSS), the basic flight software that operates the EO-1 spacecraft. The interface from SCL to the EO-1 FSS
is at the same level as ground generated command sequences. This interface is implemented by the Autonomy Flight Software Bridge (FSB), which takes a specified set of autonomy software messages and issues the corresponding FSS commands. The FSB also implements a set of FSS commands that it responds to that perform functions such as startup of the autonomy software, shutdown of the autonomy software, and other autonomy software configuration actions.

The FSS accepts low-level spacecraft commands which can be either stored command loads uploaded from the ground (e.g. ground planned sequences) or real-time commands (such as commands from the ground during an uplink pass). The autonomy software commands appear to the FSS as real-time commands. As part of its core, the FSS has a full fault and spacecraft protection functionality which is designed to:

1. Reject commands (from any source) that would endanger the spacecraft.
2. When in situations that threaten spacecraft health, execute pre-determined sequences to “safe” the spacecraft and stabilize it for ground assessment and reconfiguration.

For example, if a sequence issues commands that point the spacecraft imaging instruments at the sun, the fault protection software will abort the maneuver activity. Similarly, if a sequence issues commands that would expend power to unsafe levels, the fault protection software will shut down non-essential subsystems (such as science instruments) and orient the spacecraft to maximize solar power generation. While the intention of the fault protection is to cover all potentially hazardous scenarios, it is understood that the fault protection software is not foolproof. Thus, there is a strong desire to not command the spacecraft into any hazardous situation even if it is believed that the fault protection will protect the spacecraft.

2.3 Science Analysis

The image processing software is scheduled by CASPER and executed by SCL where the results from the science analysis software generate new observation requests presented to the CASPER system for integration in the mission plan.

This layered architecture for the autonomy SW is designed such that each lower layer is verifying the output of the higher layers. Requests from the science analysis, or from operators on the ground, are checked by the planner prior to being sent to SCL. The planner activities are checked by SCL prior to being sent on to the FSS. Finally, the FSS fault protection checks the SCL outputs as well.

3 Anomalies

As with any large software system and complex science scenarios, anomalous situations are expected to occur during operations. This section will describe how the ASE model was developed to enforce agent safety. We also discuss how the agent was
developed to detect for anomalies and several of the responses encoded within the model. Finally, we describe several of the anomalies that have occurred during in-flight testing, the cause of the anomalies, how the agent responded to these situations, and modifications taken to prevent it from occurring in the future.

3.1 Prevention

With the control aspects of the autonomy software embodied in the CASPER & SCL models, our methodology for developing and validating the CASPER and SCL models is critical to our safe agent construction process. These models include constraints of the physical subsystems including: their modes of operation, the commands used to control them, the requirements of each mode and command, and the effects of commands. At higher levels of abstraction, CASPER models spacecraft activities such as science data collects and downlinks, which may correspond to a large number of commands. These activities can be decomposed into more detailed activities until a suitable level is reached for planning. CASPER also models spacecraft state and its progression over time. This includes discrete states such as instrument modes as well as resources such as memory available for data storage. CASPER uses its model to continuously generate and repair schedules, tracking the current spacecraft state and resources, the expected evolution of state and resources, and the effects on planned activities.

Table 1: Sample safety analysis for two risks

<table>
<thead>
<tr>
<th></th>
<th>Instruments overheat from being left on too long</th>
<th>Instruments exposed to sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operations</td>
<td>For each turn on command, look for the following turn off command. Verify that they are within the maximum separation.</td>
<td>Verify orientation of spacecraft during periods when instrument covers are open.</td>
</tr>
<tr>
<td>CASPER</td>
<td>High-level activity decomposes into turn on and turn off activities that are with the maximum separation.</td>
<td>Maneuvers must be planned at times when the covers are closed (otherwise, instruments are pointing at the earth)</td>
</tr>
<tr>
<td>SCL</td>
<td>Rules monitor the “on” time and issue a turn off command if left on too long.</td>
<td>Constraints prevent maneuver scripts from executing if covers are open.</td>
</tr>
<tr>
<td>FSS</td>
<td>Fault protection software will shut down the instrument if left on too long.</td>
<td>Fault protection will safe the spacecraft if covers are open and pointing near the sun.</td>
</tr>
</tbody>
</table>
SCL continues to model spacecraft activities at finer levels of detail. These activities are modeled as scripts, which when executed, may execute additional scripts, ultimately resulting in commands to the EO-1 FSS. Spacecraft state is modeled as a database of records in SCL, where each record stores the current value of a sensor, resource, or sub-system mode. The SCL model also includes flight rules that monitor spacecraft state, and execute appropriate scripts in response to changes in state. SCL uses its model to generate and execute sequences that are valid and safe in the current context. While SCL has a detailed model of current spacecraft state and resources, it does not generally model future planned spacecraft state and resources.

Development and verification of the EO-1 CASPER and SCL models was a multi-step process.

1. First a target set of activities was identified. This was driven by a review of existing documents and reports. This allowed the modeler to get a high-level overview of the EO-1 spacecraft, including its physical components and mission objectives. Because EO-1 is currently in operation, mission reports were available from past science requests. These reports were helpful in identifying the activities performed when collecting and downlinking science data. For example, calibrations are performed before and after each image, and science requests typically include data collection from both the Hyperion (hyperspectral) and Advanced Land Imager (ALI) instruments.

2. Once the activities were defined, a formal EO-1 operations document [3] was reviewed to identify the constraints on the activities. For example, due to thermal constraints, the Hyperion cannot be left on longer than 19 minutes, and the ALI no longer than 60 minutes. The EO-1 operations team also provided spreadsheets that specified timing constraints between activities. Downlink activities, for example, are often specified with start times relative to two events: acquisition of signal (AOS) and loss of signal (LOS). Fault protection documents listing fault monitors (TSMs) were also consulted, using the reasoning that acceptable operations should not trigger TSMs.

3. With the model defined, CASPER was able to generate preliminary command sequences from past science requests that were representative of flight requests. These sequences were compared with the actual sequences for the same request. Significant differences between the two sequences identified potential problems with the model. For example, if two commands were sequenced in a different order, this may reveal an overlooked constraint on one or both of the commands. We were also provided with the actual downlinked telemetry that resulted from the execution of the science observation request. This telemetry is not only visually compared to the telemetry generated by ASE, but it can also be “played back” to a ground version of the ASE software to simulate the effects of executing sequences. The command sequences were aligned with the telemetry to identify the changes in spacecraft state and the exact timing of these changes. Again, any differences between the actual telemetry and the ASE telemetry revealed potential errors in the model. A consistent model was defined after several iterations of generating commands and telemetry, comparing with actual commands and telemetry, and fixing errors. These comparisons against ground generated se-
quences were reviewed by personnel from several different areas of the operations staff to ensure acceptability (e.g., overall operations, guidance, navigation and control, science operations, instrument operations).

4. Model reviews were conducted where the models are tabletop reviewed by a team of personnel with a range of operations and spacecraft background. This is to ensure that no incorrect parameters or assumptions are represented in the model.

Finally, a spacecraft safety review process was performed. By studying the description of the ASE software and the commands that ASE would execute, experts from each of the spacecraft subsystem areas (e.g., guidance, navigation and control, solid state recorder, Hyperion instrument, power) derived a list of potential hazards to spacecraft health. For each of these hazards, a set of possible safeguards was conjectured: implemented by operations procedure, implemented in CASPER, implemented in SCL, and implemented in the FSS. Every safeguard able to be implemented with reasonable effort was implemented and scheduled for testing. An analysis for two of the risks is shown below.

3.2 Detection

The EO-1 FSS has a set of Telemetry and Statistics Monitoring (TSM) tasks that monitor the state of the spacecraft. TSMs typically detect anomalies by comparing a state value with expected thresholds for that value.

The FSS also includes processes for transmitting engineering data from the spacecraft subsystems for recording and future playback. ASE tapped into this data stream so that it could automatically monitor spacecraft state and resources. The data is received at 4Hz and the relevant information is extracted and stored into the SCL database. SCL uses the latest information when making decisions about command execution. Spacecraft state and resources are checked:

- Prior to executing the command to verify command prerequisites are met.
- After executing the command to verify the receipt of the command.
- After an elapsed time period when the effects of the command are expected.

The SCL model also contains a set of rules that continuously monitor the state of the spacecraft and relays any change to the database to CASPER. CASPER compares the new data with its predicted values and makes any necessary updates to the predictions. Effects of these updates to the current schedule are monitored and conflicts detected. If a set of conflicts are detected, CASPER will begin modifying the plan to find conflict-free schedule.

During ground contacts, mission operators can monitor the EO-1 spacecraft telemetry in real-time, as well as playback recorded telemetry and messages. Limits are set on the various data points to alarm operators when values fall out of their expected ranges. To manually monitor ASE, we developed a set of telemetry points for each of the ASE modules. This is typically high-priority health and status data that is continuously saved to the onboard recorder and downlinked during ground contacts. The real-
time engineering data for EO-1 is monitored with the ASIST ground software tool developed at GSFC.

The FSB, which acts as a gateway, has several telemetry points to verify that we have enabled or disabled the flow of spacecraft commands and telemetry. It also has command counters for those issued to the ASE software. SCL provides telemetry on its state including counters for the number of scripts executed. CASPER provides statistics on the planning algorithm including the types of conflicts that it addresses and what changes it makes to the plan when repairing the conflicts. It also generates telemetry that identifies any differences it finds between the actual spacecraft state and the state it expects during the execution of the plan.

The telemetry points for each module is useful in providing a high level view of how the software is behaving, but debugging anomalies from this would be difficult. Therefore, each software module also saves more detailed data to log files stored on a RAM disk. As they are needed, these log files are downlinked either to debug new issues or to further validate the success of the test.

### 3.3 Response

Anomaly detection may trigger a response from any one of the software modules, or from the operations team. At the highest level, CASPER can respond to some anomalies by replanning future activities. For example, CASPER may delete low priority requests to alleviate unexpected over-utilization of resources. At the middle layer, SCL can respond to small anomalies with robust execution. In most cases, it can delay the execution of a command if the spacecraft has not reached the state required by the command. Because of interactions within a sequence, however, a command cannot be delayed indefinitely and may eventually fail. If a critical command fails, SCL may respond with additional commands in attempt to recover from the failure. This is typically a retry of the commands or set of commands that has failed. If the command remains failed, the effects of the command do not propagate to the SCL database, which may trigger CASPER to replan. Finally, at the lowest level, the EO-1 FSS fault protection is used to detect potentially hazardous spacecraft states and trigger commands to transition the spacecraft out of those states. For example, the Hyerion temperature increases while the instrument is in-use. The FSS fault protection monitors this temperature and initiates shut-down commands when the maximum temperature is exceeded.

### 4 Case Study

In this section, we describe several of the in-flight anomalies that have occurred, including the responses taken by the agent, and the changes performed to prevent future anomalies from occurring.
4.1 Anomaly Types

The anomalies that have occurred onboard can be classified into the following types: modeling, software, operator, hardware.

Modeling – This is the most common type of error, caused by an incorrect model of the spacecraft within CASPER and SCL. Many of these errors were not detected during testing and validation because the EO-1 mission did not have a high-fidelity testbed, requiring the development of simulators that made several incorrect assumptions of the spacecraft behavior.

Software – These are your standard software implementation errors that occur with any large project. The design of the agent needed to be robust to errors that occur between the CASPER, SCL, FSW, and science modules.

Operator – Commands are regularly issued from mission operators during ground contacts. These commands may modify the state of the spacecraft, so the agent will need to be robust to these situations.

Hardware – The Livingston 2 software component (described in section 5) was designed to detect and diagnose this type of error. However, because hardware errors are rare on spacecraft systems, we chose not to focus on detecting these.

The remainder of this section will describe four anomalies in detail, mainly of the modeling, software, and operator type. We will describe how they were caused, how the agent responded to the anomaly, and what work was done to prevent it from occurring in the future.

4.2 Onboard Anomalies

April 1, 2004

During this early stage of the project, we were testing a single response scenario where the onboard science module would analyze an image, and issue requests to the onboard planner for more images of that target. The scenario went as follows:

- Image Prudhoe Bay, Alaska
- Playback data from the solid state recorder into RAM for image processing (bandstripping)
- Run image classifier to identify snow, water, ice, and land. This was scheduled to run immediately after the bandstripping process.
- Run image classifier to identify for cloud coverage. This was scheduled to run 60 minutes after the bandstripping process.
- If the classification of the scene was above a threshold defined by mission scientists, request additional images of Prudhoe Bay.

Several of the constraints modeled within CASPER and SCL were:

- The bandstripping process cannot begin unless the target Prudhoe Bay was successfully imaged
The image classifiers cannot begin unless the bandstripping process was successful.

During the first ground contact following this scenario, mission operators noticed several warnings from SCL telemetry and that the onboard science module did not perform any image processing. After collecting log files from SCL and CASPER, and replaying back telemetry collected during the test, it was determined that SCL had failed the bandstripping script because of a timeout during the verification of the command completion. In actuality, this verification failure was not a result of bandstripping failing, but of a bug within the flight software time code. It is still interesting, however, to examine the response of the agent.

The failure of the bandstripping script resulted in a lack of change to a SCL database record. This record is continuously monitored by CASPER and a conflict with the scheduled image classifier algorithm was recognized. However, because the first image classifier algorithm was scheduled immediately after bandstripping, CASPER had already committed to executing the classifier activity. When making this type of commitment, CASPER locks the activity (preventing any re-scheduling) and sends the execution request to SCL. The command was received by SCL, but failed the prerequisite check, blocking the command from being sent to the science module.

The second image classifier was scheduled 60 minutes after the end of bandstripping, and thus CASPER was able to modify the plan to resolve the conflict by removing it from the schedule.

This anomaly demonstrated how the layered architecture ensured the safety of the agent. CASPER was not responsive enough to prevent the first image classifier from being issued to SCL, but the SCL prerequisite check failed and thus the command was not issued. However in the second case, CASPER was able to respond to this failure by removing the subsequent image processing activity from the schedule.

A modification of the agent architecture to prevent these false-positive anomalies from occurring would be to have redundant checks in the completion of the commands. In this example, a single SCL database item indicated that bandstripping had failed, when in fact, it had succeeded. The model could have been updated to check multiple database records for the status of the bandstripping, instead of relying solely on a single data point to verify completion of the command.

**July 15, 2004**

This anomaly demonstrates how SCL was able to respond to a verification failure of command sequence. During this test, the anomaly occurred during normal operations for an X-Band ground contact. The scenario was:

- Using the X-Band transmitter, downlink all images from the solid state recorder
- Clear all images from the solid state recorder

Several of the constraints modeled were:

- The correct voltage/current level of transceiver must be met prior to operating X-Band activities.
- The downlink must complete successfully prior to clearing all the images from the solid state recorder.
During the ground contact, mission operators noticed several warnings from SCL and also that EO-1 had not begun the X-Band downlink of images collected. The operators manually initiated the X-Band contact and completed dumping the data. After analyzing log files, it was determined that a prerequisite failure in the SCL model for the current/voltage of the transceiver prevented the contact from being initiated. As a result of the X-Band failure, SCL also rejected the command to clear all the images from the solid state recorder.

This was actually an error within the SCL model. An early version of the model included a constraint that the transceiver cannot be powered on unless the current/voltage was at the correct level. However, the threshold values for the current/voltage in reality are not valid until the transceiver is powered on.

Unfortunately, this modeling error slipped through our testing and validation process because of the lack of a high fidelity testbed. The EO-1 testbed did not have a transceiver for testing and therefore, the current/voltage values were static (at the “on” levels) in the simulation. Without valid values on the current/voltage prior to powering on the X-Band transceiver, our resolution to this problem was to simply remove the current/voltage constraint from the SCL model.

January 31, 2005

This anomaly describes CASPER’s response to an unexpected change in the state of the spacecraft. During one of the scheduled ground contacts, the agent did not initiate the command sequence as requested from mission planners. An anomaly had occurred that removed the contact sequence from the mission plan. After analysis of collected telemetry, the cause of the anomaly was due to human intervention with the spacecraft several hours prior. An unscheduled contact had been initiated by mission planners, which was performed externally from the onboard planner. The unscheduled contact required mission operators to perform a blind acquisition of EO-1 and manually power on the transceiver, changing the state of the onboard transceiver to “on”. At the end of this contact, the operators manually powered down the transceiver.

The change to the transceiver state resulted in an update to the SCL database, which propagated to the CASPER schedule and created a conflict with the next ground contact activity. The conflict was with a constraint in the CASPER model that only allowed the transceiver state to transition from “on” to “off” or from “off” to “on”. When the update to the transceiver state was received, it set the current state to the transceiver to “on”. This created a conflict with the next scheduled contact that had planned to turn the transceiver on when the state was already “on”. To resolve this conflict, CASPER removed the next contact from the plan. Once the mission operator powered down the transceiver at the end of the unscheduled contact, subsequent contacts were conflict free, but the deleted contact was not rescheduled due to the risk of inserting the goal too close to its scheduled time.

To prevent this anomaly for future operations, we simply removed the transition constraints from the CASPER model of the transceiver. While not ideal, it was determined that this presented no risk to the spacecraft, and allowed the ASE software to
support real-time contact requests from mission planners without affecting the remainder of the schedule.

In this anomaly, although the update to the state of the transceiver was short-lived as it was eventually powered off by mission operators, its affect on the planner propagated to the next scheduled contact, resulting in the contact being removed from the schedule. One possible solution to prevent this from occurring in the future is to delay resolving conflict until necessary. Some initial work has been started on CASPER to support time-sensitive repair heuristics, but is still experimental and was not deployed on EO-1.

February 26, 2005
This anomaly occurred during a normal data collect of an area in Peru. During the first contact following this image, SCL telemetry indicated 1 warning from execution. After analysis of log files, this was determined to be caused by a command verification failure when issuing the command to close the ALI cover. The response of SCL was to reissue the command to close the covers, ensuring that the covers would be closed at the end of the collect.

Further investigation into the problem showed that the cover did indeed close after the first command. However, due to a spike in CPU consumption during that time, SCL was not able to process the change in the database record indicating that the cover was closed. While SCL has the highest priority among the ASE tasks, it is not higher than any of the FSS tasks. We are still investigating the cause for the increase in CPU load.

In this situation, the actions of the agent were correct in preserving the safety of the spacecraft. However, a change to the model can be used to ensure this anomaly does not occur in the future. Again, similar to the first anomaly described in this paper, redundant checks to multiple SCL database items can be used to determine the true state of the covers. From example, the EO-1 Hyperion instrument covers have two data-points representing the state of the cover. One data-point indicates if the cover is either open or closed, while the other is a continuous value, representing how far the cover has been opened. A check that reasons using both of these data-points would be less prone to false-positives.

5 Livingston 2

More recently (Fall 2004), in collaboration with Ames Research Center, we have begun flying the Livingstone 2 (L2) [12] diagnosis system. Both L2 and CASPER use models of the spacecraft separate from the reasoning engine: the models are tailored for a particular application without the need to change the software, allowing reuse of the advanced reasoning software across applications. The diagnostic capability of an on-board agent can then use the models to monitor the health of the spacecraft and detect faults. Early development of the L2 model currently does not support responding to anomalous situations, only detection of them.
However, during the times of the described anomalies, L2 was not operational. Also its current model only supports monitoring the operations of the spacecraft and not the CASPER or SCL software. Therefore, anomalous situations within CASPER or SCL would not be detected by L2.

6 Related Work

In 1999, the Remote Agent experiment (RAX) [10] executed for a several days on-board the NASA Deep Space One mission. RAX is an example of a classic threetiered architecture [4], as is ASE. RAX demonstrated a batch onboard planning capability (as opposed to CASPER’s continuous planning) and RAX did not demonstrate onboard science. PROBA [11] is a European Space Agency (ESA) mission demonstrates onboard autonomy and launched in 2001. However, ASE has more of a focus on model-based autonomy than PROBA.

The Three Corner Sat (3CS) University Nanosat mission used CASPER onboard planning software integrated with the SCL ground and flight execution software [13]. The 3CS mission was launched in December 2004 but the spacecraft were lost due to a deployment failure. The 3CS autonomy software includes onboard science data validation, replanning, robust execution, and multiple model-based anomaly detection. The 3CS mission is considerably less complex than EO-1 but still represents an important step in the integration and flight of onboard autonomy software.

7 Conclusions

This paper has described the design of a safe agent for the Autonomous Sciencecraft Experiment along with several of the anomalies and the software’s responses that have occurred during in-flight testing. First, we described the salient challenges in developing a robust, safe, spacecraft control agent. Second, we described how we used a layered architecture to enhance redundant checks for agent safety. Third, we described our model development, validation, and review. Fourth, we described how the agent responds and detects anomalous situations. Finally, we described several case studies of anomalies that have occurred in-flight and the response taken by the agent to maintain the safety of the spacecraft.

8 Acknowledgements

Portions of this work were performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.
References

1. S. Chien, R. Knight, A. Steichert, R. Sherwood, and G. Rabideau, "Using Iterative Repair to Improve Responsiveness of Planning and Scheduling," Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling, Breckenridge, CO, April 2000. (see also http://casper.jpl.nasa.gov)


An Overview of SECMAP
Secure Mobile Agent Platform

Suat Ugurlu¹, Nadia Erdogan¹

¹Istanbul Technical University, Computer Engineering Department,
Ayazaga, 34390 Istanbul, Turkey
suat@suatugurlu.com, erdogan@cs.itu.edu.tr

Abstract. Mobile agent technology presents an attractive alternative to the
client-server paradigm; however, the lack of a feasible agent security model
seriously hinders the adoption of the agent paradigm. This paper describes a
mobile agent platform, Secure Mobile Agent Platform (SECMAP) and its
security infrastructure. SECMAP presents abstractions that ensure the
protection of agents and system components through a shielded agent model. It
provides secure agent communication and migration facilities, and maintains
security policy information to examine agent actions and to prevent
undesired/unauthorized activity.

1 Introduction

There exists a wide range of security issues in using mobile agents and, in spite of its
several advantages: the lack of a feasible agent security model seriously hinders a
wider adoption of mobile code based applications. When compared to traditional
systems, mobile agents face several security risks. Both mobile agents during their
life times and hosts executing mobile agents are under security threats [1], [2], [3].

A secure mobile agent system should not only support basic agent requirements
such as facilities for agent communication and agent mobility, but also must provide
a security model to protect agents and hosts without causing any overhead to the
programmer.

This paper describes a new mobile agent platform, Secure Mobile Agent Platform
(SECMAP) and its security infrastructure. Unlike other agent systems, SECMAP
proposes a new agent model named as the shielded agent model for security
purposes. A shielded agent is a highly encapsulated software component that ensures
complete isolation against unauthorized access of any type. SECMAP provides
secure agent communication and migration facilities as well, and maintains security
policy information to examine agent actions and to prevent undesired/unauthorized
activity. The system ensures protection of different agents and system components by
enforcing security policies for various agent activities and continuously monitors and
reports on the execution of an agent from its creation to its completion. SECMAP is written in Java and is therefore platform independent.

2  Security Model of SECMAP

In a mobile agent system, agents cannot be reliably associated with end users without taking certain precautions. The approach taken by SECMAP is to treat every agent as a distinct principal and to provide protection mechanisms that isolate agents. SECMAP differs from other mobile agents systems in the abstractions it provides to address issues of agent isolation.

SECMAP provides a lightweight implementation of agents; they are implemented as threads instead of processes. Each agent is an autonomous object with a unique identification and agents communicate via asynchronous message passing.

A Secure Mobile Agent Server (SMAS) resident on each node presents a secure execution environment on which new agents may be created or to which agents may be dispatched. A SMAS may operate in three modes according to the functionality it exhibits: standard mode (SM-SMAS), master browser mode (MB-SMAS), or security manager mode (SM-SMAS). SMAS working in standard mode provides basic agent services such as agent creation, activation, inactivation, destruction, communication, and migration. It also includes a policy engine that checks agent activity and resource utilization according to the rules that are present in a policy file, which has been received from a Security Manager. MB-SMAS an SM-SMAS are also both capable of supporting all functionalities of standard SMAS. However, they have additional responsibilities. MB-SMAS maintains the name-location directory of all currently active agents in the system. SM-SMAS, on the other hand, performs authentication of all SMAS engines and is in charge of the distribution of SMAS certificates.

SMAS provides functionalities that meet security requirements and allow the implementation of the shielded agent model. A shielded agent is a highly encapsulated software component that ensures complete isolation against unauthorized access of any type. On a request to create a new agent, SMAS instantiates a private object of its own, which is an instance of predefined object AgentShield, and uses it as a wrapper around the newly created agent by declaring the agent to be a private object of AgentShield object. This type of encapsulation ensures complete isolation, preventing other agents to access the agent state directly. An agent is only allowed to communicate with its environment over the SMAS engine through the methods defined in a predefined interface object, AgentInterface, which is made the private object of the agent during the creation process. The interface provides limited yet sufficient functions for the agent to communicate with SMAS. All variables of agents are declared to be private and they have corresponding accessor methods. Agents issue or receive method invocation requests through asynchronous messages over the secure communication facility of SMAS. Thus, a source that is qualified for a particular request, for example, that has
received the rights to communicate with a target agent, is granted to pass its message.

SECMAP allows the concurrent execution of several agents on the same host and each agent runs as a separate thread in the same memory area of the host. In this mode of operation, the shielded agent model suffices to guarantee inter agent isolation and protection.

SECMAP employs cryptographic techniques to meet security constraints. Each SMAS owns a certificate which is used to identify its identity and to encrypt and decrypt data. A request from a SMAS is not processed before the validity of the SMAS identity is verified. A SECMAP agent’s code and state information are kept encrypted during its life time using Data Encryption Standard (DES) algorithm. They are decrypted only when the agent is in running state on the host’s memory. Thus, an agent is identified as a black box on a host, except while in memory. To protect agents during migration over the network, agent code and state data are encrypted as well while in transfer and can only be decrypted on the target host after retrieving the appropriate DES key from the security manager.

SECMAP monitors, time stamps and logs all agent activity in a file, in order to be later analyzed to determine the actions an agent has carried out on the host. In case an unexpected result is recognized, the route of the agent can be traced and how the agent has executed on each host can be detected. In addition, in case of a threat, SMAS has the privilege to end the execution of an agent.

2.1 Security Policies

SECMAP employs a policy based authorization mechanism to permit or restrict agents to carry out certain classes of actions. Agent communication, migration, disk I/O, access to system resources are some of the events that require enforcement of security policies. SECMAP allows for policies to be dynamically defined and be enforced by intercepting agent service requests. Two types of policies are defined: agent policies and host policies. Agent policies are specified by the agent owner when he is deploying the agent and are carried together with the agent while migrating over the network. An agent policy simply defines the rights an agent possesses, such as disk access rights or the right to create network connections. A user may modify agent policies after deployment. Host policies, on the other hand, determine restrictions on access to the host resources by the agents. Policies are kept as encrypted XML files by each SMAS.

2.2 Agent Communication

SECMAP agents communicate via messages. The platform supports asynchronous message exchange primitives through methods of AgentInterface. Agent communication is secured by transferring encrypted message content through SSL. Agents are provided with a flexible communication environment where they can question the results of a message send request, wait for a response for a specified
period of time, and receive messages or replies when it is convenient for them.

### 2.3 Location Transparency

The system is managed with a decentralized control; several MB-SMAS and SM-SMAS may concurrently be active and they cooperate for a smooth execution. They share their data and communicate messages to keep them coherent. When initializing an S-SMAS on a node, the programmer specifies the addresses of the MB-SMAS and the SM-SMAS it should register itself to. Next, S-SMAS sends its agent list to MB-SMAS and, in return, receives the identities of all other agents active on the system. We call those S-SMAS that a MB-SMAS or a SM-SMAS cooperates with as its partners. When a MB-SMAS gets a request to return an agent identity, it cooperates with its partners to obtain the current agent identities. A similar mode of processing is true for SM-SMAS. If an SM-SMAS cannot authenticate a request, it directs it to its partners for possible authentication. Additionally, when an S-SMAS communicates with its MB-SMAS and SM-SMAS, it obtains the addresses of their partners and saves them as well, in order to use as a contact address in case its communication to its MB-SMAS or SM-SMAS fails. This approach adds robustness against network or node failures.

### 2.4 Agent Migration

SECMAP supports weak migration of agents between remote hosts on a call to the Move method of AgentInterface. The agent that wants to migrate should specify the address of the remote host where it wants to be transferred. When agent transfer completes, its new location information is updated on MB-SMAS automatically so that any new message destined to this agent is redirected to the correct SMAS. An agent cannot receive any messages while it is being transferred. Therefore, the agent programmer needs to question the result of the send operation and re-send the message if the operation has failed. All communication that is carried out between SMAS engines to complete the transfer of the agent is encrypted through SSL.

### 3 Related Work

Developers and researchers have taken a variety of approaches to provide security of mobile agent environments. Hohl [4] proposes what he refers to as Blackbox security to scramble an agent's code in such a way that no one is able to gain a complete understanding of its function. Proof carrying code [5] requires the author of an agent to formally prove that the agent conforms to a certain security policy. By digitally signing an agent, its authenticity, origin, and integrity can be verified by the recipient. The idea behind path histories [6] is to let a host know where a mobile
agent has been executed previously. State appraisal [7] attempts to ensure that an agent's state has not been tampered with and that the agent will not carry out any illegal actions through a state appraisal function that becomes part of the agent code. There does not seem to be a single solution to the security problems introduced and most of the solutions are inadequate in protecting agent and host data, while others that provide adequate protection cause an unacceptable overhead to the programmer. No DOS protection is available in any system because of the difficulty of its detection and prevention. However, a system should at least have a monitoring and logging mechanism to analyze agent activities and use these data to later prevent DOS attacks. A dynamic policy based security management is also absent in most systems.

4 Conclusions and Future Work

This paper describes a mobile agent platform, SECMAP, and its security infrastructure. The system has been especially developed against security threats that both agents and hosts may be exposed to. Security features are inserted into the system core at design time. The system has an open and flexible architecture that can further be enhanced in the future to meet additional requirements.

SECMAP allows for completely isolated lightweight agents with flexible and efficient communication facilities. Sources of requests are authenticated before they are processed to verify that they really come from their stated sources. SECMAP introduces trusted nodes into the infrastructure; to which mobile agents can migrate when required, so that sensitive information can be prevented from being sent to untrusted hosts. This approach does not appear to be fully explored elsewhere. Currently, work is in progress on detection and resolution of policy conflicts and enforcement of security policies. Our future work also includes the addition of dynamic policy creation capability to the architecture with the help of log analysis.

References

1. Christian F. Tschudin “Mobile Agent Security” Department of Computer Systems, Uppsala University, Sweden
3. V.Varadharan and D.Foster ."A Security Architecture for Mobile Agent Based Applications” World Wide Web: Internet and Web Information System, 6,93-122,2003
6. Uwe G. Wilhelm and Sebastian Stuaman “Protecting the itinerary of Mobile Agents” Laboratoire de Systemes d’Exploitation, Switzerland,1998