Toward A Quantitative Method for Assuring Coordinated Autonomy

Sagar Chaki[1]*, John M. Dolan[2], and Joseph Andrew Giampapa[1]

chaki@sei.cmu.edu jmd@cs.cmu.edu garof@sei.cmu.edu

[1]Software Engineering Institute, Carnegie Mellon University,
4500 Fifth Avenue, Pittsburgh, PA 15213-2612, USA
http://www.sei.cmu.edu/

[2]Robotics Institute, Carnegie Mellon University,
5000 Forbes Avenue, Pittsburgh, PA 15213-3890, USA
http://www.ri.cmu.edu/

Abstract. One of the biggest obstacles to the procurement and deployment of coordinated autonomous systems is the difficulty of assuring them, that is, to set and manage their performance expectations. This article introduces a reliability engineering assurance approach based on probabilistic model checking. It also introduces two models to guide and extend the reliability engineering approach: (1) a characterization of the range of autonomous coordination phenomena and (2) phases of a coordinated mission. Two instances of the models are implemented as discrete time Markov chains (DTMC). Results from in-progress validation experiments with robots are reported, as well.

Keywords: quantitative assurance method, coordinated autonomy, human-agent-robot teams, reliability engineering, probabilistic model checking

1 Introduction

As autonomous robots and agents become ubiquitous, the range of missions to which they apply will become more complex, as well. Autonomous robots and software agents, designed with highly specialized capabilities, will be applied to tasks that when combined, form more varieties of missions, with greater mission longevity, and in more diverse and dynamic environments. Many forms of coordination manage the integration of autonomous systems. They range from simple biologically inspired algorithms, to long-term team-oriented contractual commitments that involve conversations about such coordination elements as the formation of the overall team plan, role assignment, plan execution, monitoring, repair and eventual completion. One of the biggest obstacles to the procurement and deployment of coordinated autonomous systems [1] is the difficulty of assuring them, that is, to set and manage their performance expectations. This

* Authors are listed alphabetically by last name.

article introduces a reliability engineering (RE) assurance approach based on probabilistic model checking. Since reliability engineering has not been previously applied to autonomous coordination, this article suggests two models by which RE can be extended. The models can be used to guide the selection of dimensions and principal components by which reliability models can be derived.

In software engineering, *assurance* is the means by which one makes claims about properties of a system and proves them via reasoned arguments until a knowledgeable reviewer can read the assurance claims and arguments to support them and have justified confidence in the expected behaviors of the system [5]. *Testing and evaluation* is a synonym for assurance, but when both terms are used together, assurance usually applies more broadly, referring to any form of claim or logical argument that is focused on the system. *Quantitative methods* for assuring systems allow numeric measures to be made of them. Such measures not only establish the (non-)existence of the property being tested, but assign a number and scale to the property under consideration so that a comparison with other techniques for achieving it can be made. While test suites, arenas and field tests all provide means of quantitatively evaluating coordinated autonomy, they typically are presented in absence of a comprehensive assurance argument. At industrial robotics conferences, there is general public consensus that nobody knows how to make such an argument. This article is an attempt at responding to that problem.

The structure of the article is as follows. Section 2 introduces the models by which reliability engineering can accommodate coordinated autonomy. Section 3 briefly introduces reliability engineering and analysis techniques of relevance to our problem. Probabilistic model checking is introduced in section 4 and applied in two instances to our models. Our validation plan is described in section 5 and we conclude with section 6.

2 General Model

Autonomy is the property of an entity to have persistent, goal-directed behavior. Goal-directed behavior allows for alternative courses of action to achieve a goal in a dynamic and unpredictable environment. The dynamism and unpredictability of the environment is what makes autonomous systems challenging to assure, test and evaluate. The persistence ensures that the entity will attempt to achieve the goal as long as it can reason that it has a means of achieving it, or an interest in attempting to do so. The reasoning can involve reliance on commitments from other entities to assist it. Implicit in this description is the assumption of either altruistic or self-interested cooperation, and that the range of possible actions and consequences is computable, which implies that assurance claims about autonomous systems can be modeled and evaluated. Except when noted, we use the term *autonomous agent*, or simply *agent* to refer to a software autonomous entity, autonomous robot, or human in a limited socio-technical context.

To illustrate, consider the following example of altuistic autonomous cooperation. A robot might be capable of lifting a $25kg$ object as long as the object is

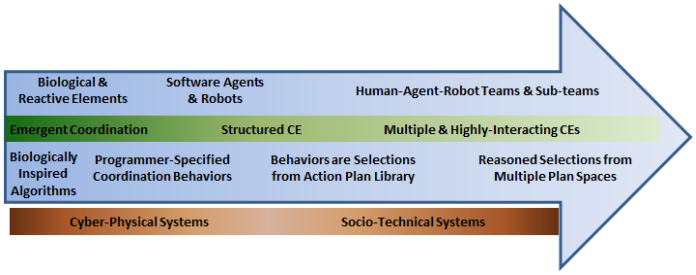


Fig. 1: Characterization of the Range of Autonomous Coordination Phenomena. CE = coordination element

no more than $2m$ in length and $30cm$ in thickness. If the length and weight of the object are doubled, the robot might not attempt to move it without the contemporaneous commitment from another robot with similar capabilities to help it move the object. The persistent and goal-directed behaviors are to attempt to move the object as long as the robots believe that it is within their collective capability to move it. The altruism is that they have no contract to guarantee a reward for self-interested participants.

Coordinated autonomy, used interchangeably with *autonomous coordination*, ensures that a goal can be achieved collectively by a group: (a) if the goal is achievable through the collective actions of the group and not just by one individual, (b) if individuals of a group can contribute capabilities that can help achieve the goal — be it by altruism, a form of capability-based coordination [14], or by short-termed service contracts, (c) through plan execution monitoring, which allows either (d) the individual to de-commit from the plan or service contract if they deem it is no longer achievable or in their interest, (e) the recruiting of additional individuals to collaborate on a team — possibly by short-termed contract — or (f) the repair of the team plan, which might involve reassigning roles or adopting another plan altogether. Coordination improves the likelihood to complete a mission by adding and managing redundancy of the resources used to achieve it. Coordination facilitates the scalability of additional resources by managing them: temporal-spatial localization, collision avoidance, synchronizing power drives, multi-tasking and parallelizing work. There are many forms of coordinated behaviors: more than even the missions or tasks that warrant them. For this, it is impossible to exhaustively enumerate them, but to group them into broad categories and to show how it is possible to derive variants of those categories as the situation warrants.

Fig. 1 illustrates such a grouping and a characterization of the range of autonomous coordination phenomena. Inexhaustive, it roughly situates the common forms of coordination that are frequently discussed in the Agents Community. In the order: *bottom, middle/green, top*, within the arrow are: (*coordination artifacts*) which implement forms of agent coordination and are often implemented in software for software agents and robots, but are also specified by norms and policies, (*coordination types*) as the name suggests, a simple taxonomy of coordination types, and (*participants*) the nature of the agents that are

participants in the coordination. The terminated arrow indicates that there is a limit to the complexity of the spectrum at one end of the range, but limitless complexity at the other end. That is, since the arrow-head includes any number of social sub-organizations of agents, each can create its own normative space that has an impact on individual, peer-to-peer, and group behavior.

The assurance of coordinated autonomy involves making and proving assurance claims about elements of the coordination artifacts layer. We call those elements *coordination elements (CEs)*. Coordination elements are situated in the coordination artifacts layer, but are generalized by the names of the coordination types in the layer above them. Contributions such as norms and policies from disciplines such as Economics, Game Theory and Computational Mechanism Design are accommodated by the coordination artifacts layer.

The brown band beneath the arrow situates two common systems engineering classifications, *cyber-physical systems (CPSs)* and *socio-technical systems* along the same axis, as a way of showing how the coordination maps to the respective systems engineering disciplines. Considering a cyber-physical system to be a software-driven system that has a closed loop with phenomena in the physical world, coordination algorithms on the CPS side of the spectrum will require more validation of their physical properties than coordination algorithms on the socio-technical end of the spectrum. This spectrum illustrates intuitions for how to balance the effort of validating any system of agents with physical characteristics against a system of agents that must reason and operate within a complex normative space. If an agent has physical properties, then it is likely that its principal contribution to the mission has something to do with its ability to perform a role in the physical world, otherwise the physicality, with all its attendant requirements for maintenance and spatial location, is unnecessary and the agent is either opportunistically included in a group or a human. If the model of its physical execution cannot be validated due to whatever reason(s), then even if its socio-technical properties can be validated, it is unlikely that the agent will be deemed suitable for its mission. The risk is too great that it will not be able to physically execute its role. The reverse of these conditions, by the same logic, where the CPS model is validated but the socio-technical component is not, may not invalidate the agent for the mission. Norms can be violated even if such violations are not welcomed by the other agents, and if the agent fills a critical physical role, then it justifies its existence in the group.

Returning to Fig. 1, *biologically inspired algorithms* (coordination artifacts) such as digital ants, swarms and flocking, are forms of *emergent coordination* (coordination types), in which the individual entities have no explicit notion of teamwork but the coordination is in the proverbial "eye of the beholder" as an emergent behavior derived from simpler ones which each agent manifests. The agents that participate in such forms of coordination may be insects, birds, or even reactive software and robotic systems (participant layer). The *programmer-specified coordination behaviors* (coordination artifacts) straddle coordination type generalizations of both *emergent coordination* and *structured coordination elements*. Systems that implement structured coordination elements, often soft-

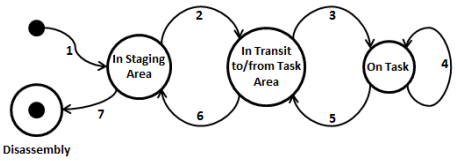


Fig. 2: State Transition Diagram Illustrating Phases of a Coordinated Mission

ware agents and robots (participant layer) are socially aware of other agents, aware of the infrastructure that allows them to discover other agents, and interact with them — often in the roles of providing or receiving services [14] and [12]. As agents in the autonomous coordination system become more self-aware and deliberative about their actions, the coordination elements that result in their assurable behaviors will more likely be drawn from an action plan library: a metaphorical, if not actual, repository of goal-directed behavior specifications that are motivated by normative space as much as by the task to be accomplished. As the socio-technical complexities increase, so too, is there an increase in the number of *multiple and highly-interacting coordination elements* which are the result of exposure to agents from multiple normative groups. The participants for such types of coordinated autonomy will be hybrid *human-agent-robot teams* [13]. Since humans act in a very complex normative and plan space, the upper bound limit on socio-technical complexity that must be assured will be determined by the number of normative and plan space interactions of software agents and robots.

The coordination elements to which the research described by this article applies are those that are selected by the "Structured CE" coordination type. The main coordination elements around which we focus our assurance arguments are:

Mission Objective which can include multiple objectives of equal or subordinate rank with respect to each other. Mission objectives provide the motivation for quantitative assurance claims and the metrics by which the claims are assessed.

Operating Context includes the physical, computing, and data communications environments, and other possible influences on the outcome.

Individual Capabilities The union of individual agent capabilities across all members of the team must satisfy the requirements of a team plan. The quantitative evaluation of individual capabilities will contribute to the evaluation of the quantitative metrics of the team.

Team Plan includes the roles, or sub-plan assignments to individual teammates or subgroups thereof, against which the individual capabilities will be evaluated. As mentioned previously, team plans and roles, as coordination elements, ensure that team scalability can be achieved, as well as remediate for individual deficiencies at achieving the mission objective.

If the above-specified coordination elements are to be evaluated in the context of an entire mission, it is necessary to subdivide the mission into principal

phases. The criteria for subdivision is based primarily on notions of the operating context, e.g. terrain, and the anticipated coordination needs for the activity that will be performed while in that context. Fig. 2 illustrates our characterization of a coordinated mission. Not all of the phases will be present in every mission. The following enumerations correspond to the edge labels in the figure.

1. *Assembly in a Staging Area* The assembly area is an important point in a robotic mission, as it is the closest that the support crews can be to the actual task area. It provides important logistical support to the deployed robotic team and access to additional resources, such as off-board computation, spare components and fuel resupply. It is also where the robotic team is assembled and prepared for performing their mission in the task area.
2. *Travel to Task Area from the Assembly Area* The transit of the autonomous team to the task area imposes constraints on the mission duration as well as on the spatial orientation of the robots for performing their mission. In the case of air vehicles, those in the air must loiter until all members of their squadron are airborn.
3. *Ingress into Task Area & Transition to Physical Roles* This is the phase when the robots prepare themselves for executing the team plan. It has been documented that for some robotic coordination missions this can be a critical moment that leads to unexpected team configurations. [9]
4. *Performance of Task* This can be further decomposed into the following coordination elements:
   (a) *Monitor Team Performance* Applicable to teams of type, *stuctured coordination elements,* this activity involves determining if any objectives will not be met based on how the team is performing its team task. If so, then the following coordination elements may be employed.
   (b) *Detect Failure* The team must agree that there is a failure that needs to be addressed.
   (c) *OPT: Repair* Typically, this refers to the following activities:
      – Recruit additional team members, either in substitution or to enrich the performance of the team, and/or
      – Adopt a new team plan, and/or
      – Reassign roles and transition to them.
   (d) *Consensual Team Plan Termination* This can occur due to a detected team failure, or due to completion of the team plan.
5. *Travel from Task Area to Departure Corridor* This is when the robots leave their roles and prepare to leave the task area.
6. *Return to Staging Area*
7. *Disassembly*

This non-exhaustive characterization identifies a segmentation of team activity in which all physical entities must engage, but also some software agents at a metaphorical level. This segmentation situates some of the coordination elements that are interleaved with actual physical processes. As the activities of the team evolve toward a predominantly socio-technical nature, other team activities would need to be added to account for more negotiation based on normative reasoning. We do not address that phenomena in this paper.

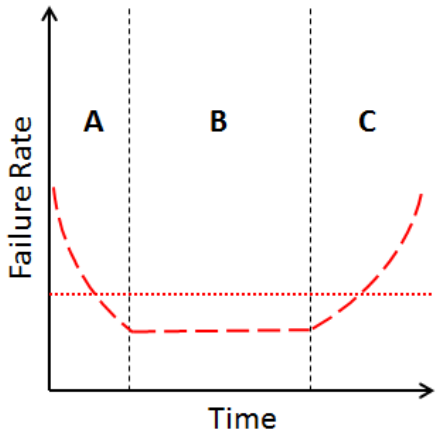


Fig. 3: The *bathtub curve*, shown as a long-dashed red line, which represents the characteristic shape of a reliability model. The dotted horizontal line illustrates a random and constant failure rate.

3 Reliability Engineering and Analysis Techniques

Reliability engineering provides quantitative methods and a precise language for measuring and discussing reliability [11] [7]. *Reliability* is the probability that a system will operate without failure for a given time. *Failure* is usually defined as the loss of a function. The *hazard rate* $h(t)$ is the instantaneous rate of failure of a component at a given time. A *reliability model* describes how the hazard rate changes over time. The characteristic shape of a reliability model, known as the *bathtub curve*, is represented in Fig. 3, which also illustrates the main phases of the reliability model [10]:

A - The *burn-in* period, or the phase of a component's life cycle when it fails due to defects introduced during the manufacturing process,
B - The *service life*, or the phase that corresponds to the component's expected useful work life,
C - The *wearout phase:* the part of a component's life in which failures will occur due to old age.

Manufacturers typically guarantee the service life of their product, that is, phase "B" of the bathtub curve. In the ideal world, the characteristic shape of the service life phase of a product's reliability model is a constant horizontal line with 0 failure rate. In reality, the failure rate is non-zero, though it may be very small, and will often have a non-zero slope. Since the reliability estimate of a product's service life is dependent on the reliability of its constituent parts, an accurate predictor of service life will be based on the shortest duration that is completely in common with the other interacting components.

Reliability analysis involves employing a collection of techniques for estimating the reliability model. The selection of techniques depend on the goals of the

analysis, which could range from cost-tradeoff analysis to critical failure analysis to determining operating temperatures and durability assessments. The selection of techniques also depends on the availability of data for a particular component, and how multiple components relate to each other for a particular analysis.

A *reliability prediction* is a quantitative assessment of the level of reliability, or lack of failure, in the design of a product. Since failure is often defined as a lack of function, reliability assessments of a product focus on assessments of each of its constituent functions. For each product function, studied in turn, it is necessary to identify the constituent components that comprise it and contribute to its function. Consequently, one of the first steps to analyzing the reliability of a system is to construct a *functional block diagram*, which illustrates the relationships among parts, assemblies and subsystems. The functional block diagram shows inputs and outputs but does not usually show how system elements are physically connected or positioned. From the functional block diagram, a *reliability block diagram (RBD)* is derived. The RBD consists of three basic types of building blocks: series configurations, parallel configurations, and combinations of series and parallel configurations. For each of these configurations, the reliability is calculated as follows:

Series Configuration The reliability of a system, when all the elements in the system are in a series, is the product of the individual reliabilities. Also known as "Lusser's law", the mathematical model is shown in Eq. (1), below.

$$R_S = R_1 \times R_2 \times \ldots \times R_n = \prod_{i=1}^{n} R_i \tag{1}$$

Active parallel configuration (redundancy) When all system elements are connected in parallel, on at the same time, and can take over in the event that any one element fails[1], the easiest way to calculate the reliability of the configuration is to determine the probability of all elements failing, and then to subtract this probability from 1. The formula is shown in Eq. (2), below.

$$R_S(t) = 1-([1-R_1(t)]\times[1-R_2(t)]\times\ldots\times[1-R_n(t)]) = 1-\prod_{i=1}^{n}[1-R_i(t)] \tag{2}$$

Standby parallel configuration (redundancy) One element is performing the necessary function and another element must be switched on in the event of failure. The failure detection, switch, and element that must be switched on, can each and jointly be a source of failure.[2]

Combined configuration Any combination of series and the above parallel configurations. To calculate the system reliability, first calculate the reliabilities of each individual configuration [7].

[1] Also known as *hot standby*.

[2] The formula for modeling this requires introducing terms that are out of scope with this article.

For the purposes of the quantitative assurance of coordinated autonomy, reliability engineering (RE) techniques are highly relevant but need to be extended. The underlying assumption is that the coordinated autonomous system will probably depart from a behavior that is being tested. Reliability engineering has the assumption of failure, and offers techniques to quantify its likelihood. Depature from an expected behavior for a coordinated autonomous system can be construed as a "failure", but if the system continues to function, provide a service and make progress toward any mission objectives, it really is not a failure. Failure of coordinated autonomy is not as much a focus of investigation as is the ability to identify and quantify differences of behavior. RE is applicable because individual behaviors can be modeled as if they were failures: the system either exhibits a specific behavior or it does not. Reliability models must be created for each individual autonomous coordination element and then tested for how it manifests itself as a behavior. If the question is to quantify which form of coordination element is more advantageous for a mission and context, then it is not important that the estimates of all behaviors sum to a probability of 1. Yet, if the question is to reason that the coordinated autonomous system will *not* exhibit unexpected behaviors, then equating the sum of the probabilities of all likely behaviors — including an "unknown" and undesired behavior — to 1, will be important and necessitate a different RBD.

The "bathtub" characteristic curve for reliability models is applicable, but to very limited contexts of high detail. Interesting to note is the intuitive and accepted practice of segmenting reliability model phenomena into phases so as to isolate predictable curves to which useful formulas can be applied. The quantitative description of coordinated autonomous systems will necessarily involve many levels of reliability modeling, with many segments that do not correspond to the equivalents of burn-in and wearout phases, although the characteristics of how power varies over time are likely applicable to describe the performance of individual robots during a mission.

Other contributions of reliability engineering that can be immediately used without modification are the equations for calculating reliabilities. They offer precise mathematical models of how to combine reliability measures. But in order to achieve that precision, they need to be applied to very specific contexts. An initial attempt to apply these mathematical models to coordinated autonomy is presented and discussed in Section 4.

4 Probabilistic Model Checking

Probabilistic model checking (PMC) is an algorithmic approach to decide whether a system S satisfies a property φ, denoted by $S \models \varphi$. The key difference between PMC and classical model checking [3] is that both S and φ are stochastic. Specifically, S is expressed as a (dicrete or continuous time) Markov chain, a Markov decision process (MDP), or a probabilistic timed automaton (PTA). The property is expressed in a probabilistic temporal logic such as probabilistic

computation tree logic (PCTL) [6]. The model checking algorithm then checks whether $S \models \varphi$ via exhaustive exploration of the statespace of S.

The advantage of probabilistic model checking (compared to simulation) is its exhaustive nature. Since realistic systems have very large (or even infinite) statespaces, simulation provides low coverage, and therefore a correspondingly higher margin of error. In contrast, PMC provides sound and precise results. However, successful application of PMC must overcome the *statespace explosion* problem – this is the price paid for being exhaustive. State-of-the-art solutions rely on two complementary approaches to ameliorate this problem.

First, while modeling the problem, manual abstraction is performed to eliminate details that are irrelevant to the target property φ. Second, modern probabilistic model checkers, such as PRISM [8], use symbolic data structures such as Multi-Terminal Binary Decision Diagrams (MTBDDs) [4] to verify systems with very large statespaces. In the rest of this section, we explore the efficacy of these two techniques for verifying the quality of distributed coordination algorithms in multi-agent systems by modeling and verifying a robotic demining scneario using the PRISM tool.

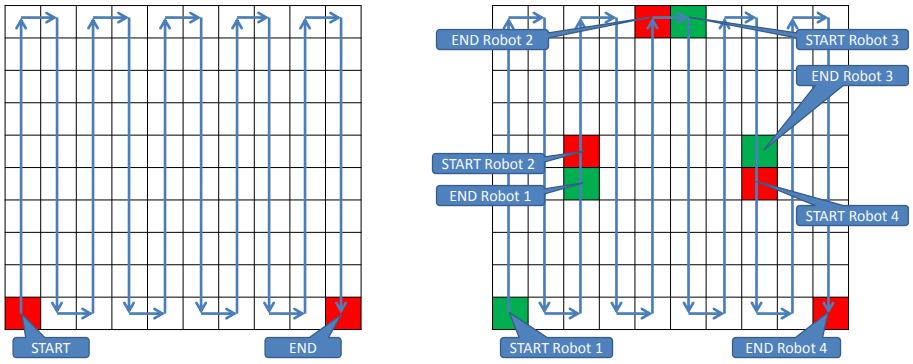
4.1 The Scenario: Robotic Demining

We consider a two-dimensional area (modeled as a 10×10 grid of cells) randomly seeded with mines. A team of N robots must sweep the area, detect each mine, and either defuse it or (failing which) mark it. The mission succeeds if all mines are detected and defused (or marked) within a specified deadline D. The mission is parameterized not only by N and D, but also the capabilities of each robot, the terrain, and coordination algorithm used by the robots.

We model the system using a discrete time Markov chain (DTMC) and use PRISM to compute the probability of mission success and expected terrain coverage under a variety of mission configurations. We use the results to pose and answer a number of hypotheses related to mission success and terrain coverage, and the number of robots used. Finally, we discuss possible areas of furture work.
Coordination Algorithms. Recall that N is the total number of robots. We consider the following two coordination algorithms:

1. **C0: Parallel Independent.** Each robot is assigned $\lceil \frac{100}{N} \rceil$ cells to demine a-priori. Each cell is allocated to exactly one robot. Each robot works independently and stops after demining all the cells allocated to it. This is an operator defined static coordination scheme.
2. **C1: Follow the leader.** All N robots move together in a team, with a single leader in the front and the remaining followers maintaining a fixed distance behind her. The leader performs mine detection, defusing, and path planning. If the leader is disabled by a mine explosion, one of the followers (decided by a leader election protocol) takes over as the new leader and continues.

Path Planning. We assume that the robots follow a pre-determined path through the grid, as shown in Fig. 4(a). In the case of coordination **C0**, if $N = 1$, then



(a) (b)

Fig. 4: Path followed by robots.

the single robot follows the path shown in Fig. 4(a). If $N > 1$, then each robot follows the same path, but only over the cells allocated to it. We assume that a robot is always allocated cells that are contiguous in the path shown in Fig. 4(a). For example, if $N = 4$, the starting and ending cells of each robot are shown in Fig. 4(b). In the case of coordination **C1**, the entire team follows the path shown in Fig. 4(a).

Behavior In a Cell. In the case of coordination **C0**, a robot maintains two variables, cells – the number of cells it has processed so far, initialized to zero, and clock – the time elapsed, initialized to zero. In each cell, the behavior of a robot is expressed via the DTMC shown in Fig. 5. In the following, t_k means the transition labeled k in Fig. 5.

The robot begins in state INIT. If all cells allocated to it have been processed (t_1), it moves to state DONE and loops there forever (t_2). Otherwise, it increments cells (t_3) and moves to state DETECT_MINE.

From DETECT_MINE, it either exceeds the deadline (t_4) and loops forever in state TIMEOUT (t_5), or proceeds with detecting a mine (t_6). The result of mine detection is either an explosion with probability p_explode_detect (t_7), a mine found with probability p_detect_mine (t_{10}), or no mine found (t_9). If there is an explosion, the robot loops forever in state BLOWNUP (t_8).

If no mine was detected (state NOT_DETECTED), then we assume that with probability p_false_neg, there is actually a mine. In this case, with equal likelihood, the robot either explodes (t_{11}) or moves to the next cell (t_{12}). In the latter case, we indicate mission failure (since a mine has been missed) by setting the flag failed to TRUE. Finally, with probability (1 - p_false_neg), the robot moves to the next cell (t_{13}), continuing with its mission. The probability p_false_neg is a function of both the robot's detecting capability and the terrain, as discussed later.

If a mine was detected, the robot attempts to defuse it. We assume that the robot is in one of three defusing situations with increasing difficulty – easy, medium and hard. Initially (DEFUSE1), the robot assumes that it is in the

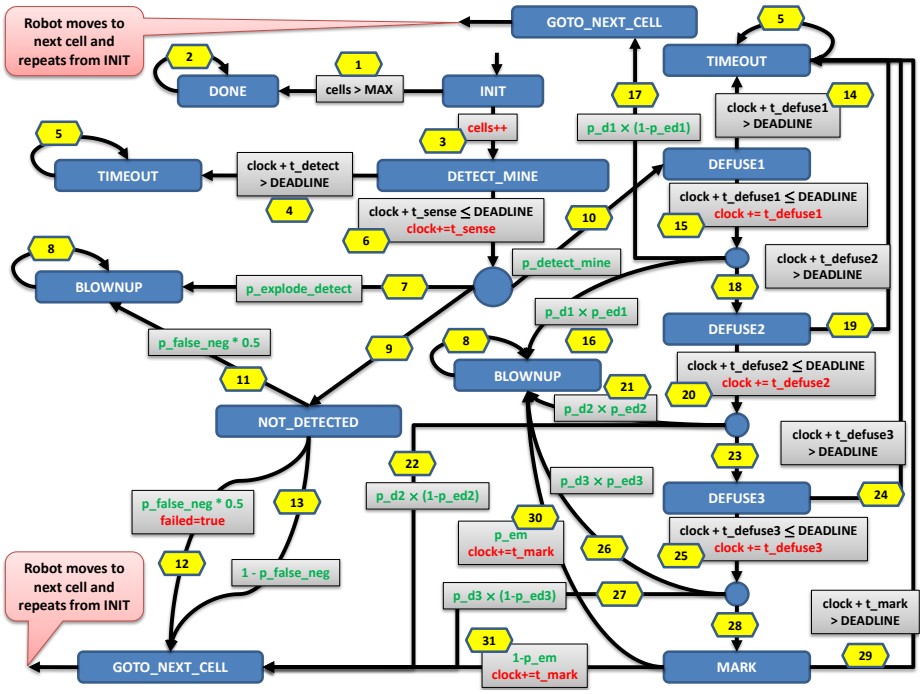


Fig. 5: Behavior of a robot in each cell. Transitions are numbered for ease of reference, and labeled by associated probabilities (green), guards (black) and commands (red). t_k = transition number k; TRUE guards and 1.0 probabilities are ommited for brevity; implied probabilities (since probabilities over all outgoing transitions from each state must sum to 1.0) are ommited. For example, the probability of t_2 is 1.0, the guard of t_2 is TRUE, and the probability of t_9 is $(1 - \mathsf{p\_explode\_detect} - \mathsf{p\_detect\_mine})$. States (e.g., TIMEOUT) and transitions (e.g., t_5) are repeated as needed to reduce clutter.

easy defusing situation. From this state, it either times out (t_{14}), or updates its clock and proceeds with the defusing operation (t_{15}). The result is either an explosion with probability ($\mathsf{p\_d1} \times \mathsf{p\_ed1}$) ($t_{16}$), successful defusal of the mine with probability ($\mathsf{p\_d1} \times (1 - \mathsf{p\_ed1})$) ($t_{17}$), or a decision to move on to the medium defusal scenario (t_{18}). Here, p_d1 is the probability that the robot is actually in an easy defusing situation, and p_ed1 is the probability that there is an explosion given that the robot is trying to defuse in an easy situation. As discussed later, while p_d1 is a function of the terrain, p_ed1 is a function of the robot's defusing capability.

In the medium defusal scenario (DEFUSE2), the robot either: (a) times out (t_{19}), or (b) updates its clock (t_{20}) and then probabilistically blows up (t_{21}), successfully defuses the mine (t_{22}), or moves to the hard defusal scenario (t_{23}). The probabilities involved in this step are: p_d2 – the terrain-dependent probability that the robot is actually in a medium defusing situation, and p_ed1 – the

probability (dependendent on the robot's defusing capability) that there is an explosion given that the robot is trying to defuse in a medium situation.

In the hard defusal scenario (DEFUSE3), the robot either: (a) times out (t_{24}), or (b) updates its clock (t_{25}) and then probabilistically blows up (t_{26}), successfully defuses the mine (t_{27}), or attempts to mark the cell (t_{28}) as being mined. The probabilities involved in this step are: p_d3 – the terrain-dependent probability that the robot is actually in a hard defusing situation, and p_ed3 – the probability (dependendent on the robot's defusing capability) that there is an explosion given that the robot is trying to defuse in a hard situation.

Finally, when the robot attempts to mark the cell, it either: (a) times out (t_{29}), or (b) updates its clock and then either blows up (t_{30}) with probability p_em, or succeeds (t_{31}) and continues to the next cell. The probability p_em of an explosion during the marking operation is a function of the robot's defusing capability, as discussed later.

Model Parameters. The DTMC in Fig. 5 is parameterized by the following:

1. The number of robots N. Note that, in Fig. 5, $\mathsf{MAX} = \lceil \frac{100}{N} \rceil$ since the cells are allocated equally to each robot.
2. The deadline D. Note, in Fig. 5, that $\mathsf{DEADLINE} = D$. Depending on the experiment, D was either fixed or varied.
3. The time required for detecting a mine (t_detect), defusing a mine (t_defuse1, t_defuse2 or t_defuse3 depending on the level of difficulty), and to mark a cell (t_mark). For our experiments, we assigned them fixed values.
4. The probability (p_detect_mine) of detecting a mine in a cell. For our experiments, this was fixed.
5. The remaining probabilities in Fig. 5 were computed from the terrain and the robot's capabilities as discussed next.

Modeling Terrain and Robot Capabilities. The robot's mine detection capability was modeled by a parameter DET with three possible values – LOW, MEDIUM and HIGH. The robot's mine defusing capability was modeled by a parameter DEF with three possible values – LOW, MEDIUM and HIGH. The terrain was modeled by six independent parameters: (i) p_fn_dc0, p_fn_dc1 and p_fn_dc2 are the probabilities of a false negative (i.e., mine present but not detected) given that DET = LOW, MEDIUM and HIGH, respectively; and (ii) p_d1, p_d2 and p_d3 are the probabilities of being in an easy, medium, or hard defusing situation, respectively. For our experiments, all six terrain parameters were assigned constant values, but in future experiments we plan to use these to represent the performance capabilities of individual robots to move precisely over the terrain.

Remaining Probabilities. The probability of a false negative in Fig. 5 are computed as follows:

$$\mathsf{p\_false\_neg} = \begin{cases} \mathsf{p\_fn\_dc0} & \text{if } \mathsf{DET} = \mathsf{LOW}, \\ \mathsf{p\_fn\_dc1} & \text{if } \mathsf{DET} = \mathsf{MEDIUM}, \\ \mathsf{p\_fn\_dc2} & \text{if } \mathsf{DET} = \mathsf{HIGH}. \end{cases}$$

The probability of an explosion while detecting a mine is computed as follows:

$$\mathsf{p\_explode\_detect} = \begin{cases} 10^{-4} & \text{if } \mathsf{DET} = \mathsf{LOW}, \\ 10^{-5} & \text{if } \mathsf{DET} = \mathsf{MEDIUM}, \\ 10^{-6} & \text{if } \mathsf{DET} = \mathsf{HIGH}. \end{cases}$$

Finally, the probabilities of an explosion while defusing or marking a cell are computed as follows:

$$\mathsf{p\_ed1} = \mathsf{p\_ed2} = \mathsf{p\_ed3} = \mathsf{p\_em} = \begin{cases} 10^{-2} & \text{if } \mathsf{DEF} = \mathsf{LOW}, \\ 10^{-3} & \text{if } \mathsf{DEF} = \mathsf{MEDIUM}, \\ 10^{-4} & \text{if } \mathsf{DEF} = \mathsf{HIGH}. \end{cases}$$

Leader Election. In the case of coordination **C1**, we have the additional complexity of electing a new leader in case there is an explosion. To this end, we modify the model in Fig. 5 as follows. First, we add a variable team_sz that indicates the current size of the team. We initialize team_sz to N, the number of robots. In addition, whenever there is an explosion (i.e., we reach state BLOWNUP in Fig. 5), we first check the current value of team_sz. If the value is 1 (i.e., the last robot exploded), we loop in state BLOWNUP. Otherwise, we either timeout, or we update the clock (by t_elect_leader), decrement team_sz, and either proceed to the next cell (state GOTO_NEXT_CELL) with probability p_elect_leader – this means that a new leader was elected successfully – or loop in state BLOWNUP – this means that leader election failed. For our experiments, the paramaters t_elect_leader (time for leader election) and p_elect_leader (probability of successful leader election) were assigned constant values.

4.2 Experiments

We experimented with two metrics of mission success: (i) *succ* = probability of covering all the cells without blowing up or missing a single mine; (ii) *cov* = expected number of cells defused or marked. The goal of these experiments is to demonstrate the suitability of using probabilistic model checking to make appropriate tradeoff decisions when designing robotic missions, and to form and validate specific hypotheses in the context of such missions. All our experiments were performed on an Intel Core i7 machine with four cores (each running at 2.7GHz) and 8GB of RAM. We used PRISM version 4.0.3, which was the latest version available at the start of this project. All our PRISM models, results, as well as instructions to reproduce them are available at `http://www.contrib.andrew.cmu.edu/~schaki/discover/arms13.tgz`.

Experiments with succ. Note that if we use coordination **C0**, increasing N has no effect on *succ*. Therefore, we first evaluated the effect of increasing N (from 1 to 10) on *succ* using coordination **C1**. Table 1 summarizes the results for four assignments – A0, A1, A2 and A3 – to parameters DET and DEF. In each case, *succ* increases with N. For both A0 and A1, *succ* is quite small, and does not seem to saturate even for $N = 10$. Indeed, the difference between A0 and A1 is marginal, indicating that an improvement in defusing capability alone provides

	succ			
N	A0	A1	A2	A3
1	1.51E-05	2.62E-05	0.3659	0.6026
2	1.08E-04	1.72E-04	0.6430	0.7570
3	3.90E-04	5.73E-04	0.7468	0.7766
4	9.57E-04	1.30E-03	0.7724	0.7782
5	1.80E-03	2.29E-03	0.7771	0.7783
6	2.80E-03	3.34E-03	0.7778	0.7783
7	3.77E-03	4.27E-03	0.7779	0.7783
8	4.58E-03	4.96E-03	0.7779	0.7783
9	5.15E-03	5.41E-03	0.7779	0.7783
10	5.51E-03	5.67E-03	0.7779	0.7783

Table 1: Results for *succ* with increasing N and coordination **C1** and the following assignments to DET and DEF: A0 = (DET=LOW, DEF=LOW); A1 = (DET=LOW,DEF=HIGH); A2 = (DET=HIGH,DEF=LOW); A3 = (DET=HIGH,DEF=HIGH).

very little benefit in terms of mission success. In contrast, for A2, *succ* is orders of magnitude higher and seems to saturate around $N = 5$. This suggests that improving a robot's mine detection capability alone provides a big boost to the probability of mission success, all else being equal. Finally, even though A2 and A3 saturate to similar *succ* values, A3 enables us to achieve the same mission success rate using fewer robots. For example, A3 provides 75% mission success with 2 robots, while A2 only provides this rate with at least 4 robots.

Experiments with cov. Next, we evaluated the effect of increasing N (from 1 to 10) on *cov* using both coordinations – **C0** and **C1**. Table 2 summarizes the results for four assignments – A0, A1, A2 and A3 – to parameters DET and DEF. In each case, *cov* increases with N. In addition, *cov* increases more if detection capability is increased, compared to an increase in defusing capability. This is the same trend as in the case of *succ*. Finally, coordination **C1** always provides equal or more coverage compared to **C0**, indicating that the more sophisticated coordination embodied by **C1** leads to improved coverage (in addition to a higher likelihood of success, as seen earlier). For example, in the case of A3, both coordinations provide equal coverage with $N = 1$. However, the coverage produced with **C1** jumps rapidly to more than 99% with even $N = 2$. In contrast, the coverage with **C0** ramps up much more slowly, crossing 98% with only $N = 7$, and never reaching 99%.

4.3 Summary

In this section, we presented an approach that uses probabilistic model checking to provide quantitative assurance for a system of coordinated autonomous robots. We demonstrated our approach by modeling and analysing a robotic demining scenario using the PRISM tool. Our results indicate that the approach

	cov							
	C0				**C1**			
N	A0	A1	A2	A3	A0	A1	A2	A3
1	18.089	19.827	70.525	88.362	18.089	19.827	70.525	88.362
2	34.168	36.841	83.646	93.997	35.794	39.022	93.573	99.057
3	45.613	48.455	88.558	95.903	52.427	56.595	98.881	99.931
4	54.998	57.728	91.495	96.997	67.037	71.392	99.813	99.984
5	61.538	64.077	93.184	97.613	78.799	82.646	99.945	99.987
6	66.028	68.388	94.218	97.984	87.380	90.316	99.960	99.987
7	69.289	71.497	94.916	98.233	93.029	94.996	99.962	99.987
8	72.788	74.812	95.620	98.483	96.388	97.559	99.962	99.987
9	74.631	76.551	95.975	98.608	98.195	98.825	99.962	99.987
10	78.522	80.204	96.691	98.859	99.079	99.392	99.962	99.987

Table 2: Results for *cov* with increasing N and coordination **C0** and **C1** and the following assignments to DET and DEF: A0 = (DET=LOW, DEF=LOW); A1 = (DET=LOW,DEF=HIGH); A2 = (DET=HIGH,DEF=LOW); A3 = (DET=HIGH,DEF=HIGH).

is promising. Our models are simple, and this enables us to avoid the statespace explosion problem. At the same time, the models are rich enough to provide quantitative feedback for judging the likelihood of mission success, as well as to make informed tradeoffs between mission configurations.

Our ongoing and future work is building on this work in several directions. One issue is that our models are based on a-priori probabilities (e.g., the probability of finding a mine in any cell). We assume that these probabilities are available with sufficient accuracy. Otherwise, the predictions made via probabilistic model checking will be correspondingly inaccurate. As part of our ongoing work, we are developing ways to estimate these probabilities via field experiments.

Another problem is the fidelity of our models, and the validity of our assumptions. For example, we assume that every robot has perfect ability to move between cells, and that every robot has identical capability. In practice, these assumptions may not hold. To address this issue, we are working on making our models richer, while at the same time avoiding the statespace explosion problem.

Finally, the work presented here is non-generative. It does not design an optimal mission given a set of mission parameters, and associated constraints. It would be interesting to look into whether probabilistic model checking techniques can be adapted to create a more generative approach that can handle an expressive range of mission configurations and constraints.

5 Validation Plan

Our validation plan is to test our models and the quantitative estimates associated with the modeled states and state transitions by constructing a variety of demining robotic missions. The test arena is 108” $\times$ 96”, with a printed “terrain

pattern" that can be changed with others, altered with printed overlays, and arranged with obstacles to simulate a variety of terrain conditions. For the purposes of testing our current set of probabilistic models, the terrain is a printed checkerboard pattern of 17×17 squares, each $5" \times 5"$. A checkerboard terrain pattern of $5" \times 4"$ provides buffer space to give the robots room to maneuver without colliding with the protective wooden wall that surrounds the arena. Each square of checkerboard pattern is designed to correspond to one cell of the PRISM model terrain pattern. The dimension of the cell corresponds to the configuration space around an individual robot. Mines are drawn in a square as colored particle clouds of varying densities to represent mines of diverse difficulties to detect and disarm. The best color saturation for a mine is in the center of the square. Navigation errors that cause the robot to drive over any part but the center of a square increase the likelihood of the robot not detecting the mine, and of possibly detonating it.

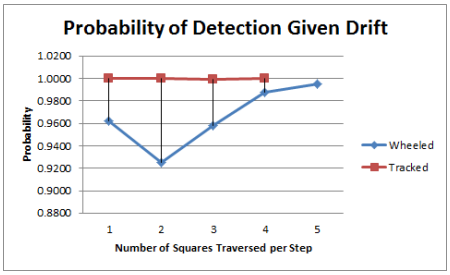
Three comodity Mindstorm NXT robots are configured for a coordinated demining mission. One robot has four wheels, a second has two and a skid, the third has two tracks. All have a color sensor in the center of their chassis, pointing downward, to detect mines, a forward-looking optical camera for recognizing landmarks and other robots, and a forward-looking range-finding IR sensor.

The robots were programmed to follow proposed scan patterns of the terrain so that their performance could be matched with the PRISM model and encoded to quantify the likelihood of the reproducibility of their behaviors. The two wheeled robots behaved alike and differed somewhat from the tracked robot, but all three deviated from their paths very quickly. This is because the mechanics and control of the robots is very simple — using odometer readings to dead reckon robot location. More sophisticated forms of dead reckoning are planned for the future based on [2], but whatever the sophistication of the robot, we know that the models must account for their imprecisions in whatever form they appear. A series of atomistic performance tests were run to characterize their performance, and are summarized in Fig. 6.

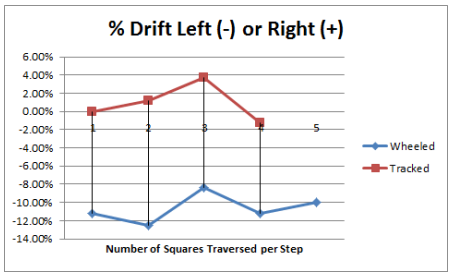
In the first set of characterizing experiments, see Fig. 6a, wheeled and tracked robots were compared on their ability to execute 90-degree turns. The wheeled robots demonstrated repeatable and consistent fidelity at turning three consecutive times without any error. On the fourth turn, a slight error was noticeable, which appeared to accumulate by the same amount — 3.5 degrees from completion — in each successive turn. That is, the fourth turn would be 86.5 degrees, fifth 83 degrees from a right angle, and so-on. The tracked vehicles, on the other hand, overshoot every turn by approximately 8.5 degrees (e.g. $98.5, 107, 115.5$, etc.). For the first missions, each robot must complete two 90-degree turns in order to move from one scan row to the other. This first evaluation leads one to understand that the tracked robots will be very poor performers in being able to execute their roles in any plan. Nonetheless, this experience revealed that our PRISM model must be updated to include performance measures of the robot in the mechanical execution of all aspects of its role — such as locomotion — and not just the application-specific aspects of its role, such as mine detection. If

90-Degree Turns	Vehicle Type	
	Wheeled	Tracked
Turns from start without drift:	3	0
Degrees of drift, each turn:	-3.5	8.5

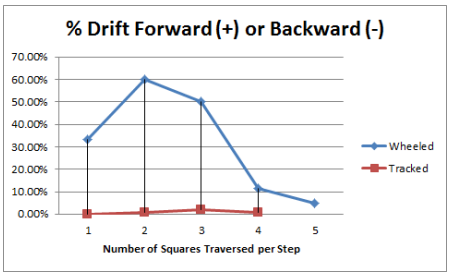
(a)



(b)



(c)



(d)

Fig. 6: Fig. 6a summarizes wheeled vs. tracked performance when turning 90-degrees. Fig. 6b illustrates the probability of detecting a mine due to navigational correctness when moving from one square to the next. Fig. 6c and Fig. 6d decompose that probability according to the tendency of the robot to drift left-right or over- or under-shoot the 5" distance of moving from one square to the next. A *step* corresponds to a "start" followed by the traversal of a variable number of squares, followed by a "stop". As the summaries indicate, performance sometimes varies according to the number of squares traversed in a step.

the robot cannot reliably navigate a scan path, its role in addressing the overall problem is similarly limited.

Fig. 6c and 6d summarize the tendencies of the robots to deviate from the controlled navigation of a path consisting of a row or column of squares. A *step* corresponds to a "start" followed by the traversal of a variable number of squares, followed by a "stop". The mechanical inertia of the tracked robots equalizes any differences between left and right traction motor start/stop conditions, rendering the tracked robots the more stable and reliable variety for traversing short distances during each step. Due to the large and variable contact area between the tracks and the smooth, flat terrain, vehicles with tracks tend to drift when traversing more squares during a movement step. Wheeled robots, however, are more sensitive to differences in drive motor timing, power, encoder errors, and sometimes to the "terrain effects" of the paper terrain pattern pillowing and bunching under the wheels. This causes their motion to vary more widely. Similarly, wheeled vehicles have more difficulty stopping and frequently roll beyond their targeted stopping position, whereas tracked robots "stop dead in their

tracks". The immediate impact of both forms of drift is that the robot scans cells unevenly, will begin to form holes in its search pattern and not approach the mine from an angle that will maximize its likelihood of detection. By not having a model to indicate the effects of navigation on performance, we crudely mapped such performance characteristics to the metric, *the probability of detecting a mine given motion drift by the robot.* This is summarized by Fig. 6b. The formula for calculating this metric is:

$$Pr(d) = 1 - |D_{lr} \times D_{fb}| \tag{3}$$

where

- $Pr(d)$ is the probability of detecting a mine given left-right and front-back drift in the robot's motion,
- D_{lr} is the probability of the robot to drift left (-) or right (+), and
- D_{fb} is the probability of the robot to pass (+) or fall short (-) of its target travel distance.

The estimates will be applied to the model that is used to calculate the probability of a mine detection in a given terrain square. Their effects are cumulative.

6 Conclusions

The contribution of this paper is to offer a strategy by which coordinated autonomous systems can be assured, tested and evaluated. We do this by adapting techniques from the reliability engineering discipline, which offers precise terminology and quantitative methods for evaluating reliable system performance. Reliability engineering is a mature discipline and is presently used for assessing complex engineered, mission-critical and safety-critical systems, but has not yet been applied to coordinated autonomy. We propose models by which reliability engineering can be so extended: a characterization of the range of autonomous coordination phenomena and the phases of a coordinated mission. We implemented two instances of those models as discrete time Markov chains (DTMCs) using a novel probabilistic model checking technology, PRISM. In an effort to validate our theoretical models, we began constructing robots and a test arena, and show how partial results can map to and be used by the probabilistic models. We demonstrate by these examples how this technique can be one of the anticipated and many new techniques that can be used to quantitatively assure coordinated autonomy.

Acknowledgment

The authors thank David S. Kyle and John F. Porter for their contributions to this research.

 This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003

with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. [3]

References

1. The role of autonomy in DoD systems. Task force report, Department of Defense Defense Science Board (July 2012)
2. Chong, K.S., Kleeman, L.: Accurate odometry and error modelling for a mobile robot. In: ICRA. vol. 4, pp. 2783–2788 (1997)
3. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge, MA (2000)
4. Fujita, M., McGeer, P.C., Yang, J.C.Y.: Multi-Terminal Binary Decision Diagrams: An Efficient Data Struct. for Matrix Repres. FMSD 10(2/3), 149–169 (04 1997)
5. Goodenough, J.B.: System of systems software assurance (2 November 2009), `www.sei.cmu.edu`, accessed 2013-02
6. Hansson, H., Jonsson, B.: A Logic for Reasoning about Time and Reliability. Formal Aspects of Computing (FACJ) 6(5), 512–535 (December 1994)
7. Headquarters, Department of the Army: Technical Manual No. 5-698-3 (2005), approved for public release: distribution is unlimited, accessed 2013-02
8. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: (CAV 2011). LNCS, vol. 6806, pp. 585–591. Springer (2011)
9. Scerri, P., Vincent, R., Mailler, R.: Coordination of Large-Scale MASs, chap. Comparing Three Approaches to Large-Scale Coordination, pp. 53–71. Springer (2006)
10. Stancliff, S.B.: Planning to Fail: Incorporating Reliability into Design and Mission Planning for Mobile Robots. PhD thesis, RI, CMU (2009)
11. Stancliff, S.B., Dolan, J.M., Trebi-Ollennu, A.: Towards a predictive model of mobile robot reliability. TR CMU-RI-TR-05-38, RI, CMU (August 2005)
12. Sycara, K., Giampapa, J.A., Langley, B.K., Paolucci, M.: The RETSINA MAS, a case study. In: SELMAS, vol. LNCS 2603, pp. 232–250. Springer-Verlag (2003)
13. Sycara, K., Lewis, M.: Integrating intelligent agents into human teams. Team Cognition: The Factors that Drive Process and Performance pp. 203–232 (2004)
14. Sycara, K., Paolucci, M., Velsen, M.V., Giampapa, J.A.: The RETSINA MAS infrastructure. JAAMAS 7(1), 29–48 (July 2003)