A Policy-Based Management Architecture for Mobile Collaborative Teams

Eskindir Asmare, Anandha Gopalan, Morris Sloman, Naranker Dulay and Emil Lupu
Department of Computing, Imperial College London, London, SW7 2RH, UK
Email: {e.asmare, a.gopalan, m.sloman, n.dulay, e.c.lupu}@imperial.ac.uk

Abstract—**Many missions are deemed dangerous or impractical to perform by humans, but can use collaborating, self-managing Unmanned Autonomous Vehicles (UAVs) which adapt their behaviour to current context, recover from component failure or optimise performance. This paper describes a policy-based distributed self-management framework for both individual and teams of UAVs. We use three levels of specifications — policy, mission class and mission instance to enable reuse of both policies and mission classes. The architecture has been tested on devices ranging from small laptops to body area networks. Initial evaluation shows the distributed architecture is scalable and outperforms a centralised mission management scheme.**

I. INTRODUCTION

The complexity of modern computing systems requires autonomic self-management [1], [2]. Self-managing, mobile autonomous systems are increasingly being used in hazardous environments for military applications, disaster management after earthquakes or floods to assist rescue operations by locating survivors. In this paper, we focus on self-management for collaborative teams of Unmanned Autonomous Vehicles (UAVs), i.e. mobile robots but the concepts have also been used in other application domains such as health care [3].

UAVs need to adapt their behaviour to current context – location, activity, available resources such as battery power and available services such as quality of (intermittent) wireless communication links. They should be self-managing in that they have to recover or adapt to component failures and optimise performance to best utilise available resources. Collaborating UAVs form a Self-Managed Cell (SMC) [3], the general autonomic computing [2] architectural principle we use for realising self-management of individual and teams of UAVs. A SMC in our scenario consists of one or more commanders which have the initial mission specification received from a command base and a team of UAVs with specific capabilities. An UAV may be composed of various sensors for vision, sound, vibration, chemical detection, location plus devices for communication and so provides specialised services for the mission. The mission specification defines how UAVs will be assigned to perform specific roles within the SMC, based on their credentials and capabilities, as well as when and how to adapt the mission to changes in context or failures. The adaptive management of UAVs is achieved by using policy-based techniques that allow dynamic modification of the management strategy relating to resources, task behaviour, communications and team management, without reloading the basic software within the UAV. SMCs can be combined in peer-to-peer or composition relationships to reflect complex collaborations between multiple teams to achieve an overall mission [4].

Although there has been research on control architectures for autonomous systems, the focus has largely been in organising intelligence [5]. We argue that if robots such as UAVs are to be used in real life applications then they should also be able to manage their intelligence. The focus of this paper is distributed self-management of a team of UAVs using the Ponder2 [6] policy framework – a generic object management system supporting dynamic loading, unloading, enabling and disabling of active managed objects capable of receiving action commands and performing actions. Ponder2 supports *obligation policies* (event-condition-action rules) to trigger specific actions to be performed when an event, such as the discovery of a new UAV occurs, or a UAV which is a member of the team fails as well as *authorisation policies* to specify conditions under which services and resources within a UAV can be accessed by other UAVs performing a specific role. Policies are interpreted, hence they can be dynamically loaded, enabled or disabled at run-time without shutting down a system in order to adapt the management strategy.

The rest of the paper is organised as follows. Section II presents the distributed mission management architecture. Section III details the mission specification while Section IV presents the implementation of the architecture. Section V describes the experiments and the ensuing results, and Section VI compares our approach with related work. Section VII gives conclusions and future work.

II. DISTRIBUTED MISSION MANAGEMENT

A mission for a team of heterogeneous robots is specified in terms of roles and role-missions. Mission specifications can be classified into (a) the domain they are targeted to – application specific [7] or generic [8], (b) the paradigm they use – plan based [9] or specification (configuration) based [8], and (c) the number of autonomic systems involved – single or multi-robot missions. A multi-robot mission could support homogeneous or heterogeneous robots and a heterogeneous-multi-robot mission could support static or dynamic task allocation. Based on the above classification, our mission specification is generic, specification-based, multi-robot and heterogeneous with dynamic task allocation.

To illustrate our approach, we consider an example mission to determine whether an area is safe for humans, with the

following main roles. *Commander (C)*: controls the mission and allocates UAVs to roles. *Surveyor (S)*: explores the area and builds a map. *Hazardous material detector (H). Communication relay (R)*: forms an ad-hoc communication network amongst the UAVs. *Aggregator (A)*: aggregates information from all UAVs to produce a map showing the detected hazardous materials.

A. Our Approach

A team of UAVs should execute a mission with a minimum number of UAVs with required capabilities although the configuration may not be optimal. When additional UAVs are available the team should expand to make use of the new UAVs, thereby increasing performance. Should there be a failure or departure of UAVs from the enlarged team, the team would contract but continue the mission. We define a *minimal team configuration* as the fewest types and number of UAVs needed to accomplish a mission. A *reasonably-optimal team* configuration has sufficient types and numbers of UAVs to achieve both redundancy and good performance. A mission starts execution when a team satisfying the minimal configuration can be formed but expands when additional UAVs join until it achieves the reasonably-optimal configuration. The reconnaissance scenario minimal configuration, is 1 C, 1 A, and 2 S, where S is the primary role given priority with respect to resource allocation and is also responsible for managing the other roles; and R and H are secondary roles. As shown in Fig. 1(a), S is collocated with roles R and H. R can be performed in parallel with either role S or H while the UAV has to switch between roles S and H as only one of these can be active at a time. Although R can run in parallel with these roles, it will potentially hinder the surveying or detection functions when trying to maximise communication link quality, so it should be placed in a separate UAV if available. Detecting hazardous objects is a much slower process than exploration, consequently the detection and surveying functions can be performed better by separate but cooperating UAVs which share information.

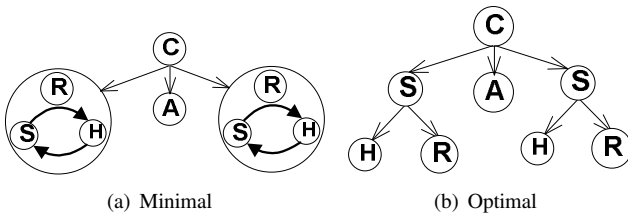(a) Minimal   (b) Optimal

Fig. 1.   Reconnaissance Mission Configuration

A reasonably-optimal mission configuration consists of 1 C, 2 S, 2 H, 2 R, and 1 A. The team started with the configuration shown in Fig. 1(a) and reached the configuration shown in Fig. 1(b) as new UAVs join the team in the mission area. The S roles, which assigned the R and H roles, serve as their commanders. Should any of the new UAVs fail or depart, the roles will revert to their minimal configuration position.

B. Conceptual Model

In this section, we present our model that uses the concepts of team-mission, role and role-mission that allow for distributed mission management of UAVs.

1) Team Mission (M): is a set of roles, each containing a set of policies that either governs the behaviour of the role or handles the assignment of UAVs to roles.

2) Role (R): is a placeholder containing specified role-missions (RM), authorisations (A) and tasks (T) which are loaded onto discovered UAVs that are assigned to the role. A role has an external and a local interface which provides a context for which role-mission policies can be specified.

2.A)External Interface: defines operations and events relating to interaction with external collaborating roles: (i) Management Operations for loading missions, policies etc. which are common to all roles, (ii) Provided Operations from the local interface, implemented by tasks in the role, that are made visible to and can be invoked by other roles in remote UAVs. (iii) Required Operations that are expected to be provided by collaborating roles. (iv) Outgoing Events generated by the tasks inside the role or propagated from the UAV components such as sensors, and published via an event bus for use by other roles. (v) Incoming Events generated by collaborating roles and required by this role, to trigger policies.

2.B) Local Interface: consists of (i) Events generated by the tasks within the UAV or propagated from UAV components such as sensors. These may trigger policies in the role-mission or map to the external interface. (ii) Operations implemented by the tasks within the UAV and invoked by local role-mission obligation policies.

3) Role-Mission (RM): a set of policies relating to a single role for controlling tasks and enabling/disabling other policies.

4) Authorisation Policies (A): specify how roles are permitted to interact with each other in terms of the events that can be triggered or operations that can be invoked via the external interface.

5) Tasks (T): are complex operations which the UAV can perform e.g. move from A to B, follow a path, track an object using video. Obligation policies in the mission may invoke operations supported by a task or activate a task. The tasks in a role are usually inherent to the type of the role and hence are specified inside the role class.

III. DISTRIBUTED MISSION SPECIFICATION

Fig. 2 shows our three levels of specifications: (a) Policies specified using Ponder2 [6] and stored in a policy repository, (b) the XML mission class specifies the types of roles needed for the mission and the management relation among the roles, and (c) the XML mission instance defines the mission parameters and role cardinalities required to instantiate a mission class. The policy specification in the repository may apply to multiple mission classes and there can be multiple instances of a mission instantiated with different parameters from a particular mission class. The policy repository is comparatively small so can be stored in the commander's memory and distributed to other UAVs as needed.
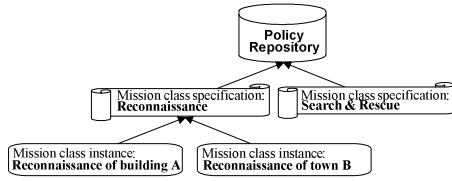
Fig. 2. Mission Specification Levels

A. Policy Specification

The policies specified for a role are broadly divided into *role assignment policies*, used to assign UAVs to roles based on their capabilities and *operational management policies* used by roles to manage their own or collaborating roles' operational behaviour. In the example role assignment policy shown in Fig. 3(a), the commander checks the capability of a newly discovered UAV, authenticates it and assigns it to the surveyor role, if it has the required capability. Fig. 3(b) shows an example re-assignment policy to deal with a failure.
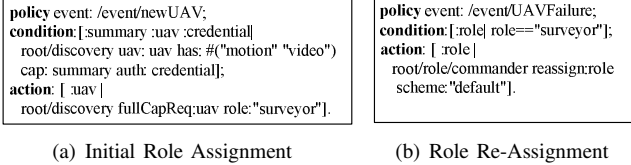
```
policy event: /event/newUAV;
condition:[:summary :uav :credential|
  root/discovery uav: uav has: #("motion" "video")
  cap: summary auth: credential];
action: [ :uav |
  root/discovery fullCapReq:uav role:"surveyor"].
```

```
policy event: /event/UAVFailure;
condition:[:role| role=="surveyor"];
action: [ :role |
  root/role/commander reassign:role
  scheme:"default"].
```

(a) Initial Role Assignment   (b) Role Re-Assignment

Fig. 3. Sample Ponder2 Policies

B. Mission Class Specification

A mission class specifies a team in terms of roles, policies a role uses to manage itself or other roles (where hierarchy exists) and indicates the management relation among the participating roles as well as the cardinality of each role. Fig. 4(a) applies to the reconnaissance scenario with a commander role managing a surveyor and an aggregator role. The surveyor role in-turn manages a hazard-detector and relay roles. The cardinality and other parameters are instantiated later. Mission parameters such as failure-timeout which are shared by all roles are also included. This specification can be used to instantiate different teams of the same configuration with different cardinalities, mission parameters and role behaviours using policies. The policy-based role behaviour specification allows for changing the behaviours of assigned roles.

C. Mission Class Instantiation

A mission class instance (which gives rise to the actual team of UAVs performing the mission) specifies values for cardinalities, mission parameters and URIs of policies which define the role behaviour as shown in Fig. 4(b).

D. Management Tree

As a means of decentralising discovery and role management, the UAVs in a mission are arranged in the form of a management tree, used for defining management hierarchies as well as data aggregation during execution of the mission.

(a) Mission Class Specification   (b) Mission Class Instance

Fig. 4. Mission Specification

Each UAV runs a tree formation algorithm which starts by broadcasting a discovery message. UAVs receiving this perform an authentication protocol (Section III-E), and reply with a summary of their capability description if they can be assigned to a role. UAVs already assigned to a role may ignore the discovery message. Upon authenticating and receiving the capability summary, the broadcaster decides whether to request a full capability description. The final decision of assigning the UAV to a role takes place after checking the full capability description against the requirements of the role. The broadcaster will be the parent of a UAV it assigns to a role and the assigned UAV will be listed as a child of the broadcaster. This tree is also used for management of communication link or UAV failures (elaborated in [10]). The initial management tree for the mission class specification in Fig. 4(a) is shown in Fig. 5(a).
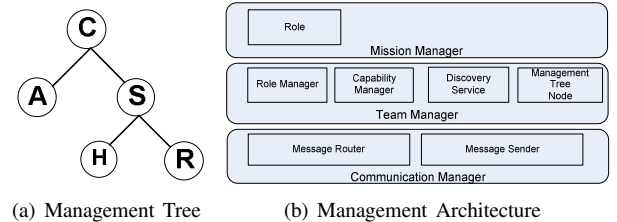

(a) Management Tree   (b) Management Architecture

Fig. 5. Management Tree and Architecture

E. Security of UAVs

The UAVs forming a team can change dynamically over time with new UAVs joining or leaving the team. These new

UAVs may also belong to other organisations (e.g. allies). Authenticating an UAV before it joins the mission and protecting the ensuing group communication is thus necessary to protect the mission [11].
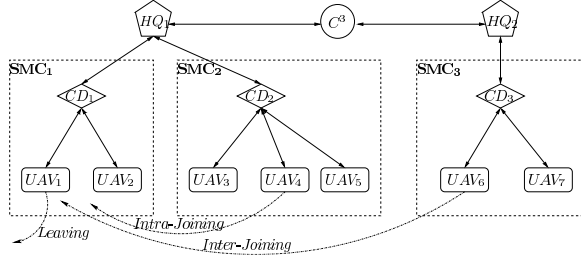

Fig. 6. UAVs belonging to different Organisations

Consider the scenario depicted in Fig. 6. UAV_1, UAV_2 and commander CD_1 form the Self Managed Cell SMC_1, while UAV_3, UAV_4, UAV_5 and commander CD_2 form SMC_2 and UAV_6, UAV_7 and commander CD_3 form SMC_3. SMC_1 and SMC_2 belong to the same organisation (HQ_1), while SMC_3 belongs to another organisation (HQ_2). Individual UAVs may join or leave a SMC team, e.g., UAV_4 and UAV_6 join SMC_1 during the mission, while UAV_1 leaves the SMC. UAV_4 joining SMC_1 is called *Intra-Joining* since both SMC_1 and SMC_2 belong to the same organisation, while UAV_6 joining SMC_1 is called *Inter-Joining* since SMC_3 belongs to a different organisational unit. We use the Certificate Public Key Infrastructure (C-PKI) [12] for authentication, confidentiality and message integrity among the UAVs. The coalition between the different organisations is achieved by using a single certification authority (C^3), which issues certified public/private keys to UAVs belonging to all organisations. During the course of the authentication, a common secret key is generated using the Diffie-Hellman protocol [13] between each role and its managing role, thus ensuring a secure communication channel. After authentication, new UAVs are assigned to specific roles and authorisation policies, in terms of the roles, control access to other roles and resources.

IV. IMPLEMENTATION

A. Mission management

The management architecture is implemented using the Java based Ponder2 policy toolkit with Tasks, Roles and Missions implemented as Ponder2 managed objects. An outline of the framework is shown in Fig. 5(b).

We use the UDP protocol for most messages but reliable delivery is implemented by the Message-Sender object for role assignment and capability description messages.

The Message-Router handles messages for multiple roles (and other objects) residing in an UAV. It registers roles to receive packets of a certain type and/or source and de-registers them on leaving. In the case when a role registers to receive packets of more than one type (or source) that intersect, the registrations are aggregated. A separate exclusion table is used to indicate a de-registered role which is part of an aggregated

registration. When a new packet arrives the dispatch and exclusion table are checked before the packet is passed to the registered roles.

The Role-Manager object is responsible for loading and withdrawing/loading a role in mission startup and reconfiguration respectively.

B. Capabilities

The capability of an UAV is the set of operations which the software and hardware in the UAV support as well as the events it generates. It depends on the current set of software tasks loaded into the UAV. Tasks support reflection so they can be queried for their interface description by the Capability module to dynamically produce the capability description.

C. Proof of Concept Demonstration

The distributed mission management architecture detailed in Section II was implemented on the Koala robots [14] which have 16 infrared proximity sensors and a camera. Each robot is controlled by software [15] running on an Asus EEE PC through a USB to serial cable. The scenario was a search and rescue mission of a wounded soldier who has a wearable computer (Asus EEE) and a body sensor network to determine his medical condition. A laptop was used for the Commander with all communication via an ad-hoc wireless network.

Fig. 7. Snapshot of Proof of Concept Demonstration

The steps are as follows: (i) Soldier is wounded in the battlefield; (ii) Wearable computer sends a distress signal to the base. (iii) Commander assembles the mission for assistance, comprising unmanned vehicles capable of navigation, communication, surveillance and hazard detection with surveyor (including hazard detector) assigned to one UAV and aggregator to another. (iv) The surveyor moves towards the soldier, but detects a hazard along the way causing it to fail. (v) On detecting the failure (through timeouts), the surveyor role is re-assigned to the aggregator by the commander. Also, the last position of the previous surveyor is relayed so that the new surveyor can avoid the "hazard". (vi) The new surveyor avoids the hazard using the information provided and reaches the soldier to deliver assistance as necessary. This proof of concept demonstration was shown as part of the Annual Conference of the ITA, 2008 [16]. A snapshot of the demo is shown in Fig. 7. The first surveyor robot stops on detecting the "hazard"

(the cylinder in the centre). The second surveyor (which previously was an aggregator before role re-assignment) avoids the obstacle and reaches the soldier (top right corner).

V. Experiments and Results

A. Experimental Setup

The distributed mission management architecture was implemented using the Java based Ponder2 policy toolkit (Section IV) which enabled experimentation on a testbed of generic Linux machines running Java to compare our distributed management versus a centralised scheme and the effect of the depth of the management tree. At the beginning of the simulation the number and types of roles are changed in the mission class specification and the requisite number of SMCs is started on various machines.

B. Results

In the first experiment the depth of the tree is set to *4*. We measure the initialisation time for the mission (which includes the UAV discovery, role assignment, loading policies from the repository and starting roles), as shown in Fig. 8. This indicates the centralised scheme works best when the number of roles is relatively small but as the number increases, the distributed scheme significantly outperforms the centralised scheme. Note that the time for mission initialisation does not increase very much, even for large numbers of roles which shows that the architecture is scalable.

Although the scenario described in the paper uses few roles, a more realistic scenario would require far more roles each with different behaviours e.g. for UAVs with specific types of sensors, for airborne UAVs to support the mission, for soldiers and medics who may also be part of the mission, as well as their support vehicles.
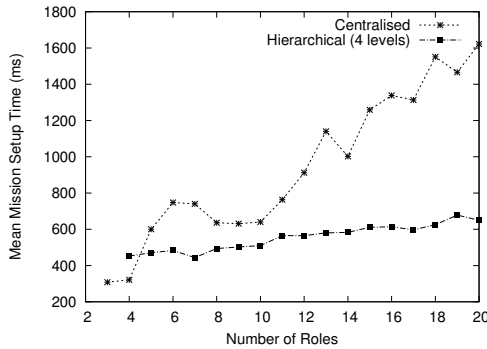

Fig. 8.   Mission Setup Time vs. Number of Roles

In the second experiment we fixed the number of roles to 210 and varied the depth of the management tree between 1 (centralised) and 10. We then measured the time taken to initialise the mission assuming that UAVs with all required capabilities are available at startup. Fig. 9 shows that the mission initialisation time decreases as we increase the depth of the tree as a result of load balancing. However it increases as the tree becomes very deep due to the delay in role assignment created by an increase in the number of hops. This

suggests the existence of a ratio of number of roles to depth which guarantees a minimal mission setup time for a given management tree. The increase in the mission initialisation time between the first and the second experiment was due to the C-PKI based mutual authentication performed by each pair of UAVs that are involved in a role assignment process, as security was not included in the first experiment.
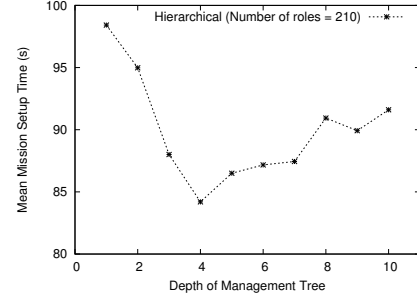

Fig. 9.   Mission Setup Time vs. Depth

VI. Related Work

Specification of the organisation of a set of primitives to obtain a sophisticated mission to perform complex tasks is presented in [8]. This includes a Configuration Description Language (CDL) to configure a single or group of robots by defining reusable assemblage agents for different tasks and instantiate the primitives. The MissionLab development environment supports graphical design of a mission using CDL, compilation and loading executable code onto to a robot or a simulator. However, the finite state machine based approach used to describe a mission is suitable only for low level components such as tasks and it is not easy to specify multi-robot missions with many participants.

A case based reasoning approach for generating mission plans [17] builds on the work in [8] and uses a Contract Net Protocol [18] based task allocation while [18] presents the Contract Net Protocol for distributing tasks through negotiation. Each node in the net takes either a manager or contractor role. Managers announce tasks, potential contractors submit bids to the managers, the managers then evaluate the bids and award contracts to the bidders. The problem-domain dependent contents of negotiation messages are specified by users.

An approach for coordination of robots based on dynamic role assignment is given in [19]. This has a layered architecture with a coordination protocol, based on utility functions defined for each role, using a publish-subscribe communication protocol. The robot with the highest role utility value is assigned to the role. Formation is selected using a voting system. Compared to other approaches (e.g., [20]), which tie the robot control architecture to the coordination architecture (mechanism), this is more general in that robots with different control architectures can coordinate.

A paradigm for cooperating robots is presented in [21] in which hybrid automata are used to represent roles, role assignments and discrete variables related to each robot. The

composition of these automata is used to model the execution of cooperative tasks. A role is defined as a function, one or more robots perform during the execution of a cooperative task and utility functions are used to decide when to change roles.

Likhachev *et al.* [22] have proposed an approach to automatic modification of behavioural assemblage parameters for autonomous navigation tasks using case based reasoning. We try to solve a similar but more generic problem using policies.

In [23], a general framework, called MURDOCH, is presented for inter-robot publish-subscribe communication and dynamic task allocation for cooperation. MURDOCH offers a distributed approximation to a global optimum of resource usage, but due to the completely distributed task assignment scheme, it suffers from the same problems as greedy algorithms – equivalent to an instantaneous greedy scheduler, where decisions are made based on only the current and/or local situation without taking into account how the decision might affect the future and/or the global situation. These algorithms may not always give the best solution. Our approach allows for optimisation on the set of discovered UAVs in the role-assignment time window. Due to the fact that a manager role is aware of future roles to be assigned by itself or its managed roles (from the mission specification), it can, to some extent, take the future/global situation into account when making the assignment decision.

In [24], a distributed constraint programming solution for assigning tasks to robots is given. This tries to minimise remote task dependencies by creating a task dependency graph, called a distributed organisational task network (DOTN), and searches for minimal dependency solutions at run time and by trading tasks using the SOLO algorithm for task reallocation.

In [25], a policy-based community specification or *doctrine* defines the roles of the participants in an ad-hoc network community, the characteristics that participants must exhibit in order to be eligible to play a role, as well as the policies governing their behaviour within the community. A *doctrine* focuses on security but we include security as well as mission specification and task allocation.

VII. CONCLUSION

We have presented models, concepts and implementation details of a distributed policy-based management architecture for mobile collaborative teams. The three levels of specifications, namely, policy, mission-class and mission-instance enables flexibility and reuse of specification in different scenarios. A proof of concept demonstration incorporating robots, body sensor nodes and portable computers showed the flexibility of the approach. Future work will mainly focus on semantic based capability matching to improve the role assignment process and configuration planning to attain more optimal team configurations.

ACKNOWLEDGEMENTS

REFERENCES

[1] P. Horn, "Autonomic computing: IBM's perspective on the state of information technology," *IBM TJ Watson Labs, NY*, 2001.

[2] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[3] E. Lupu *et al.*, "AMUSE: Autonomic management of ubiquitous e-health systems," *Concurrency and Computation: Practice & Experience*, vol. 20, no. 3, pp. 277–295, 2008.

[4] A. Schaeffer-Filho *et al.*, "Towards supporting interactions between self-managed cells," in *Proceedings of the First IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, Boston, MA, USA, 2007, pp. 224–233.

[5] R. R. Murphy, *Introduction to AI Robotics*. Cambridge, MA, USA: MIT Press, 2000.

[6] (2008) Ponder2. [Online]. Available: http://ponder2.net

[7] I. Roman-Ballesteros and C. Pfeiffer, "A framework for cooperative multi-robot surveillance tasks," *Electronics, Robotics and Automotive Mechanics Conference, 2006*, vol. 2, pp. 163–170, Sept. 2006.

[8] D. MacKenzie, R. Arkin, and J. Cameron, "Multiagent mission specification and execution," *Autonomous Robots*, vol. 4, no. 1, pp. 29–52, 1997.

[9] I. Rachid Alami and S. S. da Costa Bothelho, "Plan-based multi-robot cooperation," *Advances in Plan-Based Control of Robotic Agents*, vol. 2466, pp. 65–95, Sept. 2002.

[10] E. Asmare, A. Gopalan, M. Sloman, N. Dulay, and E. Lupu, "Adaptive self-management of teams of autonomous vehicles," in *Proceedings of the 6th ACM International Workshop on Middleware for Pervasive and Ad-hoc Computing*, Leuven, Belgium, 2008, pp. 1–6.

[11] E. Asmare, N. Dulay, A. Gopalan, E. Lupu, and M. Sloman, "Secure distributed self management framework for UXVs," in *Systems Engineering for Autonomous Systems Defence Technology Centre Conference*, Edinburgh, UK, June 2008.

[12] R. Housley *et al.*, "Internet X. 509 Public Key Infrastructure Certificate and CRL Profile," RFC 2459, January, Tech. Rep., 1999.

[13] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[14] (2008) k-team. [Online]. Available: http://www.k-team.com

[15] D. Sykes, W. Heaven, J. Magee, and J. Kramer, "From goals to components: a combined approach to self-management," in *Proceedings of the 2008 ACM/IEEE International Workshop on Software Engineering for Adaptive and Self-managing Systems*, Leipzig, Germany, 2008, pp. 1–8.

[16] (2008) ITA. [Online]. Available: http://www.usukita.org/

[17] P. Ulam, Y. Endo, A. Wagner, and R. Arkin, "Integrated mission specification and task allocation for robot teams - design and implementation," *IEEE International Conference on Robotics and Automation*, pp. 4428–4435, April 2007.

[18] R. Smith, "The Contract Net Protocol: High-level communication and control in a distributed problem solver," *IEEE Transactions on Computers*, vol. C-29, no. 12, 1980.

[19] L. Iocchi, D. Nardi, M. Piaggio, and A. Sgorbissa, "Distributed coordination in heterogeneous multi-robot systems," *Autonomous Robots*, vol. 15, no. 2, pp. 155–168, 2003.

[20] L. Parker, "ALLIANCE: An architecture for fault tolerant multirobot cooperation," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 220–240, 1998.

[21] L. Chaimowicz, V. Kumar, and M. Campos, "A paradigm for dynamic coordination of multiple robots," *Autonomous Robots*, vol. 17, no. 1, pp. 7–21, 2004.

[22] M. Likhachev, M. Kaess, Z. Kira, and R. C. Arkin, "Spatio-temporal case-based reasoning for efficient reactive robot navigation," *Mobile Robot Labaratory, Georgia Institute of Technology*, 2005.

[23] B. P. Gerkey and M. J. Mataric, "Principled communication for dynamic multi-robot task allocation," in *Proceedings of the 7th International Symposium on Experimental Robotics*, Honolulu, HI, USA, 2001, pp. 353–362.

[24] B. Salemi and W.-M. Shen, "Distributed and dynamic task reallocation in robot organizations," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 1, 2002.

[25] S. L. Keoh, E. Lupu, and M. Sloman, "PEACE: A policy-based establishment of ad-hoc communities," in *Proceedings of the 20th IEEE Annual Computer Security Applications Conference*, Tucson, AZ, USA, 2004, pp. 386–395.