SECURE DYNAMIC COMMUNITY ESTABLISHMENT IN COALITIONS

Eskindir Asmare, Naranker Dulay, Emil Lupu, Morris Sloman
Imperial College London
London, UK

Seraphin Calo, Jorge Lobo
IBM Research
Hawthorne, NY

ABSTRACT

This paper presents a mission management architecture and a policy-based methodology for specifying and assigning missions to be accomplished by a secure Community of Interest (CoI). By using policies and roles to specify missions we enable dynamic mission assignment and adaptation. The specific entities forming the community may not be known in advance, so they have to be discovered, authenticated and assigned to roles. Policies are needed to define what services and resources a role can use from another role; what information can be exchanged, and the protocols to be used for communication; and, how to deal with heterogeneity of communications, security mechanisms, etc.

INTRODUCTION

Coalitions that come together in communities of interest (CoIs) with common goals, perhaps only for a short period, require that interoperation between groups with different security policies be negotiated and that security policy decisions are made on-line in real-time. A community is specified in terms of roles and policies that identify the tasks which entities assigned to the role will perform, how the role will interact with other roles in terms of types of security mechanisms and protocols used, what services or resources it will make available to other roles, etc. Each role specifies the capabilities it expects from any entity that is assigned to it.

The approach will build upon previous work at Imperial College on Self-Managed Cells (SMCs) which can be adapted to support the types of communities described. A SMC is an architectural pattern for building ubiquitous computing applications and consists of a set of hardware and software components that form an administrative domain that is able to work autonomously. SMCs implement a policy-driven feedback control-loop which enables them to perform management and reconfiguration actions in response to events of interest, such as changes of context or changes of state in any resource. They can be combined in various ways to capture different architectural patterns.

In this paper we describe a distributed event-based self management architecture for dynamic CoIs which makes use of policies to define adaptive strategies for both individual elements and groups of cooperating elements. First we give an overview of the Self-Managed Cell concepts and policy-based management, then we present our policy-based mission management framework. Finally we conclude by summarizing the paper and indicating directions for future work.

BACKGROUND

A Self-Managed Cell (SMC) [7] is an architectural pattern for combining the services and components needed for self-management of a set of components ranging from a body-area network to large-scale distributed applications in which multiple devices collaborate to achieve a common goal. The SMC provides an extensible framework for supporting self-organization, self-healing, self-protection and self-optimization.

Figure 1: Architecture of an SMC

The management services in an SMC interact with each other via an asynchronous event bus. As a minimum an SMC should contain a policy service to implement feedback loop adaptation, a discovery service to discover the components which form the cell and a publish-subscribe event service to implement the event bus. The basic architecture of an SMC is shown in Figure 1 and the core services are briefly described in the following.

The Policy Service is the means of specifying the adaptive behavior of an SMC in terms of obligation policies or event-condition-action rules, as well as authorization policies to control what services and resources can be used by specific entities. The Discovery Service discovers components which are in range and capable of being members of the SMC.

The Event Bus is responsible for asynchronous notification of events to different management services of the SMC. Event notification is a crucial element because adaptation, protection and other self-management actions are specified in terms of obligation policies triggered by events. An obligation policy may perform an action which modifies the behavior of a single component of an SMC or it may enable or disable other policies to change the overall behavior of the SMC.

The use of an event bus enables a loose coupling between the various SMC services which can react concurrently and independently to event notifications. However all communication is not constrained to be over the event bus and simple messaging or object invocation may be used as appropriate.

The Measurement and Monitoring Service is responsible for keeping track of the SMC's operation and generates events when actions need to be taken. Events may indicate situations such as degradation or failure of components. The Context Service uses sensors to determine information such as current location, weather conditions, or what obstacles are in the vicinity as the SMC behavior will adapt to current context.

A group of cooperating SMCs may compose or federate to form a single self managing system (SMC) in order to collaborate to achieve a particular mission. An SMC is an autonomic manager (AM) [9], and four relationship types have been identified to capture interactions between autonomic managers.

The *ManagedBy* relationship exists when an autonomic manager manages a resource, or when one manager manages another manager. Such a relationship should always exist for managed resources.

ManageTogether indicates that two or more autonomic managers have complementary functions that have been composed together. The composition is accomplished by exchanging appropriate information.

SupplyService is used when one autonomic manager provides a service for another autonomic manager. Both AMs must have a common understanding of the service, but need know little else about each other.

A *Co-Manage* relationship exists when autonomic managers with similar functions collaborate to manage the same set of manageable resources. There is the potential for conflict with this type of relationship, so there must be mechanisms for detecting and resolving potential conflicts.

Figure 2: AM Interactions

The SMCs that are part of a larger SMC may retain their identities or cease to interact outside the larger unit. In the latter case, the nested SMCs would obtain any necessary services through the auspices of the externally visible SMCs.

POLICY-BASED MISSION ARCHITECTURE

Policies are rules defining choices in behavior. We use the Ponder policy framework in our mission management architecture. Ponder [3] [4], is an object-oriented policy-specification language developed at Imperial College London. It provides a common language which enables administrators to specify both Access Control and Obligation Policies for security and service management. Ponder2 [1] is the latest version of the Ponder policy framework. It is a generic object management system which allows the dynamic loading, unloading, enabling and disabling of managed objects. Generally a Ponder2 managed-object is an active object that is capable of receiving action commands and performing actions. Ponder2 is comprised of three components, a domain service which is very similar to a

directory service, an obligation policy interpreter and a command interpreter. The domain service provides a hierarchical structure for managing objects. The obligation policy interpreter interprets event-condition-action rules. The command interpreter accepts and interprets a set of commands, directed to managed-objects in the domain structure, in the form of XML documents.

Policies are interpreted; as a result they can be dynamically loaded, enabled or disabled at run-time without shutting down a system in order to adapt the management strategy being used within an SMC. Thus policy-based management enables flexible and adaptive automation of system self management. [2] [8] [9]

We use obligation policies (event-condition-action rules) to trigger specific tasks to be performed when an event such as a component failure occurs or a sensor detects an object that requires further investigation. Authorization policies specify what services and resources within an SMC can be accessed by other SMCs performing a specific role and under what conditions.

A *mission* is a set of sequential or concurrent tasks which must be performed in order to achieve an overall goal. A planning process such as that described in [10] or [11] is needed to generate the tasks from the goal. The tasks may also be sub-missions as we allow for nested missions. The planner may generate more than one strategy for achieving a goal or the context may change such that the strategy for achieving the goal has to adapt to the current situation. The implication of this is that the CoI mission execution architecture should allow adaptation of missions. We are using a policy based approach for developing a self-managing mission framework.

Arkin et al. [6] define mission, in the context of robots, as a composition of tasks, and mission specification as the process in which step-by-step instructions are generated to guide one or more robots to accomplish a set of tasks. We use a similar notion of mission. Hence, a mission, in our system, is either a simple set of tasks or a composition of sub-missions which could themselves be compositions of other sub-missions. As indicated above, a mission can be hierarchically structured. We use the term mission-specification to refer to the overall specification of policies and roles relating to a group of cooperating elements which perform the roles within a mission.

A *role* is a placeholder for an entity that can be discovered and assigned to the role if it has the required credentials and capabilities to support the tasks associated with the role. The role actually holds missions, tasks and policies which can be downloaded into the discovered entity to govern how it interacts with other roles within the mission. A role's *external interface* defines how it interacts with other roles – operations which can be invoked on it by other roles (exported interface), operations it may invoke on other roles (imported interfaces), and events it generates as well as incoming events. The role's *local interface* consists of the operations and events provided by the tasks and services that execute within the entity itself. For example, the tasks for an unmanned vehicle could be simple motion tasks, and services could relate to sensors such as video cameras.

RECONNAISSANCE SCENARIO

Our approach can be illustrated using a simple reconnaissance scenario involving unmanned autonomous vehicles (UAVs) performing a reconnaissance mission to collect data regarding the layout and contents of a house to determine whether it is safe for humans to move into the house or not. We can identify the following roles in this scenario.

Commander: this is typically a manned vehicle with a range of communications equipment. It is responsible for managing the mission.

Surveyors: UAVs which collect data (for instance video) while exploring the house. They send the collected data to the aggregator which relays it to the commander.

Relays: relays extend the wireless communication range of a UAV by serving as a repeater.

Aggregator: UAV which produces a map of the whole space and distributes it back to the surveyors so that they can use it for the remaining reconnaissance. It will also send all the aggregated information to the commander.

We assume the following high level mission specification: There will be one commander responsible for assigning roles to UAVs, two surveyors for exploring and recording/streaming a video during exploration, and one aggregator for aggregating the information collected by surveyors. In the event of a communication failure between the commander and the aggregator or between the aggregator and the surveyors, at most one surveyor should leave the surveyor role and assume a relay role to maintain communication. A UAV might assume multiple roles provided that the roles do not have

conflicting goals. For instance, the movement of a relay UAV might be controlled by the wireless communication signal strength between itself and the other UAVs to which it is serving as repeater, whereas the movement of a surveyor UAV might be random or guided by a map. These patterns of movement can potentially conflict with the goal of a relay UAV.

The domain structure holding information downloaded to a UAV which has been allocated to a surveyor role is shown in Figure 3.

Figure 3: UAV Role Domain Structure

A role, in our model, has at least two interfaces called external and local, and a capability requirement. The external interface is comprised of:

- Management operations which are common to all roles.
- Operations from the local interface that have been allowed to be visible (or accessible). This control of visibility (or accessibility) may be static or dynamic by using authorization policies.
- Events generated by the role. The events may be generated by the tasks inside the role or bubbled up from the UAV components such as sensors.

- Operations which are required by the role. These operations are expected to be provided by collaborating roles.
- Events consumed by the role. These events are expected to be generated by collaborating roles.

The local interface is comprised of operations and events provided by tasks. The operations in this interface are used by the role's missions. A task may allow an event to bubble up from its subtask to the local interface. For modularity, operations from subtasks are not visible to the local interface.

The capability requirement consists of the operations and events which must be provided by tasks and the UAV hardware in order to support the role these operations and events would be provided by:

- The basic UAV control modules such as sensors, motion control etc.
- Basic perception and control tasks in the UAV which are downloaded to support the role. Some of these may already have been loaded for previous role.

When a role is loaded to a UAV its capability requirements should be checked against the UAV's provided-capability. Any required tasks, which are not available, would have to be downloaded to the UAV.

The provided capabilities of a UAV indicate its potential in terms of the basic functionality that is expected by the tasks and policies which are loaded onto the UAV. It is analogous to a virtual machine interface available at the policy/task programming level, whereas the role of a UAV defines a specific behavior within the Mission-SMC in terms of the tasks to be performed and how the role interacts with other roles.

A role can be suspended, removed and new roles downloaded at runtime so roles are very dynamic; however, the capabilities of a UAV, often relate to specific sensors or devices on the UAV, which do not change frequently, so most of the capability is comparatively static, other than properties such as available power, memory, or spare processing capacity.

A role definition specifies the capabilities required to support the role and so only a UAV with the required capabilities should be assigned to that role. Typically assignment to roles takes place when a UAV is discovered, or if a UAV fails another may need to be reassigned to take over, assuming the role mission can

be loaded onto it by the commander or from a nearby UAV which has the role mission specification.

A team of UAVs form a Mission-SMC to cooperate to accomplish the mission with each UAV being assigned to one or more roles within the Mission-SMC. We assume that the Mission-SMC specification is preloaded onto a command vehicle or one of the UAVs which has the capabilities to perform the commander role. Figure 4 shows the main roles for the reconnaissance scenario Mission-SMC. The commander role is responsible for assigning other UAVs to appropriate roles as defined by role assignment policies specified within the Mission-SMC specification, as previously explained.

Figure 4: The Mission SMC and Roles

The assumption is that the UAVs to be assigned to surveyor and aggregator roles may be discovered as they come into radio range of the commander during the course of the mission. A discovered UAV would first have to be authenticated and its credentials (certificates) verified in order to ensure that the UAV belongs to an allied force. The UAV will then be requested for its capability description and based on its capabilities it will be assigned to a role. For instance, in the reconnaissance scenario, if a UAV has the capabilities motion, video and communication relay, it would be assigned to a surveyor role, and if it has the required processing and storage it would be assigned to an aggregator role.

The policies shown in Figure 5 specify how a newly discovered UAV can be assigned to the appropriate role, after the credentials have been successfully verified, based on the UAV's capabilities. The discovered event provides UAV address, credentials, and resources (i.e. provided capabilities) as event parameters which are used within the policies. Encoding the role assignment as policies enables us to change the strategy of this

assignment during the mission without interrupting its functioning.

```

1. oblig on discovered ( UAV, credentials, resources)
   do /smc/roles/commander.assign(UAV)
   when authenticate (credentials)
       and resources.comms = "longRange"

2. oblig on discovered ( UAV, credentials, resources)
   do /smc/roles/surveyor.assign(UAV)
   when authenticate (credentials)
       and hasCapabilities(motion, video, relay)

```

Figure 5 : Role Assignment Policies

We specify the overall mission in terms of roles using policies in order to enable mission adaptation by utilizing the dynamic nature of policies. The overall mission specification consists of a shared knowledge base which might contain certificates, overall-mission constraints, etc. For each role type, a role assignment policy defines the required capabilities and credentials. Assignment of the UAV to a role results in missions and tasks related to the role being downloaded to the UAV if they are not already present.

An outline of the overall mission specification is illustrated in Figure 6. In our scenario, the commander vehicle would have the overall mission preloaded which it would interpret, i.e., it is assigned to the commander role. When the surveyor receives an *Enabled* event the policy is triggered and enables both the explore task and the video-record function. The Surveyor periodically receives *battery level* events and if the level is less than 25 it activates the return to base task. When it receives a *comms failure* event, it loads the relay role from the commander and indicates that it is capable of performing the relay function. The commander chooses one of the surveyors to perform the relay function and notifies it via an event which causes the UAV to disable the surveyor role and enable the relay role. The authorization policies indicate what operations the aggregator and the commander are permitted to perform on the UAV.

Figure 6: Example Mission-SMC Specification

CONCLUSIONS

In this paper we have presented a policy based mission specification scheme and mission management architecture. These elements will enable systems to come together dynamically to establish communities for the

purpose of accomplishing a mission. The roles, policies, and mission specification have been illustrated using a simple reconnaissance scenario. Much more complex missions can be similarly expressed. By using policies and roles to specify missions we enable flexible mission assignment and adaptation.

The current system described in this paper assumes a single commander which executes the overall mission specification, and discovers and assigns UAVs to roles. This could be a single point of failure and the commander may not always be in communication range of new entities which could potentially join a mission. We are working on a distributed overall mission whereby entities already in the mission can discover and allocate new entities to roles for which they have mission specifications. We will also investigate specific application roles, e.g., relating to managing the motion of a UAV in order for it to perform a relay function within the ad-hoc network and allocating entities to specific security management roles to support more flexible and efficient security management functions, such as key distribution, authentication, etc. Methods for containing the complexity of the overall formulation will be investigated. Of particular interest are analysis tools for maintaining consistency of policies and roles.

ACKNOWLEDGEMENTS

The work reported in this paper was partly funded by the Systems Engineering for Autonomous Systems (SEAS) Defence Technology Centre established by the UK Ministry of Defence. This research is continuing through participation in the International Technology Alliance sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence.

REFERENCES

- [1] *Ponder2*. <http://ponder2.net/>.
- [2] M. Dam, G. Karlsson, B. S. Firozabadi, and R. Stadler. *A Research Agenda for Distributed Policy-based Management*. Report, The Royal Institute of Technology (KTH), 2002.
- [3] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. *The Ponder Policy Specification Language*. In Proceedings of Policy 2001, Workshop on Policies for Distributed Systems and Networks, pages pp. 18–39. Springer-Verlag LNCS, 2001.
- [4] N. C. Damianou. *A Policy Framework for Management of Distributed Systems*. phd thesis, Imperial College of Science, Technology and Medicine, February 2002.
- [5] E. Lupu, N. Dulay, J. Sventek, S. Heeps, S. Strowes, K. Twidle, S.-L. Keoh, and A. Schaffer-Filho. *Amuse: Autonomic Management of Ubiquitous e-Health Systems*. To be published in Concurrency and Computation: Practice and Experience, 2007.
- [6] D. C. MacKenzie, R. C. Arkin, and J. M. Cameron. *Multiagent Mission Specification and Execution*. Auton. Robots, 4(1):pp. 29–52, 1997.
- [7] J. Sventek, N. Badr, N. Dulay, S. Heeps, E. Lupu, and M. Sloman. *Self-managed Cells and their Federation*. In Workshop Proceedings of the 17th Conference on Advanced Information Systems Engineering. Springer-Verlag LNCS, 2005.
- [8] D. C. Verma. *Simplifying Network Administration using Policy-based Management*. IEEE Network, 16(2):pp. 20–26, Mar/Apr 2002.
- [9] S. R. White, J. E. Hanson, I. Whalley, D. M. chess, and J. O. Kephart. *An Architectural Approach to Autonomic Computing*. In Proceedings of the International Conference on Autonomic Computing, pp. 2–9. IEEE Computer Society, May 2004.
- [10] K. Erol, J. Hendler, and D. S. Nau. HTN Planning: Complexity and Expressivity. In Proc. of AAAI-94, pp. 1123–1228, 1994.
- [11] Sardina, S., de Silva, L., and Padgham, L. 2006. *Hierarchical Planning in BDI Agent Programming Languages: a formal approach*. In Proceedings of Fifth international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06), Hakodate, Japan, May 8 - 12, 2006, ACM Press.