Proceedings of the 8th IEEE Workshop on Policies for Distributed Systems and Networks, Bologna, Italy, June 2007, *(invited abstract and keynote)*.

Autonomous Pervasive Systems and the Policy Challenges of a Small World!

Emil Lupu[1], Naranker Dulay[1], Joe Sventek[2], Morris Sloman[1]
[1]*Department of Computing, Imperial College London*
[2]*Department of Computing Science, University of Glasgow*
{e.c.lupu, n.dulay, m.sloman}@imperial.ac.uk, joe@dcs.gla.ac.uk

Abstract

Pervasive systems are the subject of intensifying research efforts and their applications range from health monitoring and intelligent homes, to location aware services, unmanned vehicles and city-wide pervasive infrastructures. Although application-specific solutions have been proposed, their design has often raised additional challenges. This paper discusses the use of Autonomous Pervasive Systems as a fertile testbed for policy-based adaptation and for integrating techniques that span across conventional subject boundaries. Additionally, we present the Self-Managed Cell architectural pattern for realizing policy-driven Autonomous Pervasive Systems and discuss the design of the Ponder2 policy service.

1. From Distributed to Pervasive Systems

Policy-based management, including access control and security management aspects has, in the past, often focused on large-scale systems that constitute an enterprise network or span beyond the borders of the enterprise [1]. The emphasis on large-scale systems is justified because policies can be applied to large sets of managed resources in order to configure them uniformly and adapt their behaviour to changes in business requirements. The scale, complexity and distribution of the systems involved presents significant challenges and significant progress has been made to address them. In such systems, policy-based management is often based on centralised specification and deployment of policies by system administrators. There is little tool support for specifying policies or detecting conflicts. Policy-based management is often laborious, the benefits obtained are difficult to quantify in monetary terms and its deployment requires both effective standardisation across equipment vendors and significant upfront investment so it has not been widely adopted. Policy analysis and refinement techniques would provide significant advantages but substantial work remains to be done towards practically evaluating the first and achieving the latter.

Pervasive Systems refer to a world where computational devices are embedded in the environment, can be worn by users or even implanted in their bodies. Applications in this area range from body sensor networks for health monitoring [2], intelligent buildings and home automation [3], autonomous vehicles [4], and urban planning [5]; such applications require continuous adaptation at local device level as well as for collections of devices. The need for adaptation is driven not only by requirements changes but also by user mobility and context. Such devices have limited computational capabilities and strict power consumption requirements. Their operation must therefore be optimised and must constantly adapt in order to minimise resource consumption. Users are by and large not technically knowledgeable, and user interaction must be minimised in order to avoid disturbance. There are no system administrators to effect configuration changes or to resolve errors; such systems must therefore be autonomous. Policy-based approaches are particularly suited to realising Autonomous Pervasive Systems as they offer a simple, flexible and dynamic technique for implementing adaptation and feed-back control. We refer primarily to *obligation* policies in the form of event-condition-action rules, although other forms discussed in the paper are also desirable. However, many existing policy-based frameworks have not been conceived for this purpose. Their design is dependent on infrastructure support such as LDAP directories and CIM repositories and their implementation does not scale easily down to smaller devices such mobile-phones, PDAs or sensors. Policy deployment is often based on centralised provisioning and decision-making. In contrast, pervasive systems rely on collaborations between autonomous entities that must use policies in extensible architectures that cater for the management of resources and services as well as for composition and federation of policy spaces across devices.

Section 2 discusses two application scenarios and their requirements. Section 3 presents the Self-Managed Cell (SMC) architectural pattern and Section 4 focuses on the design of the SMC's policy service.

Section 5 discusses relationships between SMCs whilst Section 6 looks at future challenges in this area.

2. Examples and scenarios

2.1 Healthcare Monitoring

The AMUSE project [6] considers the case of a body area network of physiological sensors and actuators monitoring the health of a patient. Conditions that require long-term maintenance such as diabetes or episodic manifestation such as cardiac arrhythmia are good candidates for this type of monitoring. Sensors include cardiac monitoring, oxygen saturation as well as general temperature sensors and accelerometers and are based on the BSN platform [7] (Figure 1). These sensors have low-power 16-bit processors, 64 KB RAM, 256KB Flash memory, 6 analogue channels for sensors and use IEEE 802.15.4 radio. They may need to survive for long periods of time without battery replacement so communication must be reduced and sensor data must be processed on the node itself. This requires use of configurable thresholds and dynamic configuration of the actions taken when thresholds are crossed. A simple policy interpreter has therefore been designed and implemented for this platform [8].

A Gumstix device [9] hosts the management services presented in Section 3 and provides overall management for the body area network. This body area network needs to adapt continuously to varying conditions including: sensor failure or addition of new sensors, changes in user activities e.g. running, which affect cardiac thresholds and changes in the patient's condition e.g., abnormal heart rate. Additionally, the body area network needs to interact with other devices or collections of devices such as the medical equipment brought by a nurse during a home visit or at doctor's clinic. These interactions go beyond simple service invocations and comprise exchanges of notifications, policies and goals. For example, the nurse may load new policies on anomaly reporting or goals for maintaining heart rate below specific thresholds.

2.2 Autonomous Unmanned Vehicles

Autonomous unmanned vehicles exhibit similar characteristics in terms of coordinating activities e.g., movement and navigation based on multiple sensor inputs. They are typically used in situations where it is difficult or dangerous for humans to enter such as disaster sites, or areas containing explosives or chemicals. Typically, unmanned vehicles are self-propelled and include sensors for obstacle detection, video cameras and specialised sensors eg. to detect radiation or chemicals. Although a combination of planning mechanisms are typically used to control vehicle motion, policies are often used to manage the system configuration and to control the planning tasks themselves; for example to determine when plans have to be aborted or when new plans should be loaded.

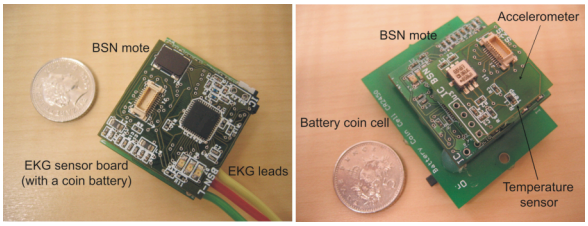

Figure 1 EKG Sensor board and BSN node with temperature and accelerometer

Autonomous vehicles rarely work in isolation but collaborate and aggregate into fleets that pursue a common goal such as to investigate a particular area. In addition to direct interactios this requires collaboration, and exchange of high-level goals between vehicles.

3. The Self-Managed Cell (SMC)

These scenarios share a similar structure: heterogeneous resources are grouped in autonomous domains that require adaptation and implement a feed-back control loop; in our case based on policies. Such autonomous domains, called *Self-Managed Cells* collaborate with each other and may compose into larger SMCs. However, this structure occurs at different levels of scale e.g. body-area networks, vehicles, rooms, buildings or large-scale distributed applications. Thus, a SMC is an *architectural pattern* that can be instantiated in different environments. It must therefore comprise services that may have different implementations in different SMC instantiations. Additionally, the set of services must be extensible to meet the management requirements of the application domain. As most management systems are event-driven, we assume that SMCs comprise a set of services that interact using a common publish/subscribe event bus (Figure 2). The event bus de-couples the services and permits the addition of new services without disrupting the behaviour of existing ones.

The SMC's *core services* are: the *event service*, the *policy service* and the *discovery service* as they are required to discover new devices and to implement a policy-driven feed-back control loop for adaptation. Further services can be added dynamically and may include *context services* that gather environment data such as location and local conditions, *planning services* that control motion and determine how to achieve goals or *provisioning services* for resource allocation. Similarly, authentication, trust or audit services can be introduced for security needs. The event bus allows

services to respond independently to the same notifications with different actions and could be used for application as well as management data. We have developed a simple publish-subscribe event bus that provides at most once persistent event delivery for body area networks and use other implementations such as XMLBlaster or Elvin in larger environments.

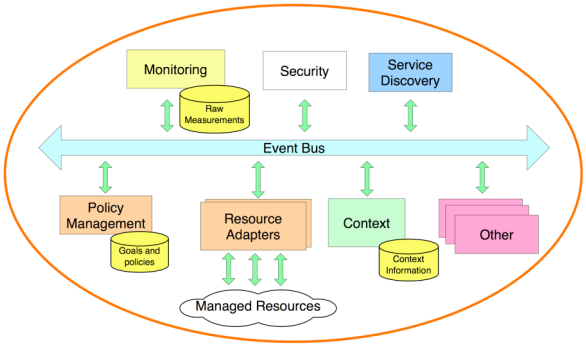

Figure 2 The SMC Architectural Pattern

The *discovery service* is used to discover nearby components that are capable of becoming members of the SMC or interacting with it. This includes sensors, services and other SMCs when they come into communication range. It interrogates new devices to establish a profile describing the services they offer and then publishes an event describing the addition of the new device on the event bus. The discovery service also maintains the SMC membership in order to detect failure or permanent departure of any components.

4. Ponder2: A policy service for SMCs

Ponder2 is a policy service for the SMC. Although based our experience with Ponder, its design and implementation are specifically aimed at Autonomous Pervasive Systems. Ponder2 has been implemented for J2SE, J2ME and in a basic form for the BSN sensors (see http://ponder2.net for the J2SE implementation).

Ponder2 focuses on the ability to interact with a running system that contains *managed objects* representing devices such as sensors, actuators, services and other SMCs. Managed objects act as adapter objects to the SMC's resources to provide a uniform view for policy enforcement and hide device specific protocols or interfaces. Managed objects provide a set of *commands* that can be invoked from Ponder2 and objects can be grouped into hierarchical *domains* for applying a common policy. Domains are managed objects themselves, thus making it possible to assign or remove objects from domains and to modify the domain structure through policies. For example, policies are used to decide which managed objects should be created when new devices are discovered, where they should be placed and which *missions* (Section 5) should be loaded on discovered SMCs.

On startup, the policy service only knows about the calling convention for executing commands on managed objects. Any code necessary for creating managed objects must be dynamically loaded by importing factories for each managed object type. Ponder2 has therefore few requirements from the environment in which it runs and can be extended to perform more complex tasks or to handle additional devices by adding new managed objects. Commands on managed objects are encoded in XML. Although compromises performance it facilitates interoperability with other systems that can interact with the policy service by generating the required XML. However, writing XML is laborious for humans and a higher-level language will soon be released.

Ponder2 caters for two policy types: *authorisation policies* that define which actions are permitted under given circumstances and *obligation policies* that define which actions should be performed in response to events if specific conditions are fulfilled (event condition action rules). We focus here on obligations as authorisations are detailed in [11]. Policies are themselves managed objects. Thus, it is possible to define new policy types by designing new factories and only factories for the policy types used need to be loaded. Obligation polices are created by invoking the appropriate factory with the event, condition, and actions as parameters. Actions include the specification of the target objects or domains to which they apply. Events are received from the event bus and an *event factory* is used as an adapter to the event bus in order to issue subscriptions for the event types required and transform received notifications into internal events that are used to trigger policies. When an event notification is received, all the policies triggered by that event are evaluated.

Ponder2 services federate and interact with each other by exchanging XML commands that can be nested. It is thus possible to invoke operations create policies, event subscriptions and raise event notifications that trigger policies in remote Ponder2 services. However, such interactions are subject to the authorisation policies as detailed in [11]. This provides the ability to support interactions across SMCs as described in the next section.

5. Relationships between SMCs

There are broadly two types of relationships between SMCs: *peer-to-peer* relationships occur between neighbouring SMCs for example between a patient body area network and the nurse SMC during a home visit or between two unmanned vehicles, and

composition relationships occur when a managed resource within an SMC is itself an SMC with its own resources, for example a patient body area network that contains a diagnostic device relying on its own sensors, or a fleet of unmanned vehicles formed to achieve a particular objective. Whilst peer-to-peer relationships occur frequently as SMCs interact with neighbouring components, *composition* relationships enable grouping SMCs into larger autonomous structures and scaling SMC management to complex environments.

Both relationships types have similar requirements in terms of types of interactions between the SMCs involved. In both peer-to-peer and composition relationships each SMC must be able to send commands to the other SMC to be executed. For example, the nurse SMC may query a patient SMC for heart rate logs since the last visit or a patient SMC may request a house control SMC to raise the ambient temperature. In both peer-to-peer and composition relationships each SMC must be able to subscribe to event notifications from the other SMC, and notify the other SMC of event occurrences of interest to which it has subscribed. For example, a patient monitoring SMC needs to notify a nurse SMC of variations in monitored values during the visit and the nurse SMC must be able to subscribe to them. And finally, in both peer-to-peer and composition relationships, each SMC must be able to load policies into the other SMC to direct it to behave in a particular way. For example, the nurse may program the patient SMC to start an intensive electrocardiograph whenever the patient's physical activity increases or an unmanned vehicle may require the vehicle behind to stop whenever its breaking lights are on. Loading policies into a remote SMC is in essence a constrained form of programming, and used in conjunction with event notifications enables an SMC to distribute its feed-back control loop by requesting remote SMCs to react in a specific way to events that the source SMC raises.

Although there are similarities in the interaction types needed for peer-to-peer and composition relationships there are also significant differences. Typically, in compositions parent SMCs will be given access to the resources and be permitted to subscribe to most events in the inner SMCs. For example, a diagnostic device in a body-area network SMC will allow the SMC to load new decision algorithms and new policies into it. Composition also implies that contained SMCs cease to advertise themselves independently and rely on the containing SMC to bind them with other devices. This allows the containing SMC to hide the complexity of the composed structure and selectively expose its internal functionality according to requirements. For example, a patient SMC would expose its sensors to doctors, but hide them from other patients.

Both peer-to-peer and composition relationships are based on common principles: a *SMC must retain autonomy* i.e., be able to deny operations that compromise its integrity, a SMC must *encapsulate* its own resources and export through *customised interfaces* only selected services and resources and must retain the ability to *mediate* interactions between its resources and external SMCs through proxies that can filter the commands received.

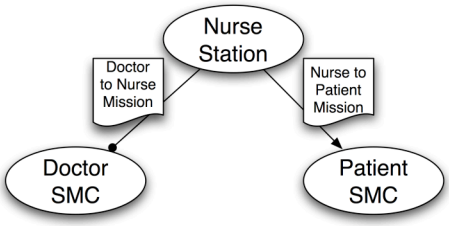

Figure 3 Mission Distribution

To facilitate the establishment of both peer-to-peer and composition relationships we introduce the concept of a *mission* as a means of grouping the duties of a remote SMC specified in terms of the policies it must enforce. Thus, a *mission* is a group of obligation policies specified in terms of the interfaces of two or more interacting SMCs. These interfaces specify the *events* SMCs are making available to the mission policies, the *notifications* that mission policies can publish in the SMCs and the commands SMCs will accept from the mission policies. Missions can be nested and a mission may contain other missions that need to be loaded on another SMC. For example, a doctor may load a mission in a nurse station that specifies policies for the nurse station to report to the doctor as well as the mission to be loaded by the nurse station in the patient SMC (Figure 3).

This distribution occurs frequently in fleets of unmanned vehicles where missions can be distributed from a commanding vehicle to all others and can be used to establish subgroups of autonomous vehicles.

6. Future Challenges

The concepts described above are only a first step towards realising autonomous pervasive systems – numerous challenges remain to be addressed. Although flexible, missions defined as sets of policies are not sufficient to capture complex interactions between multiple SMCs and higher-level abstractions are required. Just as the SMC is an architectural pattern for providing autonomy to a group of devices, *relationship patterns* that capture interactions and coordination between multiple SMCs need to be introduced. SMC

adaptation currently relies on simple obligation policies and does not assume that reasoning procedures are used. However, numerous scenarios are better implemented through SMCs exchanging goals and planning their actions to maintain or achieve the received goals. Results from other communities including multi-agent systems and robotics are applicable and their integration with policy-based management techniques must be investigated. Reasoning procedures could also be used to implement conflict detection, policy analysis and policy refinement [12] for small devices. These are required as policies exchanged between SMCs may often conflict in attempting to achieve different goals. On one side achieving this requires scaling down reasoning procedures to small devices, on the other hand this also requires understanding of how the complexity and performance of such procedures varies with the characteristics of the problem domain. As devices vary greatly in size and capabilities so the reasoning and analysis techniques must also be able to scale down as well as up to larger environments.

This paper has ignored trust and security aspects but this is not for their lack of importance. Medical scenarios raise particularly difficult challenges as the need for security, integrity and privacy often conflicts with medical requirements. Who can access the medical data in a patient's SMC in an emergency? How serious must the emergency be before access is granted? Security in pervasive systems tends to be fluid and context dependent requiring decisions based on imprecise measures of trust and risk. Policies need to focus more on how the tradeoffs are addressed rather than simply traditional authorisation rules.

7. Conclusions

The SMC pattern is being used in a several application domains. Although our focus so far has been on SMCs for health monitoring, we are also working on its use for autonomous unmanned vehicles, mobile ad-hoc networks and larger distributed systems such as virtual organisations. Its design combines the flexibility of an asynchronous event bus with a policy-based service to implement adaptation and feedback control.

Ponder2 preserves some of the core Ponder concepts yet differs significantly from the original design. Core abstractions have been reduced to a minimum to enable use on devices with varying capabilities. Dynamically loadable factories provide the ability to extend the policy service to perform complex functions and to interact with heterogeneous resources. By using the Managed Object abstraction uniformly across SMC resources and core SMC concepts such as domains, events and policies we obtain a self-managed system where policies can be used for managing other policies, domains and events but also a system that promotes separation of concerns between different management aspects: discovery of new devices, classification and domain assignment, policy enforcement.

Realising autonomous pervasive systems requires revisiting techniques developed for larger systems, simplifying their design and emphasising their complementary functions. The mobility and dynamics encountered in pervasive systems, combined with the lack of infrastructure and administrative support requires designing systems that address tradeoffs in a practical way rather than striving for more expressive solutions. Realising autonomous pervasive systems will require combining techniques from multiple subject areas such as AI, network and systems management, security and multi-agent systems and integrating these techniques on small scale devices as well as in larger systems.

Acknowledgements

We gratefully acknowledge the contribution of K. Twidle, S.-L. Keoh, S. Strowes, S. Heeps and A. E. Schaeffer-Filho to the development of the concepts and implementations described in this paper. This work was financially supported by the UK EPSRC (GR/S68040/01 and GR/S68033/01) and the CEC (TrustCoM project 1945).

References

[1] Proc. 4th-7th IEEE Int. Work. on Policies for Distributed Systems and Networks. IEEE CS Press
[2] G.Z.Yang (Ed.), Body Sensor Networks, Springer-Verlag, March 2006.
[3] Proc. 4th Int. Conf. On Smart Homes and Health Telematics, Belfast 2006.
[4] Proc. Systems Engineering for Autonomous Systems Defence Technology Centre Conf., Edinburgh, 2006.
[5] http://www.cityware.org.uk/
[6] AMUSE Project http://www.dcs.gla.ac.uk/amuse/
[7] http://www.doc.ic.ac.uk/vip/ubimon/bsn_node/
[8] Keoh, SL et al. Policy-based Management for Body-Sensor Networks. 4th IEEE Conf. on Wearable and Implantable Body Sensor Network, AAchen, 2007
[9] http://www.gumstix.com/
[10] Damianou, N et al. The Ponder Policy Specification Language. Policy Workshop, Jan. 2001, Bristol, Springer-Verlag, LNCS 1995.
[11] Russello, G et al. Authorisation and Conflict Resolution for Hierarchical Domains. IEEE Workshop on Policies for Dist. Sys. and Networks, Bologna, 2007
[12] Bandara, A. et al. Policy Refinement for DiffServ Quality of Service Management. IEEE eTrans. on Network and Service Management. 3(2):2-13, 2006