Verification of Policy-Based Self-Managed Cell Interactions Using Alloy

Alberto Schaeffer-Filho, Emil Lupu, Morris Sloman, Susan Eisenbach
Department of Computing, Imperial College London
180 Queen's Gate, SW7 2AZ, London, England
Email: {aschaeff, e.c.lupu, m.sloman, sue}@doc.ic.ac.uk

***Abstract*—Self-Managed Cells (SMCs) define an infrastructure for building ubiquitous computing applications. An SMC consists of an autonomous administrative domain based on a policy-driven feedback control-loop. SMCs are able to interact with each other and compose with other SMCs to form larger autonomous components. In this paper we present a formal specification of an SMC's behaviour for the analysis and verification of its operation in collaborations of SMCs. These collaborations typically involve SMCs originated from different administrative authorities, and the definition of a formal model has helped us to verify the correctness of their operation when SMCs are composed or federated.**

***Keywords*-policy-based management; self-managed cells; interactions; model-checking;**

I. Introduction

Management in complex pervasive environments cannot rely on human intervention, and systems must be *self-managing* with local decision making and feedback control to enable seamless adaptation. We have introduced the concept of a *Self-Managed Cell (SMC)* as a pattern for building ubiquitous computing applications [1]. An SMC implements a policy-driven feedback control-loop that determines which management and reconfiguration actions should be performed in response to events of interest such as device failures or context changes. Autonomous SMCs must be able to interact with each other in complex ways, federate or compose into larger structures. Systems such as *body-area networks* for monitoring a patient's health must continuously adapt to changes in their environment or in their usage requirements. These may comprise *"smart sensors"* and diagnosis devices, but body-area network SMCs may also interact with a number of other *peer* SMCs such as the SMC running on the PDA of a doctor, or the SMC controlling the room in which the wearer is present. SMC interactions comprise the invocation of actions from one SMC on another but also exchanges of events and policies between SMCs.

In this paper we present a formal specification of the overall SMC behaviour and its analysis in collaborations across SMCs, in which consistent policy deployment is crucial. When these SMCs are federated, inconsistencies may prevent them from operating as originally expected. The definition of a formal model assists in the design of SMC collaborations and allows us to verify the correctness of anticipated SMC interactions before these interactions are implemented and policies are deployed in physical devices (e.g. PDAs, mobile phones, sensors). We show how our formal model is used to automatically type-check policies (matching the policy events and operations with the functionality provided by the SMCs involved) and verify the consistency of collaborations, allowing us to identify conflicting or inconsistent policy deployment across distributed SMCs.

We chose the *Alloy Analyzer* [2] as the platform for this formal specification as it allows us to declaratively express complex structural constraints and behavior in SMC interactions. Defining a formal specification for Self-Managed Cells in *Alloy* allows us to: *(1)* formally capture the static and dynamic aspects of the structure and behaviour of the SMC interactions; *(2)* automatically verify the consistency of SMC collaborations; and *(3)* simulate SMC behaviour in complex interactions thus increasing our confidence in the correctness of a given model.

This paper is structured as follows: Section 2 presents an overview of the SMC architecture. Section 3 briefly describes our formalisation of Self-Managed Cells, and Section 4 shows how this model can be used to automatically type-check and verify the consistency of collaborations of SMCs. Section 5 presents our concluding remarks.

II. Self-Managed Cells and their Interactions

An SMC forms an autonomous administrative domain that consists of both hardware and software components. A typical set-up representing a patient's *body-area network* we use for healthcare monitoring comprises a Gumstix[1] device hosting management services that controls several sensors (e.g., heart-rate, temperature, acceleration) hosted on BSNs (Body Sensor Nodes)[2] as well as other devices such as diagnostic devices hosted on PDAs or other Gumstix.

An SMC [1] comprises a dynamic set of management services integrated through a publish/subscribe *event bus*. The SMC relies on a *policy service*[3], including both *obligation* and *authorisation* policies. Policies can be dynamically added, removed, enabled and disabled to change the behaviour of an SMC without interrupting its functioning. Finally, a *discovery service* is used to detect new devices in

[1] http://www.gumstix.com
[2] http://vip.doc.ic.ac.uk/bsn/
[3] http://www.ponder2.net

the vicinity of the SMC, such as sensors and other SMCs, and is responsible for managing the SMC's membership, to distinguish transient failures from permanent departures from the SMC (e.g., device out of range or switched off).

However, to realise larger applications SMCs must be able to interact with each other in complex ways. We use the concept of *roles* to facilitate these interactions, where *roles* are placeholders for remote SMCs discovered at run-time. Roles can be thought of as typed-domains in a hierarchical *domain structure* that each SMC maintains to manage its resources. A remote SMC is assigned to a role in another SMC if it fulfills the requirements for that role i.e., if it provides the operations required by that domain. Once an SMC is assigned to a role, policies previously associated with a role (i.e., having that role as subject or target) will then apply to SMCs assigned to it. This allows us to specify policies governing the interactions with an SMC before that SMC is discovered [3].

III. Formal Specification of Self-Managed Cells

Models written in Alloy [2] can be automatically checked for correctness using its analyser, and a visualiser can also be used to display example (or counter-example) structures graphically. Analysis in Alloy is performed over restricted scopes and the user controls the number of objects to be used (the user-specified scope makes the problem finite and thus reducible to a boolean formula). This is based on the *small scope hypothesis*, that for any flawed design a counter-example should be found by an exhaustive search of a comparatively small, bounded scope.

The *structure* of an Alloy model is defined through a set of *signatures*. The main component in our model is the SMC, represented by the signature *SelfManagedCell* below.

```
abstract sig SelfManagedCell
{
    provides: some Interface,
    requires: some Role,
    obligations: set Obligation,
    authorisations: set Authorisation
}
```

In the declaration of a signature body we can define a number of relations, which can be thought of as fields of an object in the OO paradigm. The *SelfManagedCell* signature specifies four relations. The first two, *provides* and *requires*, define respectively which *interfaces* an SMC is able to offer to remote SMCs and which *roles* an SMC requires to be fulfilled (by remote SMCs). The other two relations, *obligations* and *authorisations*, define the policies an SMC is enforcing. *SelfManagedCell* is an **abstract** signature, meaning it can be extended to define a specialized component in our model (e.g. *DoctorSMC*, *PatientSMC*, *SensorSMC* would all extend the base *SelfManagedCell* signature).

An *Interface* defines the *operations* (methods that can be invoked), *events* (which can be published externally) and *notifications* (which are external events of which the SMC can be notified) supported by that interface, as described in [3]. It is defined by the signature below.

```
sig Interface
{
    operations: set Operation,
    events: set Event,
    notifications: set Notification
}
```

A *Role* behaves as a placeholder for remote SMCs, which can have their interfaces assigned to it. The complete model is too large to be presented here but is available online[4].

SMCs enforce two types of policies: *obligations*, which cater for the adaptive behaviour of SMCs, and specify what management actions (or *operations*) *subjects* must perform on *targets* in response to *events*; and *authorisations*, which are access control rules that specify what actions *subjects* are permitted (positive authorisation) or forbidden (negative authorisation) to perform on a *target*. The subjects and the targets are roles within the context of the SMC in which the policy is specified.

The *ConcreteObligation* signature extends *Obligation* and defines the subject and target roles, and the event that triggers the policy and action to be invoked.

```
sig ConcreteObligation extends Obligation
{
    subject: one Role,
    event: one Event,
    action: one Operation,
    target: one Role
}
```

Similarly, the signature *ConcreteAuthorisation* defines a subject role, a target role, an action and the modality of the policy (which can be either positive or negative).

```
sig ConcreteAuthorisation extends Authorisation
{
    modality: one Modality,
    subject: one Role,
    action: one Operation,
    target: one Role
}
```

The *behaviour* of an Alloy model is defined through a set of *predicates*, which in our case represent the elementary behaviour of SMCs, such as the discovery and departure of SMCs, assignment and de-assignment of SMCs (interfaces) to/from SMC roles, and loading and unloading of policies. A common technique to represent dynamic behaviour in Alloy is to show how the *"state"* of the system before the operation differs from the *"state"* after the operation, i.e. what properties hold before and what properties hold after (pre and post conditions). The typical way of doing so is to have an additional signature that represents the entire

[4]http://www.doc.ic.ac.uk/∼aschaeff/alloy/policy-model.zip

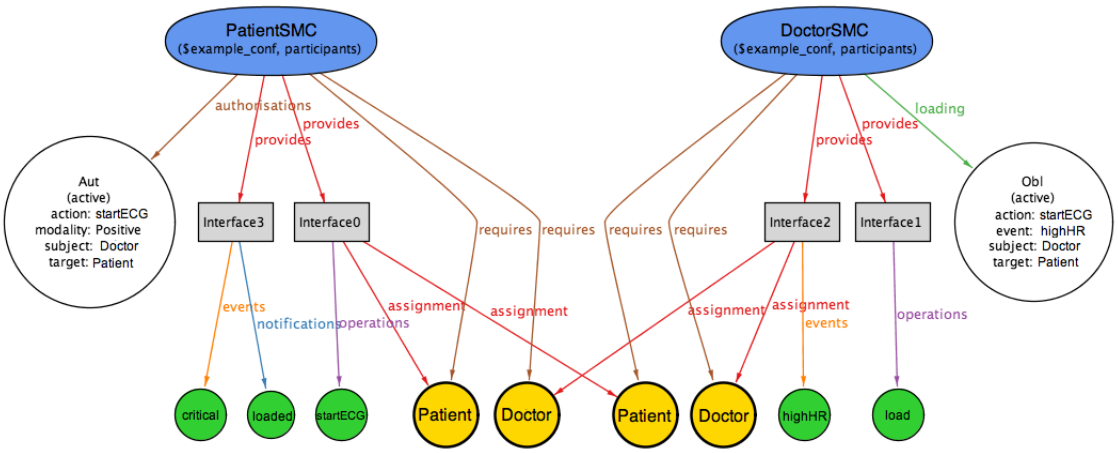

Figure 1. A valid configuration generated in Alloy (all active obligations have a corresponding active authorisation).

system being modelled. Then, for each operation we want to model, we define a *predicate*, which takes as arguments a system S and a system S', and show how S differs from S' in this predicate. This corresponds to showing the operation happened in S then resulting in S'.

We encoded this notion of *"state"* of an interaction in an additional *signature* (not shown here), named *Configuration*. An instance of *Configuration* thus represents the interaction between SMCs at a given time point, which is defined by the current set of *participants* in this interaction, the *assignments* of participants (represented by their provided interfaces) to roles, the policies that are exchanged (*loaded*) between the SMCs and, among those, the policies that are currently active in each of the participating SMCs.

IV. Model-Checking and Policy Analysis

We use the Alloy Analyzer to automatically verify the consistency of specific SMC collaborations. In particular we can verify the role assignments when establishing a collaboration, and verify inconsistent policy deployment across SMCs, where obligation policies do not always have a corresponding authorisation policy.

Consider the configuration illustrated in Figure 1. The *DoctorSMC* has an obligation policy *Obl*, which has *Doctor* and *Patient* as the respective subject and target of the policy, is triggered by the event *highHR*, to perform the *startECG* action. Note that *Doctor* and *Patient* are roles required by the *DoctorSMC*. On the other hand, the *PatientSMC* has an authorisation policy *Aut*, which defines *Doctor* and *Patient* as the subject and target of the policy respectively. This is a positive authorisation (labelled *"modality: Positive"*) that permits the execution of the *startECG* action. Here *Doctor* and *Patient* are roles required by the *PatientSMC*.

In terms of role assignments, the *DoctorSMC* provides *Interface2*, which is assigned to the local role *Doctor*. This is a local assignment, where the role *Doctor* is also the subject role of the obligation policy enforced by this SMC. In this example, the same interface is exported to be seen by the patient and assigned to the *Doctor* role in the remote *PatientSMC* (where *Doctor* is the subject role of the authorisation policy enforced by that SMC). Similarly, the *PatientSMC* provides *Interface0*, which is assigned to the local role *Patient* and then also exported and assigned to the *Patient* role required by the remote *DoctorSMC* (where *Patient* is the target role of the obligation policy enforced by that SMC).

Type-checking is required for policy activation, as the SMCs (i.e. their interfaces) assigned to a role must provide the functionality expected by the policies written in terms of that role. Note that in Figure 1, *Interface2* (assigned to the subject of the obligation) provides the event *highHR* (required to trigger the obligation policy), and *Interface0* (assigned to the target of the authorisation) provides action *startECG* (which is the action to be allowed execution by the authorisation policy). Moreover, each obligation has a corresponding authorisation, which allows the action specified by the obligation to be executed. Figure 2 illustrates the predicate to determine whether all obligations have a matching positive authorisation policy. It states that for all SMCs and for all obligations, if one SMC has this policy, then there must be some positive authorisation which is enforced by another SMC and specifies the same action as in the obligation, and which has the same interfaces assigned to the subject and target roles in both policies.

V. Concluding Remarks

We have considered several alternatives for the formal specification of an SMC's behaviour including *pi-calculus* [4], *ambient calculus* [5], *channel ambient calculus* [6] and others (which model computation operationally). In contrast, the declarative specification style used by *Alloy* was simpler to use and the associated tool set offered rapid

```
all conf: Configuration, smc1,smc2: conf.participants, obl: (conf.active & ConcreteObligation)
{
    (obl in (smc1.obligations + smc1.(conf.loading)))
        ⇒ some aut: (conf.active & ConcreteAuthorisation)
            {
                (aut in (smc2.authorisations + smc2.(conf.loading)))
                and (aut.modality in Positive)
                and (obl.action == aut.action)
                and ((obl.subject).~(conf.assignment) == (aut.subject).~(conf.assignment))
                and ((obl.target).~(conf.assignment) == (aut.target).~(conf.assignment))
            }
}
```

Figure 2. Determining whether all obligations have a matching authorisation policy (consistent policy deployment).

feedback on the model specification. Its visualiser provides a intuitive way of making sense of the solutions, which has enabled us to rapidly fine tune the specification and uncover omissions from the informal models developed earlier.

Several studies have looked at the (conflict) analysis of policies in various forms [7], [8], [9], [10], some of it based on model-checking techniques. In contrast to these studies our focus is not on the ability to detect policy conflicts, but to unambiguously specify the desired behaviour of interacting Self-Managed Cells and then verify if these SMCs are able to enforce their policies (type-checking events/operations to interfaces, all obligations are authorised, etc). This allows us to analyse SMC behaviour when interactions across several cells occur. Thus we can model policy-based SMC interactions and verify the correctness of these interactions before its actual implementation and deployment in physical devices.

We can further check whether in given circumstances, authorisations permit the execution of actions that would threaten the integrity of the SMC or whether failure or departure of devices leaves the SMC in a state where it is no longer able to fulfill its primary functions. This is particularly important when developing body area networks for health monitoring as the integrity of the sensor configuration influences the medical interpretation of the physiological parameters collected. More traditional types of analysis, such as detecting policy conflicts in the form of modality or application specific conflicts can also be performed.

ACKNOWLEDGMENT

Research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. We also acknowledge financial support from the EC IST EMANICS Network of Excellence (#26854).

REFERENCES

[1] E. Lupu, N. Dulay, M. Sloman, J. Sventek, S. Heeps, S. Strowes, K. Twidle, S.-L. Keoh, and A. Schaeffer-Filho, "AMUSE: autonomic management of ubiquitous systems for e-health," *J. Concurrency and Computation: Practice and Experience, John Wiley*, vol. 20(3), pp. 277–295, May 2008.

[2] D. Jackson, "Alloy: a lightweight object modelling notation," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 2, pp. 256–290, 2002.

[3] A. Schaeffer-Filho, E. Lupu, N. Dulay, S. L. Keoh, K. Twidle, M. Sloman, S. Heeps, S. Strowes, and J. Sventek, "Towards supporting interactions between self-managed cells," in *Proceedings of the 1st IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, Boston, USA, July 2007, pp. 224–233.

[4] R. Milner, J. Parrow, and D. Walker, "A calculus of mobile processes, I," *Inf. Comput.*, vol. 100, no. 1, pp. 1–40, 1992.

[5] L. Cardelli and A. D. Gordon, "Mobile ambients," in *Proceedings of the 1st International Conference on Foundations of Software Science and Computation Structure (FoSSaCS)*. London, UK: Springer-Verlag, 1998, pp. 140–155.

[6] A. Phillips, "Specifying and implementing secure mobile applications in the channel ambient system," Ph.D. dissertation, Imperial College London, April 2006.

[7] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, "Verification and change-impact analysis of access-control policies," in *Proceedings of the 27th international conference on Software engineering (ICSE)*. New York, NY, USA: ACM, 2005, pp. 196–205.

[8] A. Bandara, E. C. Lupu, and A. Russo, "Using event calculus to formalise policy specification and analysis," in *Proceedings of the 4th IEEE Int. Workshop on Policies for Distributed Systems and Networks (Policy)*, Como, Italy, June 2003.

[9] M. Charalambides, P. Flegkas, G. Pavlou, J. Rubio-Loyola, A. Bandara, E. Lupu, A. Russo, M. Sloman, and N. Dulay, "Dynamic policy analysis and conflict resolution for diffserv quality of service management," in *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Vancouver, Canada, 2006, pp. 294–304.

[10] J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, G. Pavlou, and A. L. Lafuente, "Using linear temporal model checking for goal-oriented policy refinement frameworks," in *Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (Policy)*, Washington, DC, USA, 2005, pp. 181–190.