Realising Management and Composition of Self-Managed Cells in Pervasive Healthcare

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

***Abstract*—Research in pervasive and autonomic computing focuses on supporting services for pervasive applications, but often ignores how such applications can be realised through the federation of autonomous entities. In this paper we propose a methodology for designing collaborations between autonomous components, using the Self-Managed Cell (SMC) framework. We focus on the structural, task-allocation and communication aspects of management interactions between SMCs. We propose a catalogue of architectural styles for SMC interactions, and a model for combining architectural styles in patterns of interactions that can be enforced by different SMCs in large collaborations. This allows us to specify the management of large-scale systems by composing management functions using architectural styles as building block abstractions. A scenario for a health monitoring application involving a number of SMCs is used throughout the paper to illustrate how complex structures can be thus built.**

I. Introduction

Management in pervasive systems cannot rely on human intervention or centralised decision-making functions. Systems such as body-area networks of sensors and actuators for monitoring a patient's health must be autonomous and continuously adapt to changes in their environment or in their usage requirements. They must therefore be *self-managing* with local decision making and feedback control to enable seamless adaptation. We have previously introduced the concept of a *Self-Managed Cell (SMC)* as an infrastructure for building ubiquitous computing applications [1]. A SMC consists of a set of hardware and software components which form an autonomous administrative domain. SMCs implement 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, context changes or changes of state in the SMC's resources.

To realise larger applications, autonomous entities such as SMCs must be able to interact with each other in complex ways, federate or compose into larger structures. For example, a body-area network monitoring a patient's health may comprise "smart" sensors and complex diagnosis devices that are SMCs in their own right. In the same way, a SMC controlling a room will be aggregated under the control of a house SMC. Body-area network SMCs may also interact with a number of other *peer* SMCs such as the SMC running on the PDA of a nurse or a doctor, or the SMC controlling the room in which the wearer is present. In all such examples, there is a need to *structure* interactions, for example, as compositions, aggregations or peer-to-peer collaborations. There is also a need to define how SMCs *interact* with each other in such aggregated groups to distribute management responsibility, application tasks or to implement communication patterns. Current work on pervasive and autonomous systems focuses on middleware services that provide supporting functionality for applications [2]. However, these studies often ignore how such applications can be realised through the federation and collaboration of autonomous components.

We advocate the use of *architectural styles* for systematically specifying reusable collaborations among *Self-Managed Cells*. Architectural styles can be used to describe management relationships between SMCs and are similar in intent to software design patterns [3], in the sense that they provide a set of standard solutions for recurring problems. However, unlike design patterns, architectural styles are associated with a parameterised implementation that enforces the semantics of the relationship. We distinguish between three main categories of architectural styles: *structural styles* specify how SMCs are organised and structured and address issues such as interface mediation, filtering, encapsulation and SMC visibility; *task-allocation styles* specify control flow and task allocation between the elements in a structural relationship; and *communication styles* specify information flow and event-forwarding behaviour between SMCs. These categories can be seen as complementary *perspectives* for defining management relationships between autonomous SMCs. The use of architectural styles provides a better understanding of the relationships between SMCs and promotes reuse of common abstractions. We aim to design larger interactions built systematically from simpler ones by combining architectural styles as design elements of a complex collaboration. In this paper we attempt to identify a catalogue of architectural styles to represent abstractions for management relationships between SMCs, and the means of combining these abstractions to build complex SMC collaborations. We focus on interactions for the design of a health monitoring application in the medical domain. However, the same principles have also been applied in a related project to the management of ad-hoc communities of unmanned autonomous vehicles (UAVs) for surveillance missions in hostile environments [4].

The remainder of this paper is organised as follows: Section II discusses some related work. Section III briefly describes the background work on the Self-Managed Cell

Digital Object Identifier: 10.4108/ICST.PERVASIVEHEALTH2009.5979
http://dx.doi.org/10.4108/ICST.PERVASIVEHEALTH2009.5979

architecture. Section IV outlines our healthcare monitoring scenario. Section V introduces the use of architectural styles for structuring management interactions between SMCs. Section VI presents our interaction model. Section VII revisits the health monitoring scenario and shows how it can be realised using the architectural abstractions. Section VIII briefly describes our implementation and Section IX presents our concluding remarks.

II. Related work

The software engineering community has long investigated software architecture-based approaches, which typically separate computation (*components*) from interactions (*connectors*). The benefits brought by the distinction between components and connectors have been widely recognised in the software community as means of structuring software development [5], [6]. However, components and connectors are low level implementation abstractions that do not cater for the adaptive behaviour of SMCs as expressed in policies for example.

Collaborations between SMCs will typically involve the composition of a number of abstractions for realising more complex applications. The specification of these interactions uses ideas inspired from *architectural description languages (ADLs)*, *module interconnection languages (MILs)* and coordination schemas in general, such as Wright [7], UniCon [8], Conic [9], Darwin [10], Rapide [11] and Mobile UNITY [12]. However, although traditional architectural description languages and coordination schemas for mobile agents bind software components or distributed agents through connections, the semantics of these connections (such as *"sends data to"*, *"controls"* or *"is part of"*) is usually unclear, failing to represent higher-level relationships between components [13]. We advocate the use of a variety of architectural styles that have individual semantics for higher-level SMC interactions. In contrast to architectural description languages we are not interested in general-purpose component interactions, but instead aim for a model that addresses the management of collaborations using the SMC infrastructure, based on abstractions for the exchange of interfaces, tasks and events.

Work on multi-agent systems has investigated the use of organisational structures for designing multi-agent societies. Holonic models [14] for example often support hierarchical structures, but not more sophisticated abstractions. Few studies however have attempted to identify a catalogue of generic and reusable patterns for interactions between agents [15]. Aridor and Lange [16], for example, propose specific patterns for mobile agent applications (e.g. patterns for specification of an agent's itinerary or for meeting other agents when arriving at a specific location). Kolp and colleagues [17] propose a macro-level catalogue for interactions between agents by using real world business organisations as an analogy (e.g. structure-in-5, joint venture, etc). Also, Deugo and colleagues [18] present a set of patterns for mobile and intelligent agents. However, most of the work in multi-agent systems focuses on *distributed problem solving* [19] where goals are achieved by a decentralised collection of agents that possess different knowledge sources, and delegate tasks to each other according to their capabilities. In our case we are focusing on architectural styles for the specification of management relationships rather than general agent interactions. Although our work targets the SMC framework, which has well defined architectural principles and means of implementing adaptation, the architectural styles defined in this paper may have broader applicability and prove useful towards building pervasive autonomous systems in general formed of autonomous components.

III. Background: Self-Managed Cells

To provide autonomous management in pervasive environments, we have previously introduced the *Self-Managed Cell (SMC)* as the basic autonomous building block for our pervasive systems [1]. A SMC forms an autonomous administrative domain that consists of hardware components such as physiological sensors, mobile phones, PDAs and computers, as well as software services and components within those devices.

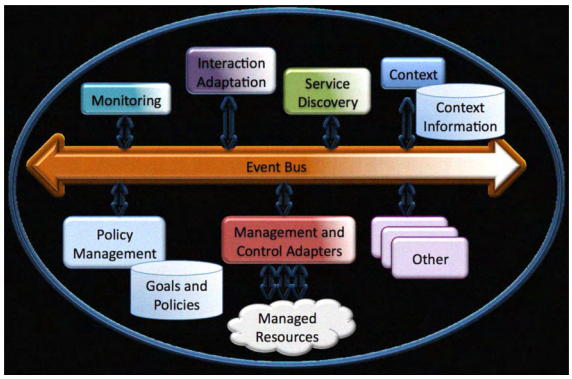

Fig. 1. Self-Managed Cell architecture.

A SMC comprises a dynamic set of management services integrated through a common publish/subscribe *event bus* (Fig. 1). This de-couples services, as event publishers do not require prior knowledge of the recipients when sending a message, and permits adding new services to the SMC without disrupting the behaviour of existing ones. The SMC relies on a *policy service*, which caters for two types of policies: *obligation policies* that define the adaptive actions that must be performed in response to events, and *authorisation policies* that specify which actions are permitted on which resources and services. Policies can be added, removed, enabled and disabled to change the behaviour of a SMC without interrupting its functioning. Our implementation of the policy service is based on the Ponder2[1] system. Finally, a *discovery service* is used to detect new devices in 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). A more detailed description of the SMC architecture can be found in [1].

[1] http://www.ponder2.net

Digital Object Identifier: 10.4108/ICST.PERVASIVEHEALTH2009.5979
http://dx.doi.org/10.4108/ICST.PERVASIVEHEALTH2009.5979

IV. Scenario Description: Health-Monitoring

Consider some of the requirements for a typical healthcare monitoring application. A personal SMC representing a patient's *body-area network* for health monitoring typically comprises a PDA or Gumstix[2] device hosting SMC management services that control several sensors such as heart-rate, temperature, acceleration, blood pressure and oxygen saturation hosted on BSNs[3] (Body Sensor Nodes). Actuators such as a pacemaker or an insulin pump SMC may be employed and activated according to conditions monitored by the sensors. PDAs or other Gumstix devices may also be used to host application services, e.g. for diagnostic. Communication with BSN nodes typically occurs through IEEE 802.15.4 radio links while communication between Gumstix devices or with PDAs occurs through Bluetooth or Wi-Fi.

A doctor or nurse SMC would typically interact with patients, loading monitoring tasks and collecting monitored results. Monitoring tasks running on the patient body-area network allow the patient to be self-monitored in his own home environment, thus promoting reduced usage of hospital resources and better medical evidence data for the clinical condition and its treatment. Tasks loaded by the healthcare worker continually run on the patient's SMC, relying on information provided by his body sensors. There are two situations of interest regarding the collected sensor data: *(a)* under normal circumstances data may be stored on a home server SMC, for synthesis and for subsequent delivery to the GP surgery (e.g. a scheduling task that sends a subset of this data every seventy-two hours); and *(b)* in an emergency data is used to request immediate assistance (e.g. in situations where the monitoring system detects that a heart attack is imminent).

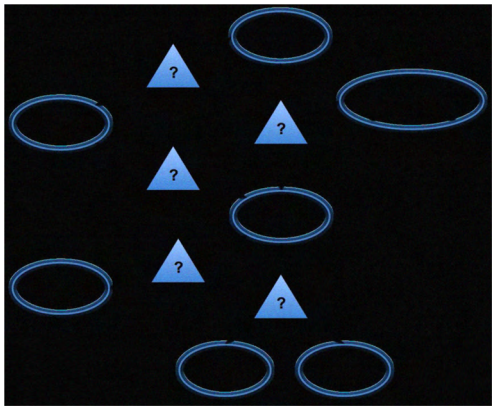

Fig. 2. Healthcare monitoring scenario: each triangle containing a question mark represents an interaction that requires a different combination of management abstractions.

Fig. 2 outlines the SMCs involved in this scenario. Note that the informal description does not specify how the relationships in this collaboration are realised. Although the *Self-Managed Cell* concept provides a suitable abstraction for representing autonomous components, we still need adequate abstractions for expressing their interactions. For example, if multiple SMCs are used on a patient to monitor related conditions they would typically be composed in a single autonomous SMC. In contrast interactions between the patient's body area network and healthcare personnel would typically be interactions between peers. Also, collected data is forwarded differently in a body-area network or in a home monitoring environment. Finally, tasks for physiological monitoring or data synthesis are loaded in specific situations and subject to different conditions. The purpose of this paper is to introduce a systematic way of designing interactions and clearly representing their management aspects.

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

V. Architectural Styles for Specifying Management Interactions

Autonomous SMCs can be dynamically assembled into larger and more complex structures that are also SMCs. In this way larger autonomous pervasive applications can scale up and be built from simpler yet autonomous components. This also allows even comparatively smaller components to work autonomously.

In such collaborations, SMC interactions rely on three elementary exchange mechanisms: *interfaces*, *policies* and *events*. By exchanging interfaces, a SMC allows remote SMCs to invoke operations on it, whilst still mediating access to its internal resources. Remote SMCs may be dynamically discovered and assigned to placeholders (termed *roles*) in the SMC. The role a SMC is playing in another determines which tasks (in the form of *missions*) prescribing its behaviour in the interaction it will receive. A *mission* is a set of policies that is dynamically loaded from a remote SMC to change the behaviour of another SMC within the context of the interaction. Finally, through the exchange of events a SMC may notify remote SMCs of changes of context, possibly triggering management actions in response to these changes. The underlying principles for realising SMC interactions in terms of interface access, role assignment and exchanges of events and policies were introduced in [20].

Based on different means of exchanging interfaces, policies and events, we propose in this paper three interrelated categories of management abstractions for realising SMC interactions. Each abstraction provides a particular manner of achieving one of these exchanges and the three categories represent respectively the *structural*, *task-allocation* and *communication* aspects of an interaction. These will be discussed in the next sections.

A. Structural Aspects

Structural aspects reflect how SMCs are organised with respect to interface access, visibility and encapsulation. For example, *peer-to-peer* interactions typically require only simple interface exchanges whereas *compositions* also need to implement *encapsulation* and *mediate* access to internal resources. Composed interactions allow one SMC to define the visibility of its inner resources to the outside world, creating a composed structure. Additional abstractions such as *filtering* of operations provide more flexibility to the interactions with

Digital Object Identifier: 10.4108/ICST.PERVASIVEHEALTH2009.5979
http://dx.doi.org/10.4108/ICST.PERVASIVEHEALTH2009.5979

TABLE I
CATEGORIZATION OF THE ARCHITECTURAL STYLES FOR INTERACTIONS BETWEEN SMCS.

Category	Architectural Style	Description
Structural Styles	Peer-to-Peer	Ordinary, symmetrical mode of interaction between SMCs that exchange services
	Composition	One SMC encapsulates another's interface and determines its visibility through mediation
	Aggregation	Inner SMC becomes resource of outer but without imposing the encapsulation (allows sharing)
	Fusion	Combines the interfaces, policies, and managed objects of two constituent SMCs into a new SMC
Task-Allocation Styles	Hierarchical Control	One top-level SMC controls the execution of a set of leaf SMCs
	Cooperative Control	One leaf SMC is controlled by a set of cooperating manager top-level SMCs
	Bidding	Cooperative task execution employing a negotiation approach (issuers and bidders)
	Collaborative	Fully decentralised interaction where SMCs can both load and receive tasks from their partners
Communication Styles	Shared-Bus	Provides a blackboard for decoupled event-based communication among SMCs
	Correlation	Individual events may be combined generating higher level events
	Diffusion	Provides a way of directly forwarding events to interacting SMCs
	Store-and-Forward	Useful in ad-hoc settings where SMCs may not have a permanent connection to their interacting partners

respect to interface exchanges, allowing one SMC some degree of control on the access of another SMC's interface (e.g. modification of invocation parameters or filtering of results). A particular combination of structural abstractions provides very specific properties for the interface exchange aspects of a collaboration.

B. Task-Allocation Aspects

Task-allocation aspects specify control-flow and task assignment between the elements in a structural relationship. We mentioned earlier that task allocation is achieved through the exchange of missions [20] among collaborating SMCs. However, these exchanges may rely on very specific abstractions. Some collaborations for example may rely on a single manager and multiple managed SMCs, while others may allow multiple managers to load tasks into a single SMC (which may require checking for and resolving conflicts between the loaded policies). Task loading may be uni-directional or bi-directional (in the latter, each SMC is both a manager and a managed node). Alternatively, tasks could be loaded according to a bidding strategy where SMCs express their willingness to receive tasks from an issuer. Finally, task loading may be conditional to a set of criteria, for example based on the capabilities provided by a SMC, its profile or the credentials that the SMC possesses.

C. Communication Aspects

Communication aspects deal with event-forwarding behaviour between SMCs. Abstractions vary from a simple *diffusion* of events from a source to a target SMC to a more elaborate *shared bus* between a set of SMCs that works as a sort of blackboard for shared events. Additional abstractions such as *correlation* of events provide flexibility in defining event patterns for triggering higher-level events. Abstractions for *store-and-forwarding* may be useful in ad-hoc settings where SMCs do not have a permanent connection to their interacting partners, and where events must be locally stored for subsequent delivery.

D. A Catalogue of Abstractions for Management Interactions

Based on these management perspectives, we have defined a catalogue of reusable abstractions (termed *architectural styles*) for management relationships among SMCs. Table I presents a brief overview of these styles. As in all such catalogues we cannot aim to be exhaustive but focus solely on the frequently occurring styles that facilitate the design and composition of complex SMCs by structuring the devolution of management responsibilities and corresponding interactions. Each architectural style relies on different abstractions for interface, policy or event exchange between the interacting SMCs. Due to space constraints, we cannot describe them individually here. Instead we aim to present how they are realised, how they work and how they are used. The next section will present our proposed interaction model for SMCs, which is used for combining architectural styles in complex SMC interactions.

VI. INTERACTION MODEL

Larger applications typically depend on the functionality provided by multiple collaborating SMCs, which must autonomously establish interactions with little or no user intervention. To facilitate these interactions, we use the concept of *roles* as placeholders for remote SMCs discovered at runtime [20]. Roles are kept in a *domain structure* that implements a hierarchical namespace similar to a file system in each SMC. A remote SMC is assigned to a role in another SMC if the former fulfills the requirements for that role (e.g. credentials, profile, capabilities). Policies previously associated with a role will then apply to SMCs assigned to it. This allows us to specify the interactions a SMC is expected to participate before it is discovered. The implementation of SMC discovery and role assignment were previously described in [20].

In this paper we propose an interaction model where instead of individual policies we associate entire architectural abstractions with a set of roles in the local domain of a SMC. Architectural styles encode template interactions that are individually instantiated among SMCs. Each template enforces a specific abstraction related to interface, event or policy exchange. The use of well-defined templates of interactions provides a better understanding of the relationships across SMCs and promotes reuse of common abstractions.

Each template defines its own set of *roles*, which refer specifically to the abstraction enforced. For example, a *composition* style defines the roles *outer* and *inner*, whereas a *hierarchical control* defines the roles *manager* and *managed* and a *diffusion* style defines the roles *source* and *target*. Additionally, template interactions are parameterised at instan-

Digital Object Identifier: 10.4108/ICST.PERVASIVEHEALTH2009.5979
http://dx.doi.org/10.4108/ICST.PERVASIVEHEALTH2009.5979

tiation, although the level of customisation allowed for each architectural style depends on the abstractions associated with it. This customisation may involve the parameterisation of *(a)* the methods to be filtered, mapped, etc (for a structural style), *(b)* the tasks and policies to be loaded and in what conditions (for a task-allocation style) or *(c)* the events to be forwarded or subscribed to (for a communication style). More formally we can define an architectural style as:

$$style = \langle Roles, Parameters \rangle$$

where *style* is an architectural style, *Roles* is the set of roles defined by this style and *Parameters* is the set of parameters that customise this style.

The configuration (or set-up) of the interactions that a SMC is expected to enforce is defined by the combination of *roles*, *architectural styles* and *bindings* defined in the local domain of this SMC:

$$domain = \langle Roles, Styles, Bindings \rangle$$

where *domain* corresponds to the domain structure of a SMC, *Roles* is the set of roles defined in this domain, *Style* is the set of architectural styles bound to these roles through the set of bindings *Bindings*.

A *binding* of a style with respect to a domain associates each *role* in the style with a *role* in the domain. Hence:

$$binding(style, domain) \iff \forall\, x \in Roles_{style}, \exists\, y \in Roles_{domain} :\ y := y \ \oplus\ x$$

where the operator $\oplus$ applied to two roles combines the behaviour associated with each role. A role in the local domain is typically bound to roles across the three layers of architectural styles. The roles defined in the local domain of a SMC will therefore combine the behaviour of the composite/overall interaction, based on the abstractions bound to them.

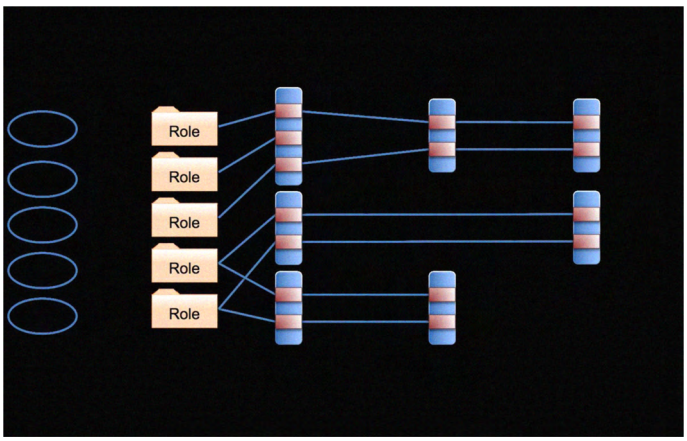

Fig. 3. Interaction model: (1) architectural styles are bound to *roles* in the local domain; (2) when remote SMCs are assigned to these roles, the semantics associated with each architectural style is enforced in the remote SMCs in the form of interface, event and policy exchanges.

Fig. 3 presents an outline of the interaction model for SMCs. Whenever a remote SMC is assigned to a *role* in the local domain, all architectural styles bound to this role will be *deployed* in the corresponding SMC. Hence:

$$assignment(SMC, role_{domain}) \iff \forall\, st \in Styles_{domain} : \forall\, x \in Roles_{st} : x \oplus^{-1} role_{domain} \rightarrow deployment(SMC, st, x)$$

where the operator $\oplus^{-1}$ returns *true* if two roles are *bound*. The *deployment* of a style in a SMC is defined by the implementation of the architectural style, in terms of exchanges of interfaces, policies and events and of the operations required to enforce the style semantics. The deployment operation takes as arguments a SMC, the architectural style to be deployed and the specific role in the style that this SMC will be playing.

This interaction model allows us to independently define layers of management where the structural, communication and task-allocation aspects of an interaction between a set of SMCs can be individually specified by reusing common abstractions expressed as architectural styles.

A. Patterns of Interaction

We aim to specify architectural descriptions where individual abstractions can be used as building blocks and arranged in particular settings. We call these *patterns of interactions*, although they are in essence complex architectural styles realised in terms of more primitive ones. Patterns exhibit very specific management properties with respect to interface access, task-loading and event-forwarding. For example, a body-area network is typically a *composition* encapsulating the sensors and mediating (and filtering) access to its internal components. It typically relies on a *diffusion* event-forwarding approach, where the sensors forward events to the PDA or mobile phone representing the patient SMC which will possibly run other tasks that make use of the information monitored. Similarly, a monitoring pattern between doctor and patient SMCs typically relies on a *peer-to-peer* structural approach as the notion of encapsulation (or ownership) does not apply to this interaction. Monitoring tasks are often loaded from the doctor into patient SMCs, and event *diffusion* may take place from patients to doctors.

We are therefore using *patterns* to represent very specific arrangements of management abstractions that are likely to occur often. Fig. 4 illustrates a succinct pseudo syntax for our architectural descriptions, where patterns can be specified and the management bindings between roles are defined (in our implementation patterns are written in Ponder2 which is more verbose). Each pattern defines a set of roles (amongst which some may be mandatory and others optional), and how they should be bound. Note that some bindings may be event triggered. When a pattern is instantiated, actual SMCs may be passed as parameters, or alternatively an *assignment policy* [4] may specify under which conditions a subsequently discovered SMC should be assigned to the respective role. A pattern is typically instantiated at a SMC, which will be responsible for enforcing the architectural styles defined in the pattern. Finally, note that large collaborations may be realised by

Digital Object Identifier: 10.4108/ICST.PERVASIVEHEALTH2009.5979
http://dx.doi.org/10.4108/ICST.PERVASIVEHEALTH2009.5979

```
//Collaboration specification
type pattern ⟨PatternName_0⟩(...) {

    import /factory/structural/composition;
    import /factory/taskallocation/hierarchical;
    import /factory/communication/diffusion;

    type pattern ⟨PatternName_1⟩(role R1, [mandatory] role R2, role R3) {

            bind style composition(outer R1,inner R2,inner R3);
            [on ⟨event⟩] bind style hierarchical(manager R1,managed R2,managed R3);
            [on ⟨event⟩] bind style diffusion(target R1,source R2,source R3);
    }
    inst pattern p_1 = ⟨PatternName_1⟩(SMC_a, SMC_b, [assignment_policy]) at SMC_1;

            ...

    type pattern ⟨PatternName_n⟩(...) {
            ...
    }
    inst pattern p_n = ⟨PatternName_n⟩(...) at SMC_n;
}
//Collaboration instantiation
inst pattern p_0 = ⟨PatternName_0⟩(...) at SMC_0;
```

Fig. 4. Pseudo syntax for the textual representation of an architectural description (notice that each architectural style must also be parameterised with the methods to be mapped or filtered, events to be forwarded or subscribed to, missions to be loaded, etc, however these details are not shown in the figure).

combining multiple patterns, and possibly patterns that contain other nested patterns.

VII. Revisiting the Health-Monitoring Scenario

Using this interaction model we can systematically elaborate our health monitoring example outlined in Fig. 2, by combining the management abstractions required to realise the collaboration. Fig. 5 shows a graphical representation of the scenario where specific abstractions were chosen for interface, event and policy exchanges. Each one of the five columns in Fig. 5 corresponds to one of the triangles in Fig. 2. The graphical notation only shows the overall view of the collaboration and the selected abstractions, but does not detail the parameterisation of each style (e.g. methods to be mapped or filtered in a structural style, events to be forwarded or subscribed to in a communication style, or tasks to be loaded and in which conditions in a task-allocation style).

The graphical representation shows that the patient and the two sensor SMCs (heart-rate and accelerometer) are bound through a *composition* structural relationship as this interaction requires *encapsulation* of the sensor SMCs (since we do not wish that the devices owned by a body-area network interact with body-area networks of patients nearby). Although the resources are encapsulated in the body-area network, operations for reading sensor measurements are typically mapped to the patient's interface. Also, sensors forward their events (e.g., the heart rate goes above a certain threshold) to the patient SMC through a simple *diffusion* event forwarding where each sensor plays the *source* role while the patient SMC plays the *target*.

The interaction between patient and doctor has different requirements. First of all, an encapsulation between patients and doctors would not apply, as a patient may interact with multiple doctors, as well as a doctor may interact with multiple patients. A *peer* abstraction is more suitable in this case, which leads to a simple exchange of interfaces but with no additional mapping, filtering or encapsulation. In terms of task-allocation, a doctor will typically specify policy loading strategies into the patient through a *hierarchical control* style. The tasks loaded are specified in the form of *missions* [20], which are used in the style parameterisation (e.g. an ECG monitoring mission). Similarly, the conditions when such tasks must be loaded are also part of the style parameterisation.

The example can be further elaborated with the specification of the interactions between the *home*, *server* and *patient* SMCs (in order to form the home environment monitoring interaction), the interactions between the *server* and *surgery* SMCs (in order to specify the data synthesis and report forwarding interaction) and interactions between *patient* and *surgery* (in order to specify the interaction which takes place in case of an emergency, e.g. the monitored data is assessed and a decision that a heart attack is imminent is taken).

VIII. Implementation

Architectural styles and patterns have been implemented using the Ponder2 framework. We have defined a library of styles divided into structural, task-allocation and communication categories. Styles are created from *factory objects*, which function as templates for creating any of the objects required by a Ponder2 application. Fig. 6 shows how a *composition* interaction is created from a factory object in Ponder2 syntax, how style-specific roles (*"outer"* and *"inner"*) are bound to roles defined in the local domain (*"Patient"*, *"SensorHR"* and *"SensorTemp"*), and how the style is parameterised with style-specific properties: in this case,

Digital Object Identifier: 10.4108/ICST.PERVASIVEHEALTH2009.5979
http://dx.doi.org/10.4108/ICST.PERVASIVEHEALTH2009.5979

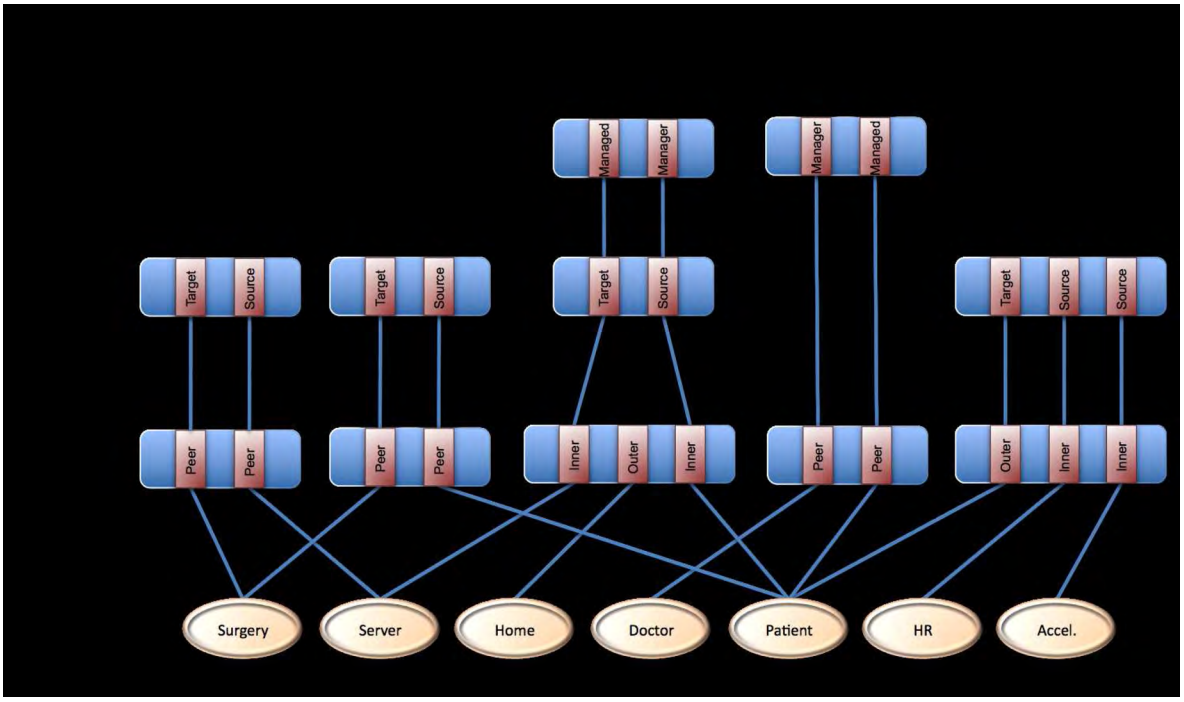

Fig. 5. Architectural representation of the health monitoring scenario: the illustration shows the overall configuration of the collaboration with the selected abstractions, but it does not detail how each architectural style is parameterised.

the operations *"SensorHR.read"* and *"SensorTemp.read"* are mapped to the operations *"readHR"* and *"readTemp"* exported by the interface of the SMC assigned to the *"Patient"* role (the encapsulation enforced by the composition style will hide the sensors from external SMCs, keeping them as managed resources of the SMC assigned to the *"Patient"* role, but the selected operations will be mapped to the latter's interface).

```
comp := /factory/structural/composition create.
comp outer: "Patient".
comp inner: "SensorHR".
comp inner: "SensorTemp".
comp map:   "SensorHR.read"   to: "readHR".
comp map:   "SensorTemp.read" to: "readTemp".
```

Fig. 6. Code for the instantiation of a composition style between a *patient* and two *sensors*, written in Ponder2 syntax.

Architectural abstractions, either simple architectural styles or more complex patterns, are bound to roles in the domain structure of a SMC and instantiated when remote SMCs are discovered and assigned to these roles. The implementation of architectural styles relies on the functionality provided by the SMC's *management interface*, which is application-independent and common to all SMCs. The management interface provides operations for interface exchange and binding, and interface mapping and filtering. It also provides operations for exchange and subscription of events, as well as operations for exchange of missions. The implementation of each style thus defines a specific script of operations to be executed on the participant SMCs using the common functionality provided by their management interfaces (Fig. 7). This ensures that the semantics specified by a style is enforced in the involved SMCs. Note that there are dependencies among the architectural styles: structural styles must be deployed first, as they enable the exchange of *application interfaces* (e.g. *doctor* interface, *patient* interface) [20]; communication styles are then deployed to define patterns in terms of the events provided by these interfaces; task-allocation styles must be the last, as the policies loaded may depend both on the operations provided by the application interface, as well as on the events forwarded by a communication style.

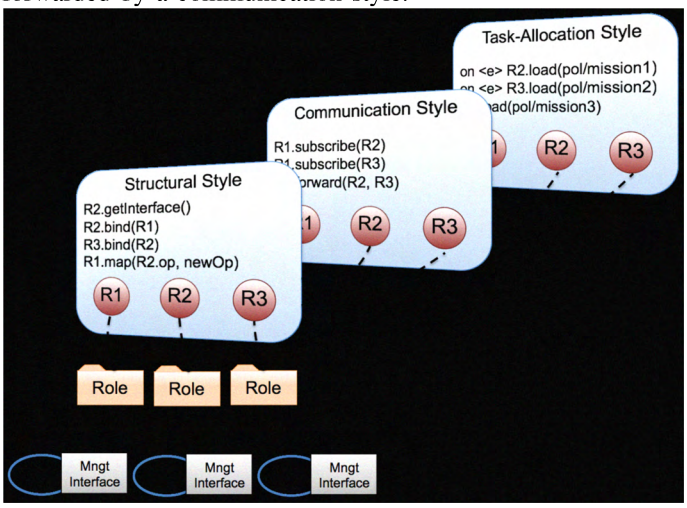

Fig. 7. The implementation of each architectural style defines the operations and relies on the functionality provided by the management interface of the participant SMCs. These operations are executed when actual SMCs are assigned to roles.

IX. Discussion and Concluding Remarks

Architectural styles provide a better understanding of the relationships across SMCs and allow us to specify the management of large-scale composable systems by reusing building

Digital Object Identifier: 10.4108/ICST.PERVASIVEHEALTH2009.5979
http://dx.doi.org/10.4108/ICST.PERVASIVEHEALTH2009.5979

block abstractions. Whilst we have identified a catalogue of architectural styles that addresses individual abstractions for exchange of interfaces, tasks and events, more complex collaborations are built systematically from simpler ones by combining styles as design elements of an interaction. We have defined an interaction model for combining architectural styles in patterns of interactions that are enforced by different SMCs in a large collaboration. In addition, collaborations of SMCs may interact with other collaborations, and reapply the styles recursively. We demonstrated our interaction model in a scenario for pervasive healthcare but we are applying the same principles to the management of communities of unmanned autonomous vehicles (UAVs) [4].

The different categories of architectural styles can be seen as alternative *perspectives* for modelling the management of SMC interactions, which cater for their structural aspects, task-allocation and communication flow/sharing. We do not claim the catalogue presented in this paper corresponds to a complete list of interaction possibilities, but focus solely on the frequently occurring styles that facilitate the design and composition of complex SMCs. We expect that the investigation of further scenarios will lead us to the identification of additional abstractions for SMC interactions as many styles and patterns of interaction are specific to the application domain. Although we have focused on the SMC framework, the proposal of a catalogue of architectural styles for modelling interactions between autonomous entities aims to benefit the engineering process of pervasive and autonomous systems in general, making such systems more flexible, robust and reusable.

The scenarios and applications that we consider are intrinsically dynamic, and rearrangements of architectural styles during run-time may be necessary due to changing conditions. This will require collaborating SMCs to achieve a secure state before their interaction configuration can be modified. Much work has been done on dynamic architectures [21], but we still have to further investigate run-time structural reconfiguration in our model. In the near future we plan to formalise the operational semantics of the interaction model presented in this paper. This will allow the verification of the overall consistency of the specification and the checking of properties achieved by a specific arrangement of architectural styles.

ACKNOWLEDGMENTS

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*, May 2007.

[2] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "A middleware infrastructure for active spaces," *IEEE Pervasive Computing*, vol. 1, no. 4, pp. 74–83, 2002.

[3] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed., ser. Professional Computing Series. Addison-Wesley, 1995, 416 pages.

[4] A. Schaeffer-Filho, E. Lupu, M. Sloman, S.-L. Keoh, J. Lobo, and S. Calo, "A role-based infrastructure for the management of dynamic communities," in *Proceedings of the 2nd International Conference on Autonomous Infrastructure, Management and Security (AIMS)*, ser. LNCS. Bremen, Germany: Springer, July 2008, pp. 1–14.

[5] D. Garlan and M. Shaw, "An introduction to software architecture," in *Advances in Software Engineering and Knowledge Engineering*, V. Ambriola and G. Tortora, Eds. Singapore: World Scientific Publishing Company, 1993, pp. 1–39.

[6] N. R. Mehta, N. Medvidovic, and S. Phadke, "Towards a taxonomy of software connectors," in *Proceedings of the 22nd ACM International Conference on Software engineering (ICSE)*, New York, NY, USA, 2000, pp. 178–187.

[7] R. Allen and D. Garlan, "Beyond definition/use: architectural interconnection," in *Proceedings of the ACM Workshop on Interface Definition Languages*, New York, NY, USA, 1994, pp. 35–45.

[8] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik, "Abstractions for software architecture and tools to support them," *IEEE Transactions on Software Engineering*, vol. 21, no. 4, pp. 314–335, 1995.

[9] J. Kramer, "Configuration programming - a framework for the development of distributable systems," in *Proceedings of the IEEE International Conference on Computer Systems and Software Engineering (CompEuro)*, May 1990, pp. 374 – 384.

[10] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer, "Specifying distributed software architectures," in *Proceedings of the 5th European Software Engineering Conference*. London, UK: Springer-Verlag, 1995, pp. 137–153.

[11] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann, "Specification and analysis of system architecture using rapide," *IEEE Transactions on Software Engineering*, vol. 21, no. 4, pp. 336–355, 1995.

[12] G.-C. Roman and J. Payton, "Mobile unity schemas for agent coordination," in *Proceedings of the 10th International Workshop on Abstract State Machines*, ser. LNCS, vol. 2589. Springer, March 2003, pp. 126–150.

[13] P. C. Clements, "A survey of architecture description languages," in *Proceedings of the 8th IEEE International Workshop on Software Specification and Design (IWSSD)*, Washington, DC, USA, 1996, p. 16.

[14] V. Hilaire, A. Koukam, and S. Rodriguez, "An adaptative agent architecture for holonic multi-agent systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 3, no. 1, pp. 1–24, 2008.

[15] F. Zambonelli, N. R. Jennings, and M. Wooldridge, "Developing multiagent systems: The gaia methodology," *ACM Transactions on Software Engineering Methodologies*, vol. 12, no. 3, pp. 317–370, 2003.

[16] Y. Aridor and D. B. Lange, "Agent design patterns: elements of agent application design," in *Proceedings of the 2nd ACM International Conference on Autonomous Agents*, New York, NY, USA, 1998, pp. 108–115.

[17] M. Kolp, P. Giorgini, and J. Mylopoulos, "A goal-based organizational perspective on multi-agent architectures," in *Revised Papers from the 8th International Workshop on Intelligent Agents*. London, UK: Springer-Verlag, 2002, pp. 128–140.

[18] D. Deugo, F. Oppacher, J. Kuester, and I. V. Otte, "Patterns as a means for intelligent software engineering," in *Proceedings of the International Conference on Artificial Intelligence (IC-AI)*, vol. 2, 1999, pp. 605–611.

[19] R. G. Smith, "The contract net protocol: high-level communication and control in a distributed problem solver," *Distributed Artificial Intelligence*, pp. 357–366, 1988.

[20] 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.

[21] J. Kramer and J. Magee, "The evolving philosophers problem: Dynamic change management," *IEEE Transactions on Software Engineering*, vol. 16, no. 11, pp. 1293–1306, 1990.

Digital Object Identifier: 10.4108/ICST.PERVASIVEHEALTH2009.5979
http://dx.doi.org/10.4108/ICST.PERVASIVEHEALTH2009.5979