Towards a generic architecture and methodology for multi-goal, highly-distributed and dynamic autonomic systems

Sylvain Frey[1,2], Ada Diaconescu[2], David Menga[1] and Isabelle Demeure[2]
[1]*ICAME department, EDF R&D, Clamart, France;* {*first name*}.{*last name*}@*edf.fr*
[2]*Télécom ParisTech, CNRS LTCI, Paris, France;* {*first name*}.{*last name*}@*telecom-paristech.fr*

Abstract

Autonomic control is vital to the success of large-scale distributed and open IoT systems, which must simultaneously cater for the interests of several parties. However, developing and maintaining autonomic controllers is highly difficult and costly. To illustrate this problem, this paper considers a system that could be deployed in the future, integrating smart homes within a smart micro-grid. The paper addresses this problem from a Software Engineering perspective, building on the authors' experience with devising autonomic systems and including recent work on integration design patterns. The contribution focuses on a generic architecture for multi-goal, adaptable and open autonomic systems, exemplified via the development of a concrete autonomic application for the smart micro-grid. Our long-term goal is to progressively identify and develop reusable artefacts, such as paradigms, models and frameworks for helping the development of autonomic applications, which are vital for reaching the full potential of IoT systems.

1 Introduction

The purpose of any computing system is to reach objectives specified by an external authority. When multiple authorities can access the system, like in the IoT (Internet of Things) context, system goals may be conflicting, while targeting overlapping system parts. Moreover, such systems must often scale to large numbers of highly-distributed resources and be adaptable to changes in their goals, execution context and constituent resources (the systems are open). Autonomic or self-* capabilities become key to the success of such systems.

This paper illustrates this challenge via a multi-goal, adaptable and open autonomic system that integrates several smart houses into a smart micro-grid. To cover both the Autonomic Computing (AC) and IoT domains in this example, the paper employs the generic term *autonomic control* [28] to designate the system logic that manages available resources for attaining goals. The only means for an autonomic controller to pursue its objectives is via actions it can perform on such manageable resources. To select actions the controller can rely on decision strategies, knowledge and runtime information from the environment and the system state. The **key challenge** lies in developing the controller logic that can successfully pursue system goals while ensuring essential system characteristics - scalability, robustness, adaptability and openness. We approach this challenge from a Software Engineering (SE) perspective. Our aim is to identify, specify and develop reusable artefacts for analysing and designing autonomic control systems with the aforementioned properties. The presented work relies on our experience with building autonomic frameworks [9][10][20][23]. The long-term aim is to build a comprehensive **reference architecture for autonomic systems**.

The generic architecture proposed is constructed on the assumption that the development and adaptation of any realistic autonomic system will rely on the *integration* of managed resources and control elements of different types; integration can occur statically or dynamically. An important challenge lies in identifying and bringing together the necessary types of *abstract architectural artefacts* and concrete *control elements* that can be used for system design and integration. Abstract artefacts can include architectural styles, design patterns and layering techniques over several axes of abstraction. Control elements include relatively straightforward control tasks - such as monitoring, decision-making, execution or knowledge-management; entire control loops; or combinations of the above [15][23]. They can be functionally organised based on well-defined abstract entities, like those indicated above, and interconnected via hard-coded or loosely-coupled bindings. The overall integration process can be controlled in a fully centralised, decentralised or hierarchical manner [15][9][10][14][23].

Another important challenge lies in coordinating control elements for obtaining coherent controllers that can pursue several goals, adapt and support highly-distributed, plug-and-play resources. Of major interest here is the detection and resolution of *conflicts* that may occur when integrating elements with contradicting goals [10] or control strategies [23]. The generic architecture and methodology presented here focus on addressing these two major challenges. Other important concerns, such as timing and synchronisation of integrated actions are part of ongoing research not covered here. The authors do not claim the novelty of all artefacts in the architecture. Indeed, most of these can be found in related fields such as automatic control [24], collective adaptive systems [18], multi-agents [14][16], robotics [6], cybernetics [2] or autonomic systems [15][7][19]. These provide a rich repertoire of solutions that address different parts of the overall challenge.

This paper's contribution consists in identifying and extending existing artefacts that can be used for designing autonomic control systems, and assimilating them into a coherent framework. Some **key aspects of the proposed contribution** include: rendering explicit the conceptual elements included in goal definitions; defining the problem of building autonomic controllers as one of mapping declarative actions (goals) into concrete actions (on managed resources), in a context-aware and extensible way; combining existing SE techniques for splitting the mapping problem into recursively smaller elements and integrating such elements into flexible overall solutions; defining integration conflicts and ways of resolving them; applying architectural templates and agent organisation techniques to ensure system coherence and runtime flexibility. This is illustrated by developing a multi-goal, adaptable and open smart micro-grid.

The ongoing aim is to help answer questions on:

- How to develop scalable and adaptable feedback loops?
- How to integrate multiple feedback loops for pursuing many goals at different scales?
- How to deal with system dynamism and openness?

Addressing these concerns is vital for reaching the full potential of Autonomic Computing and IoT paradigms. We emphasise the fact that we do not propose a concrete ready-to-use architecture; this would have to be domain-specific. Rather, we provide an abstract architecture and methodology that can guide the design process of autonomic control systems in various domains. The proposed contribution is relevant to both autonomic systems in general - as it helps design multi-goal, distributed and adaptive autonomic managers; and to IoT systems - as it shows how autonomic controllers built in this way can control system resources to ensure required properties and functions.

Section 2 describes the sample smart micro-grid application, with its requirements and design challenges. Sections 3 and 4 introduce the conceptual and design aspects of the proposed architecture, respectively, illustrating them via concrete examples from the smart micro-grid. Section 5 illustrates the complete design of the prototype application. Section 6 discusses related work and section 7 concludes the paper and indicates future research.

2 Smart Houses meet Smart Micro-Grid

2.1 Overall system

In a near-future, it can be envisaged that smart homes integrate with smart grids to form large-scale, highly-distributed, dynamic and open IoT systems. This paper considers this type of system as a relevant use case for the problem addressed. For the sake of clarity and expressiveness, the system model is often kept simple, neglecting important aspects such as business models, legal regulations or fine-grain grid behaviour.

Smart homes are seen here as cyber-physical systems that integrate and control electrical devices in order to provide automated services, such as context-aware heating, entertainment, lighting and security. Individual devices termed "smart" embed their own control logic to offer some service. For instance, a thermostat can turn itself up when detecting the home owner's presence.

A *micro-grid* is a local, low-tension electrical network. For simplicity, this paper considers a residential district organised as a tree, rooted at the district aggregator; the leaves are the end-user appliances - producers (e.g. solar panels), consumers (e.g. electrical heaters) or both (e.g. batteries). The generic term *prosumer* designates such endpoints; the associated term *prosumption* means either production or consumption. A residential tree is part of a city grid that is in turn part of the national grid (not considered here). A house grid is a sub-tree of the district grid. Its prosumption is measured by a house meter and equal to the sum of prosumptions of all appliances in the house. Likewise, the district's prosumption is the sum of all household prosumptions.

The *load* of a grid is defined as the ratio between productions and consumptions. It is said to be *high* when consumptions overshoot productions, hence requiring consumption from the parent grid; *low* load denotes the opposite. In this paper, load management consists in adjusting local productions and consumptions to minimise the footprint on the parent grid. For simplicity, the paper will globally refer to the "smart micro-grid" including implicitly the integrated smart houses.

2.2 Autonomic control requirements

Let us now define the perimeter of the smart grid's autonomic controller and identify its most important requirements. First, the controller must pursue several goals, specified by different authorities. The electricity provider imposes load management goals for the grid. In the presented scenarios, these goals take the form of a Goal Power (GP) interval - $[GP_{low}, GP_{high}]$ - within which the prosumption of a sub-grid should be maintained. The exact values will depend on business objectives at different grid scales and on the context. Home owners define different types of goals for their households. These may be related to comfort - like maintaining a temperature (heaters) or a lighting ambiance (lamps), or simply performing activities like washing (washing machine) or cooking (oven). They may also be related to cost - like minimising the electricity bill or the environmental impact. Note that such goals can be in conflict.

Hence, the autonomic controller must be able to either favour one goal over all others - like prioritising energy savings over appliance usage or conversely pursuing comfort at any cost; or target a compromise among all goals - like only ensuring comfort partially if the grid is highly loaded. Such preferences are specified by administrative authorities and may be context dependent (e.g. user presence or weather). Finally, some preferences can be overridden implicitly as users handle appliances directly (e.g. turning-up a heater or cooking).

The autonomic controller must pursue its goals by performing actions on manageable resources, including grid resources (not discussed here) and electric appliances. The presented use case focuses on two sample appliances with specific profiles. First, heaters transform electric energy into heat; their power can be monitored and set via specific touchpoints. Second, lamps transform electric energy into light; their light intensity can be adjusted via specific touchpoints that measure and set their consumption. While lamps do not usually constitute significant consumers, they are used here to model diverse equipments with similar profiles, such as microwave ovens or vacuum cleaners. Finally, privacy concerns impose that house appliances cannot be controlled from outside the house within which they reside.

In addition to meeting the goals, the autonomic controller must scale to large numbers of highly-distributed resources (e.g. appliances). Also, the controller must adapt to changes in goal specifications (e.g. power intervals), priorities (e.g. comfort vs. savings) and execution context (e.g. weather). Finally, it must handle "smart" or standard appliances being plugged-in or out.

3 Conceptual Model

3.1 Goal types and specifications

Goals represent the very purpose of autonomic systems. Generally, they define a system's *viability zone*, within which its state must be included at any one time [15][1][2]. A system's state is defined via a set of variables whose values can predict its behaviour in the near future [24] (e.g. a heater's power setting predicts the amount of heat it will produce). A system's state can also represent its end goal (e.g. a targeted temperature). Goal definitions are intimately related to the way in which they can be evaluated - typically via observations on system state variables. Goals may be declarative or procedural [17]. Declarative goals indicate *what* should be achieved rather than *how*. They are usually defined as constraints on system variables, delimiting the viability zone, and can be evaluated automatically via a utility function over the system state. Procedural goals indicate (via high-level policies) *how* the system should behave in various situations. This paper focuses on declarative goals, considering that procedural goals can be induced from these.

A goal definition can include three types of elements - **G (V, S, T)**, where V defines the viability zone, S the resources to which it applies, and T the periods over which it applies. The viability constraints (V) are compulsory and typically accompanied by a utility function for evaluation purposes. In the smart home example, a goal can define a viability interval for the power consumption. The Scope element (S) separates the viability definition from the resource domain to which it is applied (and evaluated). It is defined via domain constraints that identify, in a declarative way, the system resources targeted at any one time. In cyber-physical systems, such as IoT, Scopes can represent physical areas in Euclidian space; resources located in that area belong to the Scope. For instance, a goal defining a temperature interval can be applied to the scope of a house or only of one room. It will be evaluated using thermometers located across the house or within that room, respectively. In systems where physical space is less relevant Scopes can define other types of resource sets - e.g. a network domain in a computer cluster. Scopes are particularly relevant to open systems, where resources can change dynamically and unpredictably. For example, a power interval can be defined for a house, without explicitly identifying all its appliances. Finally, the Time element (T) separates a goal's viability constraints and scope from the periods over which they take effect. For simplicity, this element is no further developed in the paper; goals implicitly start when received and end when cancelled or overwritten.

3.2 Goal achievement and evaluation

The only means for an autonomic system to attain its goal(s) is via actions it can perform on manageable resources [Fig. 1]. Namely, an autonomic controller should act so as to influence the variables of resources within the goal's scope (S_G) to maintain them within the goal's viability zone (V) - e.g. to pursue a temperature goal in a home, a controller acts on the heaters available in that home. It can be noted here that the set of resources on which the controller acts - *action resources* - is not necessarily equal to the set of resources in the goal's scope - *goal resources*. The only constraints are that the controller should be able to monitor goal resources for evaluating its goal; and that the action resources should have a controllable influence on the state of goal resources. Considering a temperature goal in one room: the room's atmosphere is the goal resource, since its temperature is monitred and evaluated; heaters in the room and in neighbouring rooms are action resources, since the controller acts upon them to *influence* the room temperature. A controller's action resources for pursuing a goal constitute its *Action Scope (S_A)*. The set of resources whose state they influence is referred to as *Influence Scope (S_I)*. Finally, the controller may monitor resources it cannot control - *context resources* from a *Context Scope* - e.g. outdoor thermometers. In summary, a controller pursuing a goal $G(V, S_G)$ will act on resources in an action scope S_A to influence resources in a scope S_I, where S_I includes the resources in S_G that are monitored to evaluate the goal [Fig. 1].

This approach clearly separates a goal's definition from the controller's means to pursue it. This is vital for adapting a controller's strategy to changes in its goals, environment and internal resources. It can also intervene in tackling multi-goal conflicts, as discussed later. This conceptual setting allows formulating the problem that an autonomic controller must solve - i.e. how to attain its goal(s). It consists in finding a strategy, or *mapping function*, which can **transform goals into concrete actions**; the solution will *be sensitive to the external context and internal system state*. This view generalises the notion of *goal to represent a higher-level declarative action* (intentional) that must be translated into *concrete actions* (A), executed via resource effectors (imperative) [15].

3.3 Goal translation and division

This subsection identifies the main factors behind the difficulty of mapping goals into concrete actions and indicates the structural and behavioural concepts that can help analyse and address them. **One factor** stems from an increasing "distance" - or difference in the *abstraction levels* - between the goal's viability specification (V) and

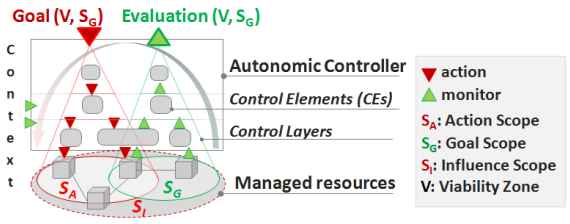

Figure 1: Goal projection and evaluation.

the concrete actions (A). E.g., in the smart house, a controller must map a "comfort" goal into concrete power configurations on heaters. **A second factor** represents a typical control problem, involving complicated *decision-making* capabilities that rely on partial knowledge, react to fluctuating inputs, avoid oscillations and optimise results. **A third factor** intervenes as controllers must *adapt* to change - e.g. integrate plug-and-play resources and change strategies to achieve evolving goals in a variable environment. **The fourth factor** stems from the *scale* of goal scopes (S_G). A large-scale S_G often implies a comparably large-scale S_A which is difficult to control, especially in an open context. This difficulty increases when plug-and-play resources are heterogeneous and belong to different legal authorities.

"Classic" Software Engineering (SE) techniques can be applied to help address these factors. **Layering** can structure controllers along three distinctive axes. First, *abstraction layers* can progressively translate goals into concrete actions (abstraction factor). Each layer maps higher-level goals (or actions) from the layer above into lower-level goals (or actions) for the layer below [Fig. 1]; this results in a *translation hierarchy*. E.g., a "comfort" goal is translated into an intermediary "temperature" goal and then into a concrete "power" configuration. This controller has two abstract layers: the highest layer pursues the "comfort" goal (declarative); it acts by setting a "temperature" goal (declarative) on the lower layer, which acts by setting a "power" goal (imperative) on a heater. Goal evaluation follows the inverse translation path - monitored data from S_G resources (e.g. temperature) is translated into concepts of the administrative domain (e.g. comfort). Abstract layers are said to implement *base-level* functions, meaning to pursue goals by acting on resources. Second, *control layers* can be introduced to add meta-control abilities to such base-level functions, hence enabling controllers to self-adapt (adaptation factor). E.g., when a smart house's goal changes from "comfort" to "saving" mode, a meta-control layer can adapt the behaviour of control elements in the base layer, as necessary to pursue the new goal. Third, *integration layers* can be added to form a control hierarchy that helps integrate decentralised control elements (be-

longing to either abstract or control layers, as discussed below). As integration and abstract layers are often superposed in real system designs the terms are at times interchanged in the paper.

Encapsulation and modularisation techniques can complement layering to address a controller's complexity and adaptability concerns (decision-making and adaptation factors). They enable the separation of concerns in the controller's logic, facilitating the reuse and integration of simpler control elements (CEs) into complicated controllers. This is the equivalent of *splitting* the controller's mapping function into complementary parts. Adaptation can be achieved by replacing or reintegrating these parts. Domain-specific algorithms are necessary for implementing CEs and are outside the paper's focus. This technique can also be applied to split the goal's scope (scalability factor). Here, a goal (G) is split into complementary goals (G_i) that define the same type of viability constraints (V) over smaller scopes (S_{Gi}). Each G_i is assigned to a different control element CE_i. E.g., a comfort goal for a house is split into comfort goals for individual smart devices; or, the power goal over a district grid is split into power goals for different houses - here, the goal value for the district power constraint is also split into smaller values for each house grid. This approach can also address the multi-authority issue. E.g., district controllers (owned by a provider) split their goals among house controllers (owned by private parties). It also intervenes in goal translation to address resource heterogeneity. E.g., comfort is converted into temperature for thermostats, and into light intensity for lamps. **Loose-coupling and dynamic binding** enable runtime integration of CEs into adaptable controllers.

From a behavioural perspective, most CEs in the aforementioned structures act only in response to incoming data, like monitoring, analysis or action, from resources, other CEs or administrators. In an integrated system, CEs trigger each others' executions thus generating a *control flow* through the system [Fig. 2]. The control flow can pass through CEs within a single layer, like the MAPE elements of a control loop; as well as across layers, like a base-control loop triggering a meta-control loop or an integrator. This is an important concept and plays a key role in identifying and resolving conflicts. When a controller pursues a goal, we say that its control flow *serves* the goal or *carries* the ensued action(s).

3.4 Multi-goals, conflicts and resolution

Most autonomic systems will have to follow multiple goals, given by one or several authorities. In some cases, multiple authorities issue goals with the same type of viability constraints (e.g. range of power values) but with different constraint values (e.g. [1 kW, 2 kW] and [1.5 kW, 3 kW]). In another case, a single authority issues goals with different constraint types (e.g. comfort and power savings). The two cases can be combined.

Each goal can be addressed individually as discussed before. The solutions can then be combined to obtain multi-goal systems. The main additional problem intervenes when the system's goals are in *conflict*. This concept must be defined before addressing the problem. At the lowest system level, a conflict occurs when concrete actions attempt to change a resource's variables to incompatible values - e.g. one action turns a heater's power up and another one down. In most cases, conflict causes can be traced through the system to various sources. Source causes can stem from conflicting goals, conflicting controller strategies, or both of the above. Goals are conflicting when they define contradictory viability constraints over overlapping goal scopes (e.g. different power intervals over the same house grid). Control strategies are conflicting when they carry contradictory actions through overlapping influence scopes (S_I) (e.g. openning a window during a cold evening to pursue an air-freshness goal influences the room temperature, causing heaters to power-up and hence jeopertise a power-saving goal). Hence, conflicts *may* occur when goals can cause contradictory actions on overlapping S_Is, the intersection area being referred to as *Conflict Zone* [Fig. 2]. Concretely, conflicts *do* occur when control flows that service contradictory goals (or carry contradictory actions) pass through a conflict zone (within a certain period, which is not discussed here). To avoid such behaviour, conflict zones must be identified and special-purpose mechanisms placed in the CEs within those zones. These include conflict-resolution design patterns [10] or agent-like CEs that can compromise among goals (subsection 5.1). Several of these can be placed along conflicting control flows to improve the robustness of the resolution process [Fig. 2].

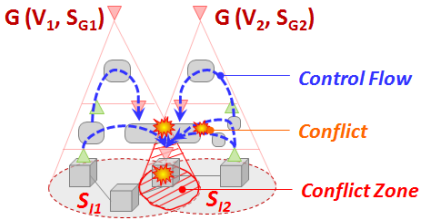

Figure 2: Conflicts and resolution.

4 Generic Architecture

4.1 Architecture Overview

We can rely on the abstract concepts discussed to define a generic architecture for designing autonomic control systems. To remain generic, the proposed architecture is a logical one, relying on and refining the Autonomic Computing Blueprint [15]. Here, an autonomic system consists of: *managed resources*, which can be acted upon and monitored; and an *autonomic controller*, which receives and pursues goals. We will show how controllers can be designed based on the layering, modularisation and loose-coupling techniques. The controller's abstract architecture will be customised and instantiated case-by-case resulting in application-specific designs. This process is highlighted in subsection 4.5 - 'methodology'.

Typically, an autonomic controller is divided into application-specific **abstract layers**, to bridge the conceptual gap between administrative goals and managed resource parameters. Each abstract layer can be enriched with one or several **control layers**, depending on the adaptability and goal-management needs of its control function (discussed below). **Modularisation** is introduced by splitting layers (abstract or control) into *control elements* (CEs) of various types (discussed below).

Loose-coupling enables the flexible *integration* of control elements (CEs) both within and across layers. Integration can be performed statically or dynamically, to initially develop and then adapt the controller. Hence, **integration layers** can be added to coordinate the disparate actions of CEs within abstract or control layers. Integration layers can also be modularised via CEs. To complicate things, integration and control layers can be further split into abstract layers and/or augumented with additional control layers.

In a generalised view, abstract, control and integration layers are conceptually orthogonal. In principle, any layer type can be divided recursively or added on top of layers of any other type. All layers are composed of CEs that can be added, updated or removed during runtime. In reality, particular combinations of layers and CEs types will most likely emerge in various application domains.

4.2 Types of layers and control elements

More concretely, CEs can represent: i) *control tasks* - control-related functions (e.g. monitoring, decision, execution, knowledge management, other atomic functions or combinations of these, as shown in [23]); ii) *integration tasks* - integration-specific functions (e.g. application-specific conflict resolution, as detailed in [10]); and iii) *control composites* - flexible compositions of control tasks and (optionally) integration tasks, for providing more advanced control structures and functions (e.g. single or integrated feedback loops). Control composites can or not be *encapsulated*. When encapsulated, they allow building fractal-like structures, which appear from the outside as a single well-integrated CE [Fig. 3], hence identical to a control task.

For instance, a base abstract layer that controls a heater to reach a temperature may consist of four control tasks: monitoring the temperature, analysing it with respect to a target, planning a power ajustement and executing it on the heater. If alternative planning tasks are available, an integration task can be added to select the best plan to use in each context. The integrated tasks form a complete control loop that can represent a control composite (encapsulated or not). This composite can be integrated with similar composites controlling other heaters and coordinated via an additional integration task.

From a behavioural perspective, CEs may be:

- *reactive* - acting in response to external stimuli; reactions can be stateless or using internal knowledge;
- *self-adaptive* - able to modify its reactions in response to changes;
- *agent*[1]*-like* - managing and negotiating goals given by other entities; only accepted goals are pursued.

To achieve such behaviours, the proposed architecture defines **three types of control layers**. Each CE may include one or several of these layers, in order to display a more-or-less sophisticated type of behaviour. First, a *base control layer* monitors and acts on managed resources following a pre-selected control strategy; it enables reactive behaviour. Second, a *meta-control* or *adaptation layer* ensures the base layer's adaptation to change, by altering or fine-tuning its control strategy [2][19]; it enables self-adaptive behaviours. Finally, the *goal management layer* receives requests for pursuing goals and decides whether or not to accept them; it enables agent-like behaviours, which are critical for conflict resolution. An agent's decision may be binary or more nuanced, based on the new goal's requester authority and conflicts with already accepted goals.

As indicated above, **abstract layers** for goal translation are application-specific - hence, no generic types were identified. Concerning **integration solutions**, several applicable designs were identified in [10] in the form of integration design patterns (e.g. hierarchy, controller, stigmergy or cooperation). While abstract, control and integration layers can be separated conceptually, in reality they can often be implemented within the same application layers of CEs (exemplified in section 5).

4.3 Requirements for integration

Integrating CEs must rely on standardised interfaces and protocols. While the details of these are domain-specific, their general semantics and purpose can be identified. This view is compliant with the Autonomic Computing Blueprint [15], but extended from control loops to all CEs [Fig. 3]. Hence, from an external view, CEs are quite similar. They *require* monitoring and action interfaces for accessing managed resources, including CEs in lower layers. They also *provide* monitoring and action interfaces for giving access to administrators and CEs in higher layers. These interfaces are the main enablers for CE layering and integration. Their semantics will differ depending on the CE type and conceptual layer - e.g., monitoring and execution touchpoints for control tasks in a base-control layer; and, goal specification and evaluation touchpoints for control loops in an adaptation layer. Their implementations will also differ - e.g., reactive CEs simply execute incoming actions, while agent-like CEs may execute, negotiate, or ignore them.

CEs may also provide and require functional interfaces for exchanges with other CEs [Fig. 3]. As before, these exchanges are application-specific, but their general purpose will depend on the CE's function - e.g. in the base-control layer, they can enable the integration of control tasks into feedback loops; in the self-adaptive layer, they can provide access to search and discovery services; for agent-like CEs, they can intervene in agent negotiation and self-organisation. Depending on its use, a CE may or may not provide all of these interfaces.

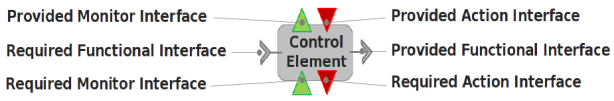

Figure 3: Control Element interfaces.

4.4 Integration and adaptation

Integrating CEs into multi-goal, distributed and adaptable autonomic controllers requires handling problems of communication, coordination and control. The proposed architecture identifies several integration-specific CEs to help with such issues. Essential elements include distributed communication infrastructures, discovery and repository services. Many such artefacts are available from related research domains and so not treated here. Conversely, coordination and control are key concerns that are especially challenging when CEs and resources must be integrated dynamically.

Two **complementary techniques** can be adopted to address these key issues. The first one relies on facilities for runtime evaluation, reporting and autonomic adaptation. These allow an autonomic controller to evaluate itself and adjust its internal composition accordingly, so as to remain within the viability zone. Support for this aspect was included in the conceptual model and will be further developed in future work. The second technique (developed here) relies on imposing architectural templates or *organisations* - a term borrowed from the agent community [14][26][29]. An organisation defines an *invariant system core*, or *template*, which can be "filled-in" dynamically with concrete resources depending on their availability and state. Imposing an organisation can ensure, to a certain extent, structural and behavioural properties for the resulting adaptive system [9].

An organisation consists of several *roles* that interact in a well-defined way. A role is defined as a set of well-specified capabilities - e.g., a "prosumer" role in a smart grid organisation. The role can be assigned (statically or dynamically) to any concrete resource that provides those capabilities - e.g., a heater takes the prosumer role. Based on the proposed architecture, autonomic controllers are designed as one or several organisations composed. Each organisation is defined based on the generic artefacts in the architecture, including layers and CEs. The smart micro-grid exemplified below shows how organisations designed in this way facilitate system development and adaptation. A catalogue of reusable organisations can be progressively developed. A core set was presented in [10] as integration design patterns. These include centralised orchestrators, decentralised coordination via functions embedded in CEs, hierarchical multi-layer organisations, aggregators and filter interceptors for integrated control flows.

4.5 Development methodology

We propose a certain sequence of *indicative steps* for developing autonomic control systems based on the generic architecture proposed:

1. specify system **goals**, defining their viability constraints and scopes - $G_i(V_i, S_{Gi})$; the authorities involved are also identified here.

2. identify the managed resources and the concrete actions that can be executed on them to influence the state of resources in S; these will make-up the controller's **Action Scope** (S_{Ai}).

3. identify 'relevant' resources that fall into the **Influence Scope** (S_{Ii}) of this Action Scope (S_{Ai}).

4. identify the context resources that can be monitored and influence control decisions; these will form the controller's **Context Scope** (S_{Ci}).

5. for each goal, define the main **abstract layers** and their control elements (CEs) required goal translation and splitting.

6. define an initial **integration infrastructure**, based on one or several **organisations**, ideally predefined in a catalogue. Essential services like communication and dynamic discovery must also be addressed here.

7. identify **conflict zones** (at each layer) based on the intersection of influence scopes (S_{Ii}) serving incompatible goals.

8. modify or extend the initial organisations with **conflict-resolution** facilities. One option (developed in the experiments below) is to add goal-management control layers to CEs located in the conflict zones; these CEs become 'agents'. Another option is to introduce special-purpose CEs implementing more complicated conflict-resolution design patterns (as shown in [10]).

9. enhance CEs (as needed) with self-adapting capabilities, by adding **adaptation-control layers** to them.

10. further **refine** any of the existing layers in any of the CEs (as needed) to enable more complex behaviours - e.g. add an adaptation layer to a CE's goal-management layer to change its policies during runtime; or, further split an adaptation layer into abstract layers to deal with its complexity.

The first four steps allow defining the controller's overall perimeter - its external inputs (goals and monitoring information) and outputs (actions and state reporting). This paper only focuses on goals and actions; future work will include evaluation and reporting. The fourth step starts defining the controller's internal design, based on successive layers and CEs. Steps five to seven set in place the integration infrastructure including conflict-resolution support. The last two steps refine the design with self-adaptation capabilities at all layers. If needed, they can further complexify each layer by splitting or enhancing them with otrthogonal layers. The smart micro-grid application exemplifies this procedure next.

5 Illustration via a smart micro-grid

5.1 Design and implementation

Like most SE contributions, evaluating the generic architecture proposed cannot rely on formal proofs and would require too vast experimental resources to rely on a meaningful empirical approach - e.g. [16]. Hence, for now, it can only be validated through rigorous argumentation and relevant exemplification, which is the aim of this section. We show how a smart micro-grid prototype was designed following the methology depicted above. Please note that many of the conceptual layers identified are superposed in the concrete controller design.

We first identify the **authorities** involved and the types of **goals** they could specify (step 1). The authorities are *electricity providers* and *home owners*. Electricity providers define *power goals* over their district grids - $G_{power}([P_{d\_low}, P_{d\_high}], district_{id})$. Home owners define *mode goals* within their houses, to prioritise either "comfort" or "saving" modes - e.g., $G_{mode}(''comfort'', house_{id})$ and $G_{mode}(''saving'', house_{id})$. Mode goals can also be set on individual devices directly - e.g. $G_{mode}(''saving'', device_{id})$. Home owners can also specify power goals; for simplicity we only allow this for devices - $G_{power}([P_{h\_low}, P_{h\_high}], device_{id})$. Figure 4 shows how these goals are defined (in simplified notation) for one district and two houses.

Let us now identify the S_A and S_I for each one of these goals (steps 2-3). For simplicty we ignore context scopes. The district's power goal - $G_{power\_district}$ in Figure 4 - has an S_A that comprises all electric devices in the district. Yet, for legal reasons, it can only act directly at the house level; then each house acts on its own devices (as discussed later). For a house's mode goal - e.g. $G_{comfort\_house1}$ - the S_A covers all devices in the house. Each device's goal (power or mode) has an S_A that includes that device. Finally, all influence scopes S_I include all electric devices since these are all connected to the same district grid. In addition, the S_Is of mode goals include the atmosphere of targeted rooms and of neighbouring rooms.

For each goal, we now define the **abstract layers** and the corresponding goal-translation process (step 5). The controller pursuing the district power goal $G_{power\_district}$ has three abstract layers: district, house and device. Each layer is composed of one or several CEs: $CE_{district}$ in the district layer; CE_{house1} and CE_{house2} in the house layer; and $CE_{heater1}$, $CE_{heater2}$ and CE_{lamp} in the device layer [Fig. 4]. The CE in the district layer translates the district's power goal (declarative) into individual commands (procedural) for CEs in the house layer; and then into commands for CEs in the device layer. Commands are further discussed when defining the exact integration organisations. Device controllers pursuing mode goals comprise two abstract layers. Hence, each CE in the device layer is further split into these two abstract layers [Fig. 6]. For a heater, these translate mode goals into temperature intervals (predefined) and then into concrete power values (via a PID controller). For a lamp, mode goals are translated to light intensity then to power val-

ues.

Considering the architectural layout so far we decide to adopt a **hierarchical organisation** for **integrating CEs** within district, house and device layers (step 6). For the power goal, this organisation includes two roles: *power managers* that pursue power goals by orchestrating *prosumers*. In [Fig. 5] $CE_{district}$ plays a power manager role; device CEs prosumer roles; and CE_{house1} both roles - prosumer (for $CE_{district}$) and manager (for $CE_{heater1}$, $CE_{heater2}$ and CE_{lamp}).

We now define the power-related *commands* that are exchanged among roles in the organisation. When a manager detects that measures approach the power interval's high limit it sends a *reduce_load* order to its prosumers; for the lower limit it sends a *rise_load* order; when well in-between the limits it sends an *any_load* order to cancel the previous ones. To avoid oscillations, these orders are sent progressively, in random order, and the effects observed before new orders are sent. For mode goals, "comfort" or "saving" goals are simply transmitted from house CEs to device CEs.

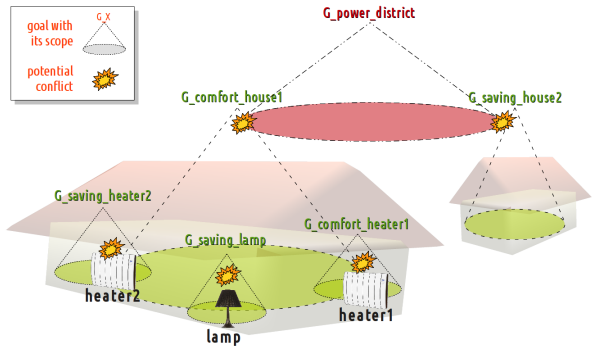

Figure 4: Goals and scopes in the smart micro-grid.

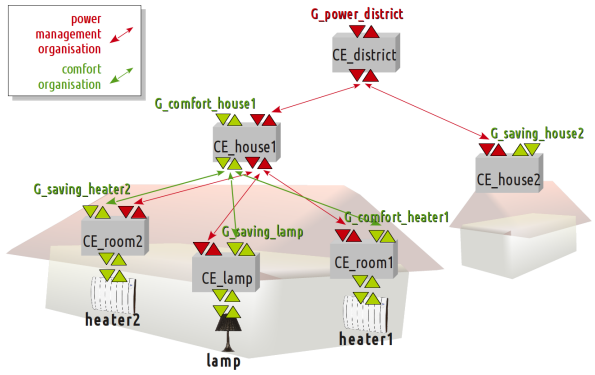

Figure 5: Integration organisations in the micro-grid.

To obtain a controller that pursues all presented goals simultaneously, the corresponding organisations are superposed onto one hierarchy. Here, CEs at each level combine the architectural layers of all previous organisations. Let's see some of the **conflicts** that can occur (step 7). One conflict is caused by district power goals and house mode goals intersecting over the house scope - the **conflict zone** includes all CEs and resources in the home, as they belong to the S_A of both goals. Since S_Is of all houses also intersect, house power goals are also conflicting. Yet, this conflict is equivalent to the previous one since house consumptions will be reflected in the district's power evaluation. Another conflict can occur between mode goals set for the entire house and directly on each device; here, the conflict zone includes the concerned device.

Both conflicts are **resolved** by adding **goal management layers** to CEs in the conflict zones [Fig. 6] (step 8). This control layer receives conflicting goals as inputs and provides a coherent goal as output for the control layer below - e.g. a power interval for power managers in house CEs; and, a temperature interval for PIDs in heater CEs. Goal managers give priority to mode goals. If in "comfort" mode, it ignores orders from power managers above; if "saving", it modifies the interval for the manager below depending on the orders received ([Fig. 7] and [Fig. 8]). In addition, goal managers of devices prioritise goals that are set on the device directly over those that are derived from the house mode goal.

Finally, the prototype does not include self-adaptation functions or further refinements (steps 9 and 10). From a design perspective, an adaptation-control layer could be added for instance to power management CEs, so as to discover and integrate new prosumers. Also, an adaptation layer could be added to the heater's PID abstract layer for reconfiguration purposes.

5.2 Scenarios, results and discussion

The scenarios depict the smart micro-grid when the outside temperature is dropping, heaters increase consumption thus rising the district load [11]. They focus on the behaviour of two district houses - h1 and h2. H1 is set to a "comfort" mode - most heaters ignore load-related orders and sustain a 23°C temperature; only a few that were directly set in "saving" mode respond. Hence, h1's power target is crossed [Fig. 8-a]. This conforms to the user's "comfort" goal and will impact the bill. The district manager detects a consumption increase and starts sending reduce_load orders to house prosumers. Since in "comfort" mode, h1's CE ignores them. H2 is initially set to "saving" and reacts by lowering its power interval [Fig. 8-b]; it then sends reduce_load orders to device prosumers to fit the new range. Heaters react by lowering their temperatures to 20°C which therefore lowers their consumptions and helps the district manager. To illustrate a **dynamic goal change**, let's assume that h2's

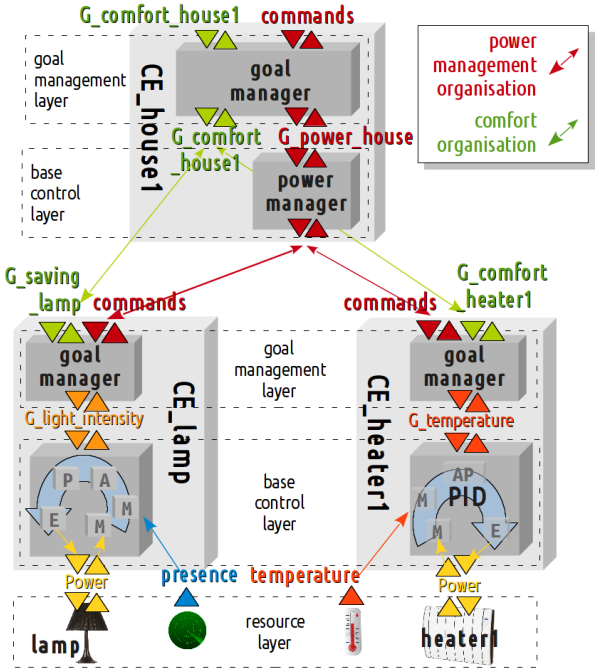

Figure 6: Design detail for the house control.

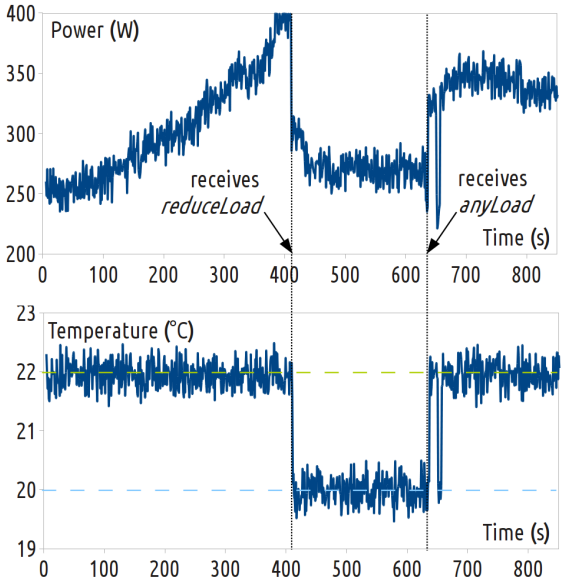

Figure 7: Management of a flexible heater.

owners switch its mode to "comfort", to accommodate an unexpected guest. H2 **self-adapts** - its prosumers ignore orders and consume more [Fig. 8-b].

The scenarios were run on a smart grid simulator that models physical entities, like houses, rooms, grid, heaters, lamps or solar panels; with related behaviours and attributes such as heat transfer, temperature and prosumption (in simulation time [s]). It is based on a service-oriented component technology - iPOJO/OSGi - and Akka middleware. These enable devices and CEs to be deployed, reconfigured and removed at run-time. A miniature house model was also built to ensure realistic behaviour. For limited space reasons, the presented results (based on the simulation) were selected for illustrative purposes; a web version of the simulation is available online for further explorations[2].

6 Related Work

This paper's contribution intersects several interrelated works, from various domains, from which we can only cite a few here. Separating goals from the means of achieving them has been proposed in autonomic computing [17], system engineering [21] and software agents [26]. In [17], management objectives can be defined as procedural policies, declarative goals or utility functions. In [21], "posed problems" are separated from the "resources" that can solve them, hence delaying resource selection until runtime. In the AI domain, "intelligent"

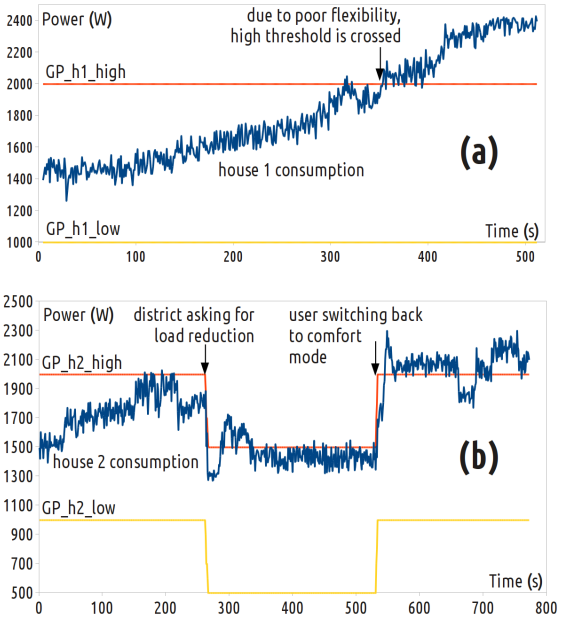

Figure 8: Power management in h1 (a) and h2 (b).

agents modify the environment to achieve declarative goals [26]. They can adapt internal strategies when faced with unpredictable situations. In cybernetics, Ashby proposed an "ultra-stable" architecture that relies on two superposed control loops for adapting the system and its control strategy [2].

The generic architecture presented is consistent with

these approaches - it generalises goal definitions to include scope and time (also identified in [18]), separates goals from control logic and introduces meta-control layers to self-adapt this logic and negotiate goals. Similar layered architectures have been proposed in various domains, including Brook's subsumption architecture in robotics [6] or Kramer and Magee's architecture [19] for autonomic systems. We drew inspiration from these proposals and identified the different natures of concerns that lead to system layering. The model thus proposes abstract, control and integration layers, which address orthogonal concerns and can be combined in recursive ways. The organisation paradigm is common in the Multi-Agent Systems domain [14][29] and was adopted in the model to enable internal adaptation and integration of plug and play resources while conserving important invariants. We are exploring this idea further in a parallel project [9]. In [10] we have presented an initial catalogue of organisations focused on conflict resolution. Splitting controllers into CEs of various types - such as control tasks and feedback loops - relies on previous projects [23]. The *feedback loop* appears as a first-class entity in all autonomic systems [7][15].

The generic architecture presented is complementary with many contributions that address particular issues of autonomic computing and IoT. These include numerous application-specific solutions that propose ad-hoc ways of constructing or integrating control-loops - e.g. monolithic control in DigiHome [25]; hierarchical managers in fANFARE [22], AutoHome [5], or using a single coordination manager [12]; or agent-oriented managers [16]. These fit the generic architecture, representing particular instantiations. Another category of complementary contributions focus on specific communication protocols and integration middleware for heterogeneous plug-and-play devices, like DigiHome [25] or RoSe [22] home automation platforms. Finally, [8] presents a generic integration model focused on categorising control loops based on their reciprocal interference (via shared knowledge) and proposing coordination and synchronisation protocols to integrate them.

Self-management requirements for the smart micro-grid have been identified in several works [3][4][13][27]. Notably, [3] proposes a distributed load management algorithm, where "colour" statuses solve management conflicts between load balancers and appliances. These fit the arhitectural model and can be adopted to implement corresponding CE layers. The smart grid domain was targeted here as a rich use case for illustrating the architecture and highlighting its main contributions.

7 Conclusions and Future Work

This paper proposed a generic architecture and methodology to help analyse and design multi-goal, multi-scale, adaptive autonomic control systems operating in distributed open environments, such as the IoT. It relies on the assumption that autonomic systems of this kind will be built by integrating control elements (CEs) of diverse types. Taking a SE-oriented approach, it aims to identify the reusable artefacts that can help instantiate this type of solution.

The contribution includes a conceptual model; a generic architecture adopting these concepts; and a development methology indicating how control applications can be designed guided by these concepts and architecture. The conceptual model considers *goals* as key elements that should be separated from the control logic necessary to pursue them. It provides a goal definition that can be translated, split and propagated across various CE types in the system, down to concrete actions on resources. The main difficulty factors are identified - including conceptual abstraction gaps, logistical complexity, adaptability and scalability issues - and suitable SE techniques identified for addressing them - including orthogonal types of layering and flexible modular architectures. The conceptual model also identifies *conflicts* as stand-alone elements that must be clearly defined, identified and addressed. The generic architecture relies on this conceptual base to define more concrete artefacts for system design. It includes several types of CEs - control tasks, integration tasks and control composites - featuring different behaviours and hence requiring different facilities - base, adaptive and agent-like functions. To integrate artefacts into flexible systems while ensuring core properties the model adopts an organisation-oriented approach inspired from the multi-agents. It indicates how this can be extended with reusable artefacts specific to conflict-resolution to handle multi-goal scenarios.

To illustrate its applicability and benefits the paper showed how a sample smart micro-grid was designed and implemented based on the generic architecture and methodology. Several runtime scenarios were selected to show how to define goals in business-specific terms, translate and split them among several abstraction levels, deal with multiple authorities and heterogeneous resources, handle multi-goal conflicts, adapt to dynamic context changes and goal reconfigurations, and integrate new resources. The paper did not address issues related to security concerns and the possible incompatibility of integrated CEs. System robustness and scalability were considered in the general architecture model but not yet tested or shown here.

Future work will concentrate on analysing autonomic systems in other domains to further test the architecture's

applicability and extend it if necessary. This will include time-related concerns, which are critical to decision-making, decentralised coordination and system stability. The authors intent is to bring forward the understanding of autonomic systems operating in the IoT context and the associated support for developing them.

References

[1] ALLERDING, F., BECKER, B., AND SCHMECK, H. Decentralised energy management for smart homes. In *Organic Computing A Paradigm Shift for Complex Systems*, C. Muller-Schloer, H. Schmeck, and T. Ungerer, Eds., vol. 1 of *Autonomic Systems*. Springer Basel, 2011, pp. 605–607.

[2] ASHBY, W. R. *Design for a brain.* New York :Wiley, 1954.

[3] BEAL, J., BERLINER, J., AND HUNTER, K. Fast precise distributed control for energy demand management. In *Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2012, Lyon, France* (2012).

[4] BECKER, B., ALLERDING, F., REINER, U., KAHL, M., RICHTER, U., PATHMAPERUMA, D., SCHMECK, H., AND LEIBFRIED, T. Decentralized energy-management to control smart-home architectures. In *ARCS* (2010), vol. 5974 of *Lecture Notes in Computer Science*, pp. 150–161.

[5] BOURCIER, J., DIACONESCU, A., LALANDA, P., AND MCCANN, J. A. Autohome: An autonomic management framework for pervasive home applications. *TAAS 6*, 1 (2011), 8.

[6] BROOKS, R. A. *Cambrian Intelligence: The Early History of the New AI*. MIT Press, 1999.

[7] BRUN, Y., MARZO SERUGENDO, G., GACEK, C., GIESE, H., KIENLE, H., LITOIU, M., MÜLLER, H., PEZZÈ, M., AND SHAW, M. Software engineering for self-adaptive systems. Springer-Verlag, Berlin, Heidelberg, 2009, ch. Engineering Self-Adaptive Systems through Feedback Loops, pp. 48–70.

[8] DE OLIVEIRA JR., F. A., SHARROCK, R., AND LEDOUX, T. Synchronization of multiple autonomic control loops: Application to cloud computing. In *COORDINATION* (2012), M. Sirjani, Ed., vol. 7274 of *Lecture Notes in Computer Science*, Springer, pp. 29–43.

[9] DEBBABI, B., DIACONESCU, A., AND LALANDA, P. Controlling self-organising software applications with archetypes. In *Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference on* (2012), pp. 69–78.

[10] FREY, S., DIACONESCU, A., AND DEMEURE, I. Architectural integration patterns for autonomic management systems. *9th IEEE International Conference and Workshops on the Engineering of Autonomic and Autonomous Systems (EASe)* (2012).

[11] FREY, S., HUGUET, F., MIVIELLE, C., MENGA, D., DIACONESCU, A., AND DEMEURE, I. Scenarios for an autonomic micro smart grid. *1st International Conference on Smart Grids and Green IT Systems (SmartGreens 2012)* (2012).

[12] GUEYE, S. M.-K., RUTTEN, É., AND TCHANA, A. Discrete control for the coordination of administration loops. In *IEEE Fifth International Conference on Utility and Cloud Computing, UCC 2012, Chicago, IL, USA* (2012), pp. 353–358.

[13] HERMANNS, H., AND WIECHMANN, H. Demand-response management for dependable power grids. In *Embedded Systems for Smart Appliances and Energy Management*, C. Grimm, P. Neumann, and S. Mahlknecht, Eds., vol. 3 of *Embedded Systems*. Springer New York, 2013, pp. 1–22.

[14] HORLING, B., AND LESSER, V. A survey of multi-agent organizational paradigms. *Knowl. Eng. Rev. 19* (December 2004), 281–316.

[15] IBM. An Architectural Blueprint for Autonomic Computing. 2006.

[16] JENNINGS, N. R., AND BUSSMANN, S. Agent-based control systems. *IEEE Control Systems Magazine 23* (2003), 61–74.

[17] KEPHART, J. O., AND WALSH, W. E. An artificial intelligence perspective on autonomic computing policies. In *POLICY* (2004), IEEE Computer Society, pp. 3–12.

[18] KERNBACH, S., SCHMICKL, T., AND TIMMIS, J. Collective Adaptive Systems: Challenges Beyond Evolvability, 2009.

[19] KRAMER, J., AND MAGEE, J. Self-managed systems: an architectural challenge. In *2007 Future of Software Engineering* (Washington, DC, USA, 2007), FOSE '07, IEEE Computer Society, pp. 259–268.

[20] LALANDA, P., MCCANN, J., AND DIACONESCU, A. *Autonomic Computing: Principles, Design and Implementation*. Undergraduate Topics in Computer Science Series. Springer London, Limited, 2013.

[21] LANDAUER, C. Problem posing as a system engineering paradigm. In *ICSEng* (2011), H. Selvaraj and D. Zydek, Eds., IEEE, pp. 346–351.

[22] MAUREL, Y., CHOLLET, S., LESTIDEAU, V., BARDIN, J., LALANDA, P., AND BOTTARO, A. fanfare: Autonomic framework for service-based pervasive environment. In *IEEE SCC* (2012), L. E. Moser, M. Parashar, and P. C. K. Hung, Eds., IEEE, pp. 65–72.

[23] MAUREL, Y., LALANDA, P., AND DIACONESCU, A. Towards a service-oriented component model for autonomic management. In *IEEE SCC* (2011), H.-A. Jacobsen, Y. Wang, and P. Hung, Eds., IEEE, pp. 544–551.

[24] OGATA, K. *Modern Control Engineering*, 2nd ed. Prentice Hall PTR, 1990.

[25] ROMERO, D., HERMOSILLO, G., TAHERKORDI, A., NZEKWA, R., ROUVOY, R., AND ELIASSEN, F. The digihome service-oriented platform. *Software: Practice and Experience* (2011).

[26] RUSSELL, S. J., AND NORVIG, P. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[27] SCHMECK, H., AND KARG, L. E-energy - paving the way for an internet of energy (auf dem weg zum internet der energie). *it - Information Technology 52*, 2 (2010), 55–57.

[28] UCKELMANN, D., ISENBERG, M.-A., TEUCKE, M., HALFAR, H., AND SCHOLZ-REITER, B. Autonomous control and the internet of things: Increasing robustness, scalability and agility in logistic networks. In *Unique Radio Innovation for the 21st Century*, D. C. Ranasinghe, Q. Z. Sheng, and S. Zeadally, Eds. Springer Berlin Heidelberg, 2011, pp. 163–181.

[29] WEYNS, D., HAESEVOETS, R., HELLEBOOGH, A., HOLVOET, T., AND JOOSEN, W. The macodo middleware for context-driven dynamic agent organizations. *TAAS 5*, 1 (2010).

Notes

[1] Software agent may be of these types [26], here we only use goal-oriented agents that can manage goals.

[2] try the simulation online at http://perso.telecom-paristech.fr/ sfrey/