Towards a reference model for multi-goal, highly-distributed and dynamic autonomic systems

[1]Sylvain Frey, [2]Ada Diaconescu, [1]David Menga and [2]Isabelle Demeure
[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 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 micro smart 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 reference model for multi-goal, adaptable and open autonomic systems, exemplified via the development of a concrete autonomic application for the micro smart 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.

Keywords: autonomic control, software engineering, model, multi-goal autonomic systems, control loop integration, open adaptable and distributed applications, agent organisations, smart home, micro smart grid.

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 micro smart grid. To cover both the Autonomic Computing (AC) and IoT domains involved in this example, the paper employs the generic term *autonomic control* [1] 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, available 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 and systems [2] [3][4][5][6]. The long-term aim is to build a comprehensive **reference model for autonomic systems.**

The proposed reference model 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 entities* 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 [5][6][7]. 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 [3][4][5][6][7][8].

Another important challenge lies in coordinating control elements so as to obtain coherent controllers that can pursue several goals, adapt dynamically 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 [4] or control strategies [5]. The reference model presented here focuses 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 reference model. Indeed, most of these can be found in related fields such as automatic control [9], collective adaptive systems [10], multi-agents [8][11], robotics [12], cybernetics [13] or autonomic systems [7][14] [15]. 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 employed for designing autonomic control systems, and assimilating them into a coherent reference model. 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 a problem of mapping declarative actions (goals) into concrete actions (on managed resources), in a context-aware and extensible way; bringing together 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 autonomic micro smart grid.

The ongoing aim is to help answer questions on: How to develop scalable and adaptable feedback loops? How to integrate multiple feedback loops so as to pursue 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. The proposed contribution is relevant to both autonomic systems in general – as it helps design multi-goal, highly-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 micro smart grid application, highlighting its requirements and design challenges. Sections 3 and 4 introduce the conceptual and architectural aspects of the proposed model, respectively, illustrating them via concrete design examples from the micro smart grid. Section 5 relies on these examples to illustrate the complete design of the application. Section 6 discusses related work and section 7 concludes the paper and indicates future research.

2. Smart Houses meet Micro Smart 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 'micro smart grid' implicitly including 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 be related to cost – like keeping the electricity bill under a threshold, or minimise consumption to reflect an ecological attitude. 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 partially ensuring comfort 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 directly handle appliances (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 electrical 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 [7][13][16]. A system's state is defined via a set of variables whose values can predict its behaviour in the near future [9] (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 automatically evaluated by calculating 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 and considers that procedural goal definitions 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), so as 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 – the *action resources* – is not necessarily equal to the set of resources in the goal's scope – the *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 (e.g. for a temperature goal in one room, the controller may act upon heaters within that room as well as within the neighbouring rooms). A controller's action resources for pursuing a goal constitute its *Action Scope (S_A)*. The set of resources whose state they can 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.

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 clearly 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 eventually translated into *concrete actions* (A), executed via resource effectors (imperative) [7].

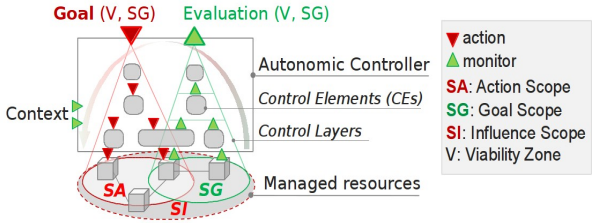

Figure 1: Goal projection and evaluation.

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 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.

'Classical' 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 (first 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. Goal evaluation follows the inverse path – monitored data from S_G resources is translated into administrative domain concepts. Second, *control layers* can add meta-control abilities to base-level functions, thus enabling controllers to self-adapt (third factor) – e.g. when a smart house's goal changes from 'comfort' to 'saving' mode a meta-control layer adapts the base control behaviour. Third, *orchestration layers* can be added on top of otherwise decentralised control elements (discussed below) to form an integration control hierarchy. As orchestration and abstract layers are often superposed 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 (second and third 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

(fourth 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 generating a *control flow* through the system [Fig. 2]. It can pass through CEs within a layer, like the MAPE elements of a control loop; as well as between layers, like a base control loop triggering a meta-control loop or an orchestrator. 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 one case, 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, one 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 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, conflicts 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 (exemplified above). Control strategies are conflicting when they carry contradictory actions through overlapping influence scopes (S_I). 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 [4] 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].

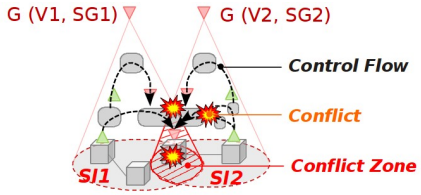

Figure 2: Conflicts and resolution.

4. Architectural Model

4.1 Types of layers and control elements

To remain generic, the reference model proposed here consists in a logical architecture, which relies on and refines the Autonomic Computing Blueprint [7]. It promotes an approach based on the *integration* of *control elements* (CEs) of various types. This can be performed statically or dynamically, to develop and then adapt the system. Hence, an autonomic system consists of managed resources, which can be acted upon and monitored; and an autonomic controller, which receives and pursues goals.

An autonomic controller can be designed based on the abstract and concrete elements in the conceptual model. It can be recursively split into various combinations of abstract, orchestration and control layers, each one implemented via concrete CEs that pursue partial goals over complementary scopes. Concretely, CEs can represent: i) *control tasks* – providing control-related functions (e.g. monitoring, decision, execution, knowledge management, other atomic functions or combinations of these [5]; ii) *integration tasks* – providing integration-specific functions (e.g. conflict resolution [4]); and iii) *control composites* – consisting of flexible compositions of control tasks and (optionally) integration tasks, for providing more advanced control structures and

functions, such as single or integrated feedback loops. Control composites can or not be *encapsulated*. When encapsulated, they allow building fractal-like structures, viewed from the outside as a single well-integrated CE [Fig. 3] – hence identical to a control task.

From a behavioural perspective, CEs may be: *reactive* to external stimuli; *reactive with additional state*, or knowledge; *self-adaptive* to changes; or *agent*[1]*-like* managing and negotiating goals given by other entities. To achieve such incremental capacities, the model defines three types of control layers, where any CE may include one or several of these layers. The *base control layer* monitors and acts on managed resources following a pre-selected strategy; it enables reactive behaviour. The *meta-control* or *adaptation layer* ensures the base layer's adaptation to change, by altering or fine-tuning its strategy [13][14]; 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. The decision may be binary or more nuanced, based on the requester's authority, on already accepted goals and their conflicts with the new goals. Finally, abstract layers for goal translation will be application-specific.

4.2 Requirements for integration

Integrating CEs must rely on standardised interfaces and protocols. While the details of these are domain and application-specific, their general semantics and purpose can be identified. This view is compliant with the Autonomic Computing Blueprint [7], 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, which can also represent lower-level CEs. They also *provide* monitoring and action interfaces for allowing administrators and higher-level CEs to access them. These interfaces are the main enablers for CE layering and orchestration. Their semantics will differ depending on the CE type and conceptual layer – e.g., they will represent concrete touchpoints for monitoring and execution control tasks in a base control layer; and, goal specification and evaluation touchpoints for control loops in an adaptation layer. Interface implementations will also differ – 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.

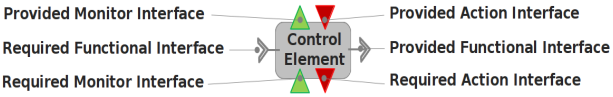

Figure 3: Control element interfaces.

4.3 Integration and adaptation

Integrating CEs into multi-goal, distributed and adaptable autonomic controllers requires handling problems of communication, coordination and control. The architectural model 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 no further treated here. Conversely, coordination and control are key concerns, which are especially challenging when CEs and managed resources must be integrated dynamically. Two complementary techniques can be adopted here. The first one relies on facilities for runtime evaluation, reporting and autonomic adaptation. This aspect was included in the conceptual model and will be further developed in future work. The second one relies on imposing architectural templates or *organisations* – a term borrowed from the agent community (e.g. [8][18][19]). 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 [3].

An organisation consists of several *roles* that interact in a well-defined way. A role is defined via well-specified set of capabilities (e.g. a 'prosumer' role in a smart grid organisation). The role can be assigned to any resource that provides these capabilities (e.g. a heater takes the prosumer role), statically or during runtime. Autonomic controllers are designed based on one or several composed organisations, which are in turn defined based on artefacts in the reference model. The exemplified micro smart grid shows how predesigned organisations facilitate system development and enable adaptation. A catalogue of reusable organisations can be progressively developed for autonomic systems. A core set was presented in [4] as integration design patterns. These include centralised orchestrators, decentralised coordin-

[1] Software agent may be of these types [18], here we only use goal-oriented agents that can manage goals.

ation via functions embedded in control elements, hierarchical multi-layer organisations, and aggregator and filter interceptors for integrated control flows.

5. Illustration via the micro smart grid

5.1 Application design and implementation

Like most SE contributions, evaluating the presented reference model cannot rely on formal proofs and would require too vast experimental resources to rely on a meaningful empirical approach – e.g. [11]. Hence, for now, it can only be validated through rigorous argumentation and relevant exemplification, which is the aim of this section. [Fig. 4] depicts the overall design of the exemplified micro smart grid. In short, we first identified the authorities involved and the types of goals they could specify. For each goal, we spotted the difficulty factors and addressed them using model artefacts. This resulted in well-defined organisations, whose roles were then assigned to CEs with pre-coded behaviours; these could also be discovered and plugged-in at runtime. To address all goals at once, initial organisations were combined and extended with conflict-resolution mechanisms placed in the conflict zones. The overall design takes the shape of a hierarchy, where abstract, control and orchestration layers overlap at each node (CE). CEs pursue different goals, sometimes conflicting, and operate at different scales. Let us look at some details.

An electricity provider defines a power goal over the district grid – ([GP_d_low, GP_d_high], district). The district controller (CE_d) has a large S_A including all appliances – here, heaters and lamps – in all district houses, with different owners. Hence the district power goal is **split** amongst goals with progressively smaller scopes; each one is assigned to a new CE – e.g. CE_h1 and CE_h2 for house scopes; CE_r1, CE_r2, CE_l for device scopes. CEs with larger scopes **orchestrate** those with smaller scopes, resulting in a hierarchical architecture. Next, **abstraction layers** are superposed over the CE hierarchy to translate district power goals (declarative) for the CE_d into orders (procedural goals) for house and device CEs. This leads to an **organisation** that includes two roles: *load managers* that pursue power goals by orchestrating *prosumers*. In [Fig. 4] CE_d plays a load manager role, device CEs prosumer roles and CE_h both roles – prosumer (for CE_d) and manager (for device CEs). 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. Finally, an adaptation **control layer** can be added to manager CEs to discover and integrate new prosumers (not shown).

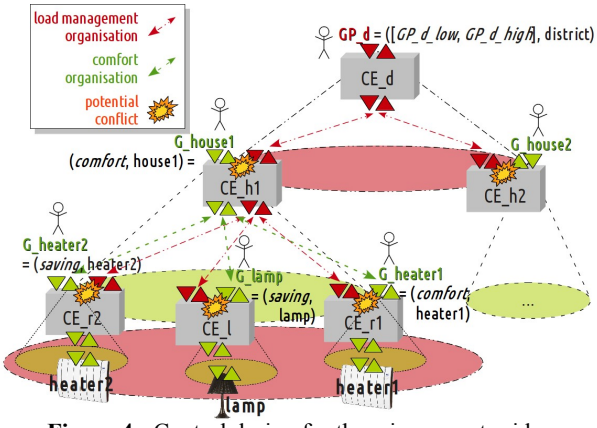

Figure 4: Control design for the micro smart grid.

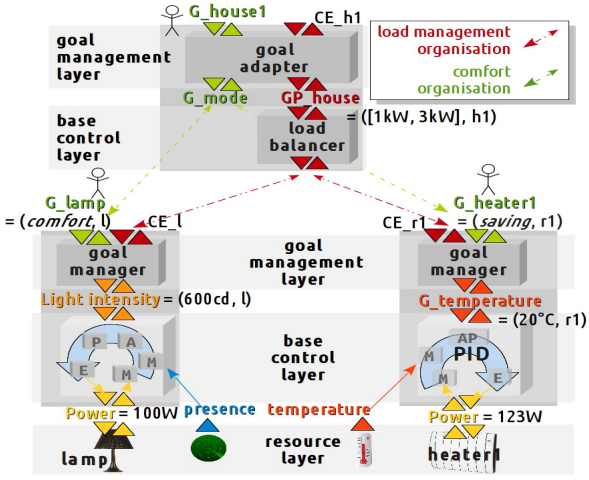

Figure 5: Design detail for the house control.

Home owners define goals for their house. These can prioritise either a 'comfort' or 'saving' mode. As before, this leads to a hierarchical **organisation** that **splits** the house goal (e.g. G('comfort', h1) – declarative) among device goals (e.g. G('comfort', heater1) – declarative). For device CEs, mode goals must be translated into concrete actions on devices. Hence, these CEs are split into several **abstract layers** – e.g. for the CE_r1 heater in [Fig. 5] modes are translated to temperatures (predefined) and then to power settings (via a PID controller); an adaptation **control layer** could also be added to the PID abstract layer for reconfiguration purposes (not shown). Finally, home owners can also set goals directly on devices, at different abstraction layers – e.g. 'saving' mode or temperature on heater1.

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. 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_1 of both goals. Another one can occur between mode goals set for the entire house and directly on each device – the conflict zone includes the concerned device. Both conflicts are handled by adding **goal management layers** to CEs in the conflict zones [Fig. 5]. This layer receives conflicting goals as inputs and provides a coherent goal as output for the control layer below – e.g. power for load managers in house CEs and temperature for PIDs in heater CEs. It also represents an **abstraction layer** as it translates declarative and procedural goals (mode and orders) into a declarative goal (power or temperature interval). A goal manager gives priority to mode goals – if 'comfort', it ignores orders from load managers above; if 'saving', it modifies the interval for the manager below depending on the orders received ([Fig. 6] and [Fig. 7]). In addition, device goal managers prioritise goals set directly on the device over those derived from the house mode.

5.2 Scenarios, results and discussion

The scenarios depict the micro smart grid when the outside temperature is dropping, heaters increase consumption thus rising the district load [26]. 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. 7-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. 7-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 owners switch its mode to 'comfort', to accommodate an unexpected guest. H2 **self-adapts** – its prosumers ignore orders and consume more [Fig. 7-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].

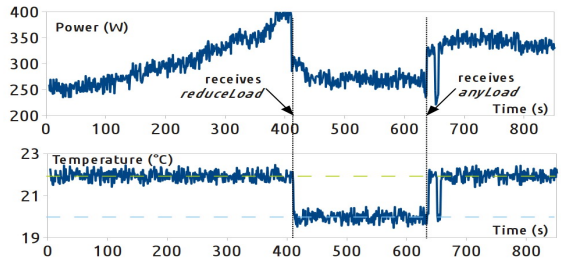

Figure 6: Management of a flexible heater.

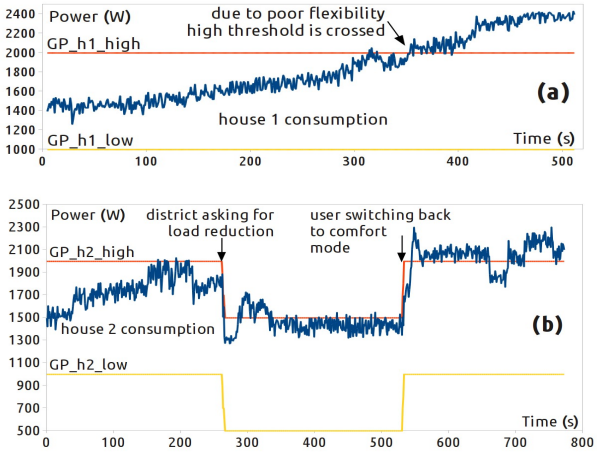

Figure 7: Power management in h1 (a) and h2 (b).

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 [20] and software agents [18]. In [17], management objectives can be defined as procedural policies, declarative goals or utility functions. In [20], 'posed problems' are separated from the 'resources' that can solve them, hence delaying resource selection until runtime. In the AI domain, 'intelligent' agents modify the environment to achieve declarative goals [18]. They can adapt internal strategies when faced with unpredictable situations. In cybernetics, Ashby proposed an 'ultra-stable' architec-

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

ture that relies on two superposed control loops for adapting the system and its control strategy [13].

The presented reference model is consistent with these approaches – it generalises goal definitions to include scope and time (also identified in [10]), 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 [12] or Kramer and Magee's architecture [14] 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 orchestration layers, which address orthogonal concerns and can be combined in recursive ways. The organisation paradigm is common in the Multi-Agent Systems domain [8][19] 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 [3]. In [4] 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 [5][6]. The *feedback loop* appears as a first-class entity in all autonomic systems [15][7].

The presented reference model 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 [21]; hierarchical managers in fANFARE [22], AutoHome [6], or using a single coordination manager [23]; or agent-oriented managers [11]. These fit the reference model representing particular instantiations. Another category of complementary contributions focus on specific communication protocols and integration middleware for heterogeneous plug and play devices, like DigiHome [21] or RoSe [22] home automation platforms. Finally, [24] 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 micro smart grid have been identified in several works [25][27][28][29]. Notably, [25] proposes a distributed load management algorithm, where "colour" statuses solve management conflicts between load balancers and appliances. These fit the reference 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 model and highlighting its main contributions.

7. Conclusions and Future Work

This paper proposed a reference model 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 model comprises two complementary parts – conceptual and architectural. 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 architectural model 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 micro smart grid was designed and implemented based on the reference model. 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 model but not yet tested or shown here.

Future work will concentrate on analysing autonomic systems in other domains to further test the model'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] D. Uckelmann et al., "Autonomous Control and the Internet of Things: Increasing Robustness, Scalability and Agility in Logistic Networks", in Unique Radio Innovation for the 21st Century, Springer-Verlag, Berlin Heidelberg, 2010.

[2] P. Lalanda, J.A. McCann, A. Diaconescu, "Autonomic Computing – Principles, Design and Implementation", Springer, due: May 2013.

[3] B. Debbabi, A. Diaconescu, P. Lalanda "Controlling self-organising software applications with archetypes", IEEE SASO, Lyon, France, 2012.

[4] S. Frey, A. Diaconescu, I. Demeure, "Architectural Integration Patterns for Autonomic Management Systems", IEEE EASe, Novi Sad, Serbia, 2012.

[5] Y. Maurel, P. Lalanda, A. Diaconescu, "Towards a service-oriented component model for autonomic management", IEEE SCC, Washington DC, USA, 2011.

[6] J. Bourcier, A. Diaconescu, P. Lalanda, J. McCann, "AutoHome: an Autonomic Management Framework for Pervasive Home Applications", ACM TAAS, Vol. 6, Iss. 1, Feb. 2011.

[7] "An architectural blueprint for autonomic computing", IBM Whitepaper, June 2005, 3rd edition.

[8] B. Horling and V. Lesser, "A survey of multi-agent organizational paradigms", Knowledge Engineering Review, Vol. 19, No. 4, 2004, pp 281-316

[9] K. Ogata, "Modern Control Engineering", Prentice Hall, 2nd edition, 1990.

[10] "Collective Adaptive Systems", Expert Consultation Workshop Report, FET Proactive, Nov. 2009

[11] N. Jennings and S. Bussmann, "Agent-Based Control Systems – Why are They Suited to Engineering Complex Systems?", IEEE Control Systems Magasine, June 2003, pp 61-73.

[12] R. A. Brooks, "Cambrian Intelligence: The Early History of the New AI", A Bradford Book, 1st edition, July 1999, ISBN-13: 978-0262522632

[13] W. R. Ashby, "Design for a Brain: The Origin of Adaptive Behaviour", Chapman and Hall, Ltd. London, 2nd edition, 1960. (1st edition in 1952)

[14] J. Kramer, J. Magee, "Self-Managed Systems: an Architectural Challenge", Future of Software Engineering, Washington DC, USA, 2007.

[15] Y. Brun et al., "Engineering Self-Adaptive Systems through Feedback Loops", in "Self-Adaptive Systems", Springer, Berlin, 2009, pp. 48–70

[16] C. Müller-Schloer, H. Schmeck, T. Ungerer (Edt.), "Organic Computing – A Paradigm Shift for Complex Systems", Springer Basel AG, 2011

[17] J. O. Kephart, W. E. Walsh, "An Artificial Intelligence Perspective on Autonomic Computing Policies", IEEE POLICY 2004, pp. 3-12

[18] S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach", Prentice Hall, 3rd ed., 2009

[19] D. Weyns, R. Haesevoets, A. Helleboogh, T. Holvoet, W. Joosen, "The MACODO Middleware for Context-Driven Dynamic Agent Organizations", ACM TAAS, 5(1):3.1–3.29, 2010

[20] C. Landauer, "Problem Posing as a System Engineering Paradigm", IEEE ICSENG, Washington, DC, USA, 2011, pp. 346-351.

[21] D. Romero et al., "The DigiHome Service-Oriented Platform", Softw. Pract. Exper, 2011; p1–27

[22] Y. Maurel et al., "fANFARE: Autonomic Framework for Service-based Pervasive Environment", IEEE SCC 2012, Honolulu, USA, pp. 65-72

[23] S.M.-K Gueye, E. Rutten, A Tchana, "Discrete Control for the Coordination of Administration Loops", IEEE Intl. Conf. UCC 2012, pp.353-358

[24] F. A. de Oliveira, R. Sharrock, T. Ledoux, "Synchronization of Multiple Autonomic Control Loops: Application to Cloud Computing", CO-ORDINATION 2012, Stockholm, Sweden.

[25] J. Beal, J. Berliner, K. Hunter, "Fast Precise Distributed Control for Energy Demand Management", IEEE SASO 2012, Lyon, France.

[26] S. Frey et al., "Scenarios for an Autonomic Micro Smart Grid", Intl. Conf. on Smart Grids and Green IT Systems, Porto, Portugal, 2012. (4 pages)

[27] B.Becker et al "Decentralized Energy-Management to Control Smart-Home Architectures", Lect. Notes in Computer Science, Vol. 5974, 2010.

[28] H. Schmeck and L. Karg, "E-Energy – Paving the Way for an Internet of Energy", Information Technology, Vol. 52, No. 2, 2010, pp. 55-57.

[29] H. Hermanns, H. Wiechmann, "Demand-Response Management for Dependable Power Grids". Embedded Systems for Smart Appliances and Energy Management. Springer New York, 2013, pp 1-22.