A Decentralised Architecture for Multi-Objective Autonomic Management

Sylvain Frey
ENSIMAG (student)
Grenoble INP
Grenoble, France
sylvain.frey@ensimag.imag.fr

Philippe Lalanda
LIG Laboratory
University of Grenoble
Grenoble, France
philippe.lalanda@imag.fr

Ada Diaconescu
CNRS LTCI
Télécom ParisTech
Paris, France
ada.diaconescu@telecom-paristech.fr

***Abstract*—Designing and organising large numbers of autonomic resources into a coherent system is a difficult endeavour. It necessitates handling complex interactions among dynamic, heterogeneous components, autonomic managers and human policies. Several architectural models have been proposed for organising these interactions. This paper focuses on a decentralised approach, while also considering two other possibilities – centralised and hierarchical. An architectural model is proposed and a prototype implementation with corresponding experimental results are subsequently presented and discussed.**

***Keywords :* autonomic, architecture, decentralised, conflict**

I. Context and Vision

An architectural blueprint for autonomic computing from IBM [1] defined an autonomic system as "*a collection of self-managing resources*", with Autonomic Managers (AMs) organised according to "*design patterns that offer models for the structure and arrangement of components*". Heterogeneous, volatile software elements interacting in an unpredictable environment raise complex design and administration issues, notably due to multiple unpredictable interactions and incompatibilities – or *conflicts* – which may appear among the resources, as well as among their AMs.

Given the high specialisation of reusable domain-related components, administrators are not always able to understand the internal logic of the resources and the AMs they manage. This happens a fortiori when the administrator is not a computer engineering expert, as it would commonly be the case in a pervasive computing context (e.g. intelligent home applications). This situation reinforces the need for autonomic models that shelter administrators from complex resource-level management.

Furthermore, human administrators can also be a source of conflicts. Poor knowledge of the applicative layer leads to imprecise management policies that can prove conflicting at the lower, domain-specific level. Even an expert administrator may set up contradictory high-level instructions, such as 'maintain performance' and 'reduce resource consumption'. In our view, autonomic systems should be able to handle this sort of situation.

II. Architectural Discussion

A standard conflict situation is the following: several AMs administer the same software resource and try to set the same parameter according to their own policy. A centralised way of solving the conflict is to add an arbitrator, i.e. a higher-level AM. This arbitrator can decide which AM is "right", or get all the proposals and deduce its own final value. In contrast, in a decentralised solution the conflicting AMs are expected to find a "collective" solution by themselves, without necessitating any additional decision logic from a central controller.

While a centralised solution provides a better control of conflicts, it also necessitates a domain-related management logic which is not always available, or even conceivable. On the other hand, a decentralised, bottom-up architecture provides less control, but generates opportunistic behaviours that may compensate for the lack of well-defined solutions. The difficulty here lies in providing sufficient flexibility to these behaviours, while guaranteeing conformity with critical system requirements.

None of these two solutions provides a definitive answer to all the possible conflict-resolution issues. Centralised or decentralised control will be most suitable in different contexts. In a mixed solution, based on a hierarchical architecture, managers solve conflicts locally when possible and delegate to a higher-level manager otherwise. Conceiving such hierarchy requires an architecture providing both centralised and decentralised patterns.

III. Multi-objective Architecture

The aim of the proposed architecture is to enable the creation of multi-objective management systems by integrating reusable, domain-specific AMs dealing with a single management concern (e.g. performance or energy savings). Each resource is encapsulated into a membrane that contains all its AMs, as shown in Fig. 1. Communication is based on messages (publish/subscribe model).

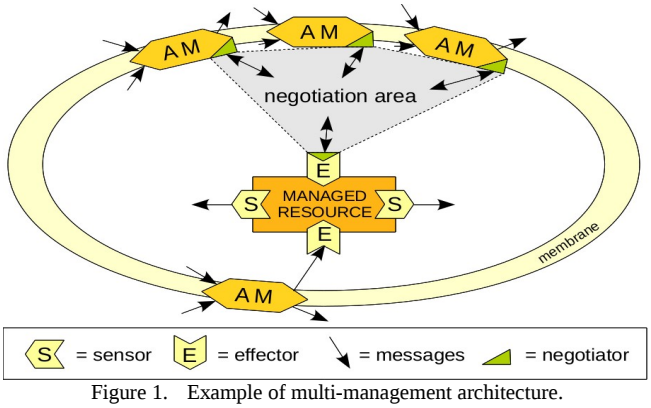

Figure 1. Example of multi-management architecture.

The resource's sensors publish data that can be read by AMs in the membrane only. Additionally, AMs may interact with the membrane's exterior (e.g. read user policies from dedicated "goal" topics or negotiate with AMs of other resources – see below). Effector instructions produced by conflicting AMs are intercepted by *negotiators*. Negotiators share a common negotiation area where they perform a collective synthesis of all concurrent AM proposals. Each negotiation area corresponds to one resource effector and provides a local conflict-resolution medium for AMs tackling different, possibly incompatible goals.

The actual nature of a negotiation area is a domain-specific question, depending on the context, on the nature of the AMs and on the managed resource. The negotiations may be based on direct message exchanges between negotiators or may necessitate additional services such as a blackboard. Some examples of distributed algorithms include: priority-based elicitation, computation of a mean, inhibition of concurrent negotiators or priority-based weighted sum. When a centralised resolution is conceivable and preferable, introducing an arbitrator is also a possibility. Adapting the resolution mechanisms to the context – possibly at run-time – is an important meta-management feature, notably when the resolution logic is not clearly defined (e.g. see part **V.)**.

Given this local architecture the overall application is constituted of interacting membranes. The important task of translating application-level semantics (e.g. user's goals) into resource-level semantics (e.g. setting an effector) is performed by the AMs in the membranes. As a consequence, user goal conflicts result in local effector conflicts and will be resolved as such.

Gathering conflicting AMs in negotiation areas is also the model for organising interactions between AMs of different resources. The difference is that inter-membrane negotiations deal with global, application-level issues whereas negotiations inside a membrane concern the setting of an effector. In both cases a standard domain-related negotiation protocol defining the semantics of the communications must be specified.

IV. Prototype Implementation

In order to simulate a dynamic, unpredictable but also intuitive environment the context of a house with autonomic equipments (heaters consuming electricity, thermometers, windows) was chosen. A simulator was implemented using **iPOJO** [2], a component-oriented service framework; the autonomic layer was developed using **Cilia** [3] - a data mediation framework built on top of **iPOJO**. Each room of the house is modelled according to Newton's laws of heat transfer and exchanges heat with its neighbours. The temperature of the outside world oscillates so as to simulate a day/night cycle. Heater power levels are adjusted by thermostats. In this context, each of the manageable equipments (e.g. heaters and windows) can be encapsulated in a management membrane. At the highest level the user can dynamically set management objectives, such as room temperatures, global electrical consumption and the relative priority between these two goals.

Each heater is managed by two conflicting AMs trying to set the thermostat according to their own objective: temperature or consumption. In the prototype, conflict resolution is performed by decentralised negotiators computing the final thermostat value as an average of proposed AM values, weighted by priorities deduced from simple high-level user policies (e.g. temperature more important than energy savings). Specific AMs opening and closing windows allow seeing indirect influences among resources (e.g. window and heater in the same room). Inter-membrane communications addressing this issue will be developed in our future work.

V. Results and Analysis

After some initial adaptation and tuning, the expected behaviour was obtained: the user could dynamically set different goals with variable priorities and the system self-adapted in order to reach them as closely as possible. External perturbations, such as a temperature fall, proved the system's capacity to stabilise. Further documentation and commented results are available online [4].

The main difficulty lied in finding suitable response times and amplitude values for the AMs' reactions to external changes. The choice of the negotiation mechanism (e.g. collective computation of a weighted mean vs. inhibition of concurrent negotiators) also necessitated several trials and failures. Such meta-management concerns are the key to the viability of the proposed management system. Clearly separating conflict resolution from AM logic, while providing dynamic adaptation of negotiators, are the first steps towards a systematisation of meta-management. Therefore, the discovery of viable management solutions through the use of automatic procedures (e.g. machine learning) may be envisaged.

VI. Conclusions and Future Work

This paper proposes a decentralised architecture for integrating heterogeneous AMs. Although the proposed architecture supports the introduction of centralised controllers and hierarchies of AMs, the presented implementation does not use them. These possibilities have been largely developed in **Ceylon** [5] which uses the same **iPOJO** technology. The strong similarities between these approaches suggest the possibility of a hybrid, hierarchical solution mixing centralised and decentralised control. Finally, the discovery of undefined autonomic behaviour through the use of machine-generated solutions would be a significant step towards the definition of generic conflict resolution mechanisms.

References

[1] IBM, Autonomic Computing: "An architectural blueprint for autonomic computing", white paper, fourth edition, 2006.

[2] iPOJO Project: www.ipojo.org

[3] CiliaMediation Project: www.ciliamediation.org

[4] perso.telecom-paristech.fr/~diacones/am-bb/results.pdf

[5] Y. Maurel, A. Diaconescu and P. Lalanda, "Creating complex, adaptable management strategies via the opportunistic integration of decentralised management resources", ICAIS, 2009.

[6] www2.ece.arizona.edu/~hpdc/projects/Autonomia_Programmable/

[7] AutoMate Project: www.caip.rutgers.edu/TASSL/Projects/AutoMate/

[8] Johann Bourcier, "Auto-Home: A Framework for Autonomic Pervasive Applications", PhD Thesis, Grenoble University, 2008.