ANS (Autonomic Networked System): A Position Paper

Julie A. McCann,
Department of Computing, Imperial College, London

Overview

Essentially the ANS (Autonomic Networked System) is a ubiquitous computing management tool, which is designed to mimic the ANS (Autonomic Nervous System) of living creatures. The organic ANS is the part of the nervous system controlling many organs and muscles within the body. However, we are unaware of the workings of the organic ANS because it functions in an involuntary, reflexive manner. What is particular about the organic ANS is that it is flexible, constantly in operation and that it happens in the background without our interference or knowledge of its mechanism. This is exactly what is needed to support ubiquitous computing environments, especially in the application of the 'intelligent home' and medical applications where constant technical support is impossible. Essentially the organic ANS is our body's resource manager, likewise the ubiquitous computing ANS Manager becomes the replacement operating system or virtual machine which provides the systems' ability configure and reconfigure itself under differing conditions and ever-changing environments. Further, such a system should provide the 'intelligence' to optimise its operation through constant monitoring and tuning to achieve its goal.

Currently, we do not visualise our computing technology to be disposable, such that we can throw away a faulty unit and replace it as easily as we would a light bulb. However this is exactly the paradigm required to achieve Weiser's vision of ubiquitous computing [Weiser 93]. Further, currently we expect the user to find ways to use the computer however we currently do not emphasise how the computer can find its own way to serve the user – a further paradigm shift. An application where these paradigms combine is the *intelligent home* where '*the system*' is the homes' complete computing power and consists of many computing units of varying capacities and basic functionality. We must be able to seamlessly plug and unplug units into '*the system'*. The functionality of these "ubiunits" would be determined by the context into which they are plugged or located. For example a *webpad* unit can be the remote control for your television when in the sitting room, the interface to your fridge in the kitchen or just idly hung on a the wall when not needed; possibly doing backups of system data to the network or ordering groceries to stock a depleted refrigerator. When a unit breaks or upgrades are required, new units can also be added seamlessly. This requires that the system have a high degree of automatic adaptability and reconfigurability since it is inconceivable in this context that we should require users to perform explicit systems management and maintenance.

FIGURE 1 ADAPTATION FRAMEWORK

(Arrows show the flow of control)

Basic ANS Architecture

Figure 1 illustrates the general component-based adaptive system architecture. The main feature is that this system can self-(re)configure with the help from monitors, which provide environmental data (e.g. current performance statistics). The adaptive technology is based on our previous work [Law00, McCann 00].

The typical tools used to develop such systems consist of architecture description languages (ADLs) and constraint solvers [Magee 95]. An architecture is generally considered to consist of components and their interactions. Architectural description languages allow the software engineer to formally abstract the architecture so as to reason about it. An ADL can give a global view of the system and when augmented with constraints, the validity of change (the reconfiguration of components) can potentially be evaluated at runtime [Garlan 97, Magee 95].
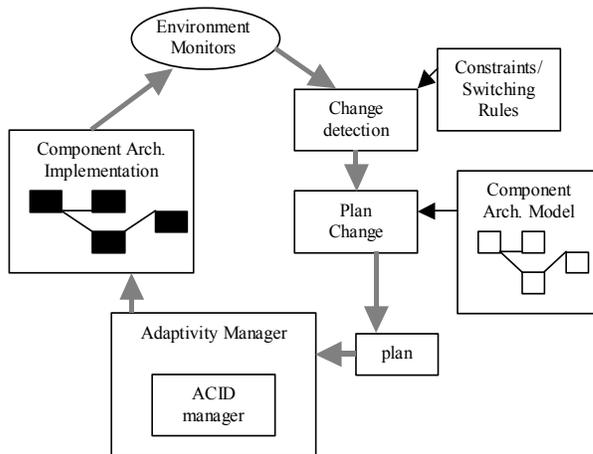
Sophisticated adaptive systems can be composed of components that in turn are composed of sub-components. In our architecture a component consists of both the application logic, the architectural description of itself (i.e. the component structure) and a copy of the switching rules relevant to it as well as a lightweight adaptivity manager. Further, each component follows the semantic interfacing rules using an ontology understood by all the components that enter that system. This ensures that the component binding will carry out the correct procedure. A session manager is fed information from monitors or *gauges* (which aggregate raw monitor data for more lightweight processing). The current configuration operation is being

monitored by the session monitor who constantly checks constraints and, if broken, consults the switching rules to decide how best to overcome the problem. The constraints can be anything from system management related constraints such as bandwidth becoming lessened to trust/security constraints such as 'is this device allowed to receive this data?'. When adaptivity is triggered the component architecture model allows an alternative execution plan to be designed. The session manager decides how to instantiate the alternative component architecture and passes his alternative over to the Adaptivity Manager. The Adaptivity Manager then carries out the unbinding and rebinding of components (establishing any glue necessary to achieve the binding). To do this it must ensure the instantiation adheres to transactional style prosperities. That is, the switch can be backed off if something goes wrong.

Another issue is being able to reconfigure the architecture to better fit the current processing and knowing which components to choose from the list available. Here semantic interfacing comes into play. That is, each component is described in terms of what it requires and what it provides at the level of data structure. However, what is unique about this architecture is that it further provides meaning to that interface. That is, what the data structure means and what the processing that that component carries out, actually means. This better enables the configuration process to assemble the best components for the reconfigured architecture.

Advancing the State of the Art

Currently most of the Autonomic computing and reflective self-configuring computing has been focused on single (parallel) machine and server cluster environments. Following its '*Autonomic Computing Manifesto*', released last year, IBM has earmarked the majority of its US $5.3 billion annual R&D budget to autonomic related research. Industry research and leading edge product examples come from Sun (N1 Dataserver), Motorola (Grid Engine), HP (Planetary Computing servers – particularly the Utility Data Center) and IBM's (iQueue data manage, eLiza and IceCube and AutoAdmin servers). Thus far the application of autonomic computing to highly heterogeneous ubiquitous sensor environments has not been carried out. A significant missing part of fine-grained component-based adaptive architecture computing is able to borrow from the Grid and Semantic Web research in terms of Semantic interfacing standards and methods. Finally this project will be the first (that we are aware of) that will combine the work on ADLs (architecture description languages) with that of adaptive systems architecture design to produce empirically tested software that will provide runtime adaptive environments.

Summary

Simply put, the resulting software emerging from this project will be designed in a component-based way. That is, the software's functionality is split into its elementary component parts so that functionality can be swapped in and out to reconfigure the overall architecture. Each software component consists of the code to carry out its function and a means to communicate how it does this to the other components in the system (we call this the semantic interface). This information is used when the overall system requires a new component to join it (i.e. the system will reconfigure) to allow it to choose the best component to do the job. The decision to reconfigure is made as a result of some monitoring component detecting change. Change can be in terms of a failed hardware unit or performance degradation. The aim of the system is to meet some form of Quality of Service (QoS) guarantees. When those guarantees cannot be met at a given time the system decides to reconfigure to try and overcome this. Before reconfiguring, the system checks that the reconfigured architecture will actually work (i.e. that combination of components is valid) so that misbehaviour such as deadlock is avoided.

Consequently, research carried to build such a system involves component-based software engineering and architecture modelling, knowledge of modern execution environments and tools, semantic interfacing and QoS definition and modelling. The ANS will be designed to integrate with the co-proposed projects in the Imperial College UbiCare Centre. Essentially the ANS will be concatenating the streamed data coming from the sensor systems developed by the UbiMon project (though the ANS project is not completely tied to this technologies).

References

[Garlan 97]Garlan D., Monroe R. T., Wile D., 'Acme: An Architecture Description Interchange Language', In *Proc. of CASCON '97*, November 1997

[Law 00] Law G., McCann, J.A., 'A New Protection Model for Component-Based Operating Systems', in proceedings of IEEE Conference on Computing and Communications, Phoenix, Arizona, Feb. 2000

[Magee 95] Magee J., Dulay N., Eisenbach S. Kramer J., Specifying Distributed Software Architectures. In *Proc. of the Fifth European Software Engineering Conference*, Barcelona, 1995.

[McCann 00] McCann J.A., Howlett P., Crane J.S., 'Kendra: Adaptive Internet System', Journal of Systems and Software, Elsevier Science, Volume 55, Issue 1, 5 November 2000 .pp 3-17.

[Weiser 93] Weiser M., 'Some computer science issues in ubiquitous computing', Communications of the ACM, vol. 36, no. 7, pp 75-84, July 1993