

# ANS (Autonomic Networked System) for Ubiquitous Computing

Julie A McCann  
Department of Computing  
Imperial College London  
London, SW7 1AZ, UK  
jamm@doc.ic.ac.uk

Asher Hoskins  
Department of Computing  
Imperial College London  
London, SW7 1AZ, UK  
asher@doc.ic.ac.uk

Gawesh Jawaheer  
Department of Computing  
Imperial College London  
London, SW7 1AZ, UK  
gawesh@doc.ic.ac.uk

## ABSTRACT

In this paper we describe the ANS system, which is middle-ware used to provide autonomic management of ubiquitous systems.

## Keywords

Ubiquitous computing, Autonomic, Self-adaptive systems

## 1. INTRODUCTION

The ANS (Autonomic Networked System) is a ubiquitous computing network management tool. Like the autonomic nervous system of living creatures it operates without the need for control by the user, functioning in an involuntary, reflexive manner in the background without their interference or knowledge of its mechanics. 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.

To achieve Weiser’s vision of ubiquitous computing [5] a ubiquitous network must have a high degree of automatic adaptability and reconfigurability since it is inconceivable that we should require users to perform explicit systems management and maintenance. The ANS is self-configuring, self-optimising, and self-repairing. The user can seamlessly plug and unplug units into and out of the network.

## 2. BASIC ANS ARCHITECTURE

An ANS network is composed of devices that all use the same simple ANS packet-level protocol to communicate, there is no distinction between “sources” and “sinks” as with some ubiquitous sensor network protocols. Devices may provide one or more “services”, essentially extra protocols that piggy-back onto the ANS, so that facilities offered by that device may be accessed by other devices. The ANS protocol is design such that data flowing to and from a service may

be cached and routed by ANS devices even if they do not understand that service’s protocol.

As well as traditional unicast point-to-point network routing the ANS allows multicasting by “attribute”. This allows packets to be, for example, routed to all devices at a particular physical (as opposed to network) location or to all devices owned by a particular person, allowing personal networks to spread far beyond personal space over the internet [4].

The ANS’s service discovery mechanisms allow for automatic self-configuration and self-optimisation of a system. Rather than an application having to know exactly which devices can provide what services it can request services (optionally using attribute-based routing to select where it would like the service provided) using a number of parameters to describe what it would like the service to provide. Devices that can provide a suitable service reply with details on how well they can provide it so that the application can choose the best.

Should a service provider fail then the application simply makes another service request (“retenders”). All ANS applications are written so as to be able to cope in the event that there is no suitable service available. Regular retendering allows applications to restart once a service becomes available again and to upgrade automatically to a better service provider should one join the network [3].

## 3. SOFTWARE DEVELOPMENT

As well as the ANS network protocols we are developing a dynamic component based software development system for use on ANS devices. Components are self-contained blocks of code linked together using an architecture description language (ADL) [2]. The ADL can specify multiple configurations of these components and so allow the application to reconfigure itself at runtime. The correctness of these configurations and the validity of the reconfigurations can potentially be evaluated at compile time [1, 2].

Runtime component reconfiguration makes the job of writing applications to deal with the naturally changeable nature of a ubiquitous computing environment much easier. Rather than large trees of conditional statements obfuscating the source code, components may be written to deal with a single case only and then switched around as the system changes.

## 4. ANS SCENARIO

In order to provide some boundaries for the ANS proof-of-concept that we wish to deliver we have developed a scenario showing how the ANS would assist in the care of a coronary heart disease patient.

The character in the scenario is called Jim, living in a house equipped with ANS driven sensors. Jim interacts with devices on the ANS by talking to ELLEN, an interactive virtual face. By aggregating observations from RFID tags, pressure sensors in cabinet shelves and vision sensors, an ANS application can monitor Jim's drug intake. Sensors embedded in the floor and in Jim's bed can be used to detect uncharacteristic patterns in his behaviour. New computing devices introduced into the environment such as the PDAs used by Jim's visiting nurse or by the paramedics who attended to him when he had his heart attack can readily discover and use services provided by the ANS. By dynamically reconfiguring the binding between components in the system, the ANS can react to unplanned events such as a failing sensor or Jim's heart condition deteriorating. For example, the ANS may rebind sensor and communication components in order to ensure that the monitoring of Jim's vital signs and the communication of this data to other agents are uninterrupted. Overall, the ANS must achieve its main goal of keeping Jim alive.

## 5. PROOF-OF-CONCEPT ROOM

To implement the scenario an office has been turned into a simulation of a house with different parts of the office representing different rooms and the outside of the office representing the "outside world". Various ubiquitous devices are fitted around and outside the room, forming the ANS. The patient communicates with ELLEN via various keypads and simple voice recognition systems. ELLEN can communicate with the patient as a talking head on a TV screen or via wireless speakers or text displays.

In a real system, the patient would wear a heart sensor. For this testbed a portable device simulates the signals this would generate. Buttons allow it to simulate various heart states so that the system can react to these.

RFID readers, microphones and a camera allow the system to monitor where the patient is and so adjust their outputs to suit. An intelligent bathroom cabinet monitors the patient's pill taking and allows the system to remind them when to do it.

While it would be possible to do a lot of this work using a pure software simulation, we believe that building hardware is useful in order to better study the interactions between devices. Providing realistic data to a simulation is often hard and one may overlook situations where, for example, multiple sensors overlap unintentionally and thus generate confusing signals which the system must deal with.

Unlike some other "intelligent home" setups which have been built, our focus is not on the interpretation of sensor data or the building of a particular application (our scenario is used purely to set some arbitrary boundaries). We are interested in the underlying infrastructure that would be required for a ubiquitous computing environment and

the problems caused by heterogeneous devices and devices with very limited computing power.

## 6. HARDWARE

The ANS testbed is comprised of two classes of hardware: PC-based and microcontroller-based. The PC-based devices use a small "Mini-ITX" format motherboard and run FreeBSD. They communicate with the outside world using an 802.11 wireless card and with the microcontroller devices via a wireless interface on the serial port.

The microcontroller-based devices, known as "Beasties", use an 8-bit Atmel ATmega8 microcontroller and a 433MHz FM radio module. They have a backplane expansion connector to allow daughterboards to be added. The daughterboards customise the Beasties for each particular application. The Beasties are designed to be simple to make and modify and so are constructed out of low-density through-hole chips on a simple double-sided board. The PC controls the more processor intensive parts of the testbed. It generates audio for ELLEN's voice, which can be received by (analogue FM) wireless speakers, and handles the graphics for ELLEN's talking head on a TV screen. Some patient position and motion information is captured by a small camera and processed on the PC before being sent out to the Beastie network. Everything else is based on a Beastie. Some of the daughter-boards contain their own microcontrollers in order to increase the number of interface pins available. These microcontrollers communicate with the primary microcontroller via an I<sup>2</sup>C bus.

## 7. ACKNOWLEDGMENTS

We thank the DTI for funding the ANS project and the UbiCare Centre to which this project belongs.

## 8. REFERENCES

- [1] D. Garlan, R. Monroe, and D. Wile. ACME: An architecture description interchange language. In *Proceedings of CASCON'97*, pages 169–183, Toronto, Ontario, Nov. 1997.
- [2] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying Distributed Software Architectures. In W. Schafer and P. Botella, editors, *Proc. 5th European Software Engineering Conf. (ESEC 95)*, volume 989, pages 137–153, Sitges, Spain, 1995. Springer-Verlag, Berlin.
- [3] J. McCann, P. Howlett, and J. Crane. Kendra: Adaptive internet system. *Journal of Systems and Software*, 55(1):3–17, Nov. 2000.
- [4] S. Pitchers and D. Eves. A protocol for body centric networks. In *IEE Eurowearable '03*, pages 87–92, Birmingham, UK, Sept. 2003.
- [5] M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, July 1993.