From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

24

Programming iSpaces — a Tale of Two Paradigms

V Callaghan, M Colley, H Hagras, J Chin, F Doctor and G Clarke

24.1 Introduction

'*iSpace, the final frontier*' — this parody of *Star Trek* encapsulates many of our aspirations for this area as, in the longer term, iSpaces are likely to be the key to mankind's successful exploration of deep space. In outer-space, or hostile planetary habitats, it is inevitable that people will survive in wholly technologically supported artificial environments [1]. Such environments will contain numerous communicating computers embedded into a myriad of devices, sensing, acting, delivering media, processing data and providing services that enhance the life-style and effectiveness of the occupant, and, in outer-space, preserving human life. Such environments will also include robots [2]. In today's iSpaces, while human life will not normally be at stake, the underlying principles and technology are much the same. Today our homes are rapidly being filled with diverse types of products ranging from simple lighting systems to sophisticated entertainment systems, all adding to the functionality and convenience available to the home user [3]. The iSpace approach envisages that, one day soon, most artefacts will contain embedded computers and network connections, opening up the possibility for hundreds of communicating devices, co-operating in communities serving the occupant(s). The seeds of this revolution have already been sown in that pervasive technologies such as the Internet and mobile telephones already boast over 200 and 680 million users, respectively [4, 5]. Today embedded computers account for 98% of all computer production, with an annual production of around 8 billion microprocessors [6], most being integrated into domestic appliances such as video recorders, washing machines, mobile telephones and all manner of everyday electronic appliances. Furthermore, nano-technology is opening up new possibilities such as embedding dust-particle-sized computers into hitherto unconventional mediums such as clothing fibres, paint pigments, etc. Thus, the embedded market is massive and ripe for the addition of networking to realise the iSpace vision. While these technological advances are fuelling significant changes in both the high-tech market-place and living environments, the most radical paradigm shift perhaps originated from the way these technologies can be applied. Firstly, communities of appliances can collaborate to provide new synergetic functionalities (e.g. a telephone ringing can be made to interact with other devices, such as pausing the TV), creating higher order 'virtual appliances'. Secondly, the nature of the device is being questioned; is it a traditional appliance with multiple prefixed functionalities or is it an appliance with

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

its constituent sub-functionalities, logically or physically decomposed (functional decomposition is intrinsic to the pervasive computing world). Thirdly, programming of key functionalities (e.g. co-ordinated community actions) is transferred from the manufacturer to the user, empowering end users to design novel functionalities that match their individual needs. When users are given the freedom to choose combinations of devices, then they can create unique and novel functionalities, some of which may not have been envisaged by the manufacturers, making preprogrammed solutions virtually impossible. One challenge, and the focus of much of the discussion in this chapter, is how to manage and configure (program) such co-ordinated pervasive computing devices to do the end user's bidding, without the user incurring prohibitive cognitive loads — a task that, without support, could quickly become prohibitive and an obstacle to the achievement of the pervasive home-networking environment vision. This chapter explores the issue of programming iSpaces by examining two possible approaches to supporting programming in the end user's environment — the use of autonomous intelligent embedded agents and the application of programming by example.

24.2 Degrees of Intelligence and Autonomy

For the iSpace vision to be realised in domestic environments, people must be able to use computer-based artefacts and systems in a way that gives them some control over aspects of the system, while eliminating cognitive awareness of parts of the system in which they have no interest, and are happy to leave to automation or implicit programming processes. Where the line between fully autonomous intelligent systems and manual programming should be drawn is a subject of much research and argument. At the University of Essex we have chosen to provide an approach that allows the full spectrum of possibilities to be experimented with; we have therefore developed a range of autonomous intelligent embedded agents and some user-centric techniques. In this chapter we present a review of all these techniques, although we shall start by describing our test-beds for intelligent spaces — the iDorm and the new iDorm-2.

24.3 The iDorm

The intelligent dormitory (iDorm) shown in Fig 24.1 is a real pervasive computing test-bed comprised of a large number of embedded sensors, actuators, processors and networks in the form of a student bed-sitting room. The iDorm is a multi-use, multi-user space containing areas for different activities such as sleep, work and entertaining. It contains the normal mix of furniture found in a typical student study/ bedroom environment, including a bed, work desk and a wardrobe.

A common interface to the iDorm and its devices is implemented through Universal Plug and Play (UPnP) which is an event-based communication middleware that allows devices to plug and play, thus enabling automatic discovery and configuration. A gateway server is used to run the UPnP software devices that interface with the hardware devices on their respective networks. Our experimental

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

Fig 24.1 The iDorm.

agent mechanisms are built on top of the low-level UPnP control architecture enabling it to communicate with the UPnP devices in the iDorm and thus allowing it to monitor and control these devices. Figure 24.2 shows the logical network infrastructure of the iDorm.

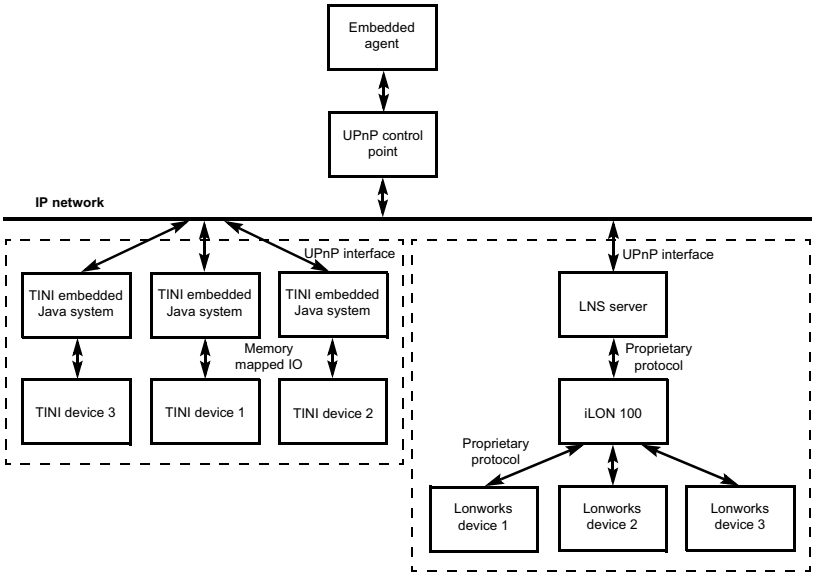

Fig 24.2 The iDorm logical network infrastructure.

Entertainment is one of the behaviours used as a benchmark in the iDorm for performance assessment in projects such as the BT led PHEN [7] project. There is a standard multimedia PC driving a flat-screen monitor and a video projector which can be used for both working and entertainment (see Fig 24.3).

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

Fig 24.3 Entertainment and work in the iDorm.

Any networked computer that can run a standard Java process can access and control the iDorm directly. Thus any PC can also act as an interface to control the devices in the room. Equally interfaces to the devices could be operated from wearable artefacts that can monitor and control the iDorm wirelessly such as a handheld PDA supporting Bluetooth wireless networking or a mobile telephone as shown in Fig 24.4. In principle, it is possible to adjust the environment from anywhere and at any time subject to user and device privileges. There is also an Internet fridge in the iDorm (see Fig 24.4d) that incorporates a PC with touchscreen capability, which can also be used to control the devices in the room. Control can of course still be exerted directly on the devices themselves via conventional switches, buttons, etc.

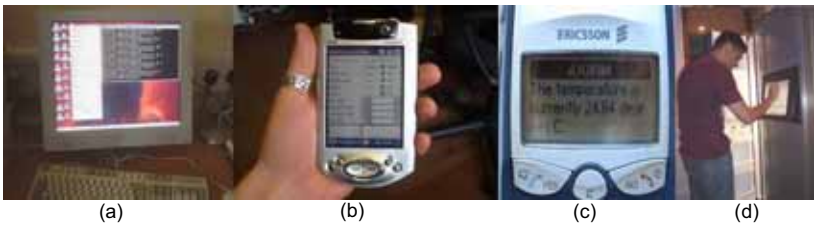

Fig 24.4 PC interfaces.

There are a variety of computers in the iDorm which are used to interface with sensors and actuators and run agents; all of them being configured as Java environments. At the low performance end we use TINI [8] and SNAP [9] embedded Internet boards, these are mainly used for sensors and actuators. There are also more powerful processor boards capable of running agents such as jStik [10] and ITX [11]. For experiments where maximum flexibility is required, it is also possible to run agents on UPnP enabled workstations. This allows the granularity of agent to device to be varied, from an agent controlling an entire environment, down to one-to-one mappings between devices and agents.

24.3.1 The iDorm-2

With the success of the iDorm, Essex University is currently constructing a new test-bed to support R&D in pervasive ICT. The new facility, funded by the HE SRIF programme takes the form of a domestic apartment and has been called iDorm-2.

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

The iDorm-2 has been built from the ground up to be an experimental pervasive computing environment with many special structural features such as cavity walls/ ceilings containing power and network outlets together with provision for internal wall-based sensors and processors, etc. There are numerous networks in place ranging from wired and power-line, through wireless, to broadband and high-bandwidth multi-mode fibre connections to the outside world. All the basic services are electrically controlled wherever possible (e.g. heating, water, doors). The basic layout of the apartment is show in Fig 24.5 (due to be opened in June 2005 at the international 'Intelligent Environments 05' workshop). When finished this will be one of few such facilities in the world.

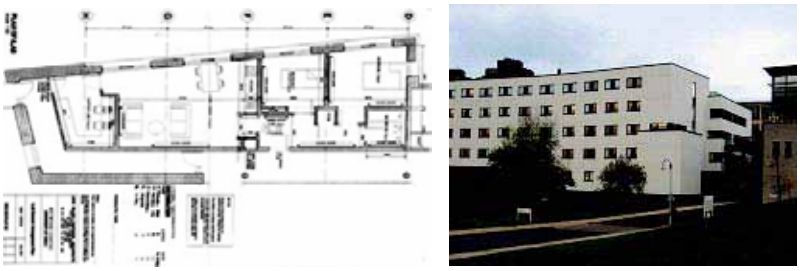

Fig 24.5 iDorm-2.

24.4 Embedded Agents

The principal argument in support of utilising artificial intelligence (AI) in support of the creation (programming) and management (control) of intelligent pervasive computing-based spaces is that much of the cognitive load associated with using the technology (which is an obstacle to market penetration) can be off-loaded from the user to software processes. However, this is far from easy as such 'intelligent entities' operate in a computationally complex and challenging physical environment which is significantly different to that encountered in more traditional PC programming or AI. Some of the computational challenges associated with creating systems of intelligent artefacts are discussed below.

24.4.1 Embedded Intelligence

Embedded intelligence can be regarded as the inclusion, in an artefact, of some of the reasoning, planning and learning that people possess. An intelligent artefact would normally contain only a minimal amount of 'embedded intelligence', sufficient to do the artefact task in question. Embedded computers that contain such an intelligent capability are normally referred to as 'embedded agents' [12]. Intelligent artefacts would, in effect, contain an embedded agent. Individually, such an embedded agent can harness intelligence to undertake such tasks as enhancing device functionality (i.e. enabling the artefact to do more complex control tasks), as

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

well as reducing configuration or programming complexity and costs by enabling the pervasive computing system to autonomously learn its own program rules, or alternatively assisting the lay end user to program rules in a non-technical way (see section 24.6.3).

24.4.2 Embedded Agents and Intelligent Spaces

There are a variety of approaches to this problem, perhaps the most relevant being those originating from the context-aware and embedded-agent communities. In embedded-agent work the goal is to utilise some form of AI to relieve the cognitive loading associated with setting up and running an iSpace system (i.e. transfer some of the cognitive processes from the person to the computer). Typically researchers have employed approaches such as neural networks, based on traditional machine learning theory, to control the users' environment. However, these approaches utilise objective functions that either aim to derive a minimal control function that satisfies the needs of the users' 'average' or are aimed at optimising between a number of competing needs (e.g. energy efficiency and user comfort). In both cases the user has little control over the system and has to accept some degree of discomfort, or adapt to the conditions determined by the iSpace agents [13].

A contrasting agent-based paradigm is to see the 'User as King' and create agents that '... particularise (rather than generalise) to a specific user's needs, and respond immediately to whatever the end user demands (providing it does not violate any safety constraints)' [14, 15].

Work at Essex University (as part of the EU's Disappearing Computer programme and the UK Government's UK-Korean Scientific fund) has addressed this problem using behaviour-based systems (pioneered by Brooks [16]) and soft-computing (fuzzy logic, neural networks and genetic algorithms). This approach stems from our finding that embedded agents used in pervasive computing are equivalent to robots, experiencing similar problems with sensing, non-determinism, intractability, embodiment, etc [14]. Our earlier work [15, 17, 18] was in the field of robotics, which has allowed us to recognise the underlying similarities between robotics and intelligent artefacts. Models in both robotics and pervasive embedded computer devices have proved difficult to devise, mainly because of the intractability of the variables involved (and in the case of modelling people, non-determinism). A principal advantage of behaviour-based methods is that they discard the need for an abstract model, replacing it by the world itself — a principle most aptly summarised by Brooks as '... the world is its own best model' [16].

24.4.3 Agent Learning

Learning can be viewed as the process of gathering information from the environment and encoding it to improve the efficiency of a system in achieving a certain goal. However, the difficulty that arises concerns finding the most appropriate learning algorithm/technique to use. Most learning algorithms use a measure of the quality of the solution, given either by examples of the desired behaviour of a system, or by an assessment of the quality of the internal and/or external state.

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

The learning algorithm very much depends on the characteristics of the 'problem' itself. The best choice of learning algorithm can be made by comparing the problem characteristics against the learning-algorithm characteristics. The following describes a limited number of these characteristics.

- Problem characteristics

  Dynamics — to what degree do the environment variables change during the learning?

  Complexity — is the set of all possible solutions, search space, finite/countable?

  Uncertainty — does the information regarding the state contain noise, and are the actions performed noisy?

  Pre-acquired knowledge — can some knowledge about the solutions be acquired before learning starts?

  Observability — is the current/past states known to the learning algorithm?

  Type of data — is the data provided discrete-valued, real-valued, and complex-structured or states and transitions?

  Feedback type — should the learning algorithm respond as an immediate, on-demand, delayed or no-response feedback?

  Physical limitations — what is the processing capability or memory size of the system where the learning algorithm runs?

- Learning-algorithm characteristics

  Internal parameter type — what type of parameter does the algorithm contain and how does it change?

  Input data — what kind of input data can the learning algorithm deal with and can it adapt to noisy data?

  Solution/goal type — can the learning algorithm produce approximations in real valued functions?

  Dynamics — can the solutions be changed during the environment's execution or can the learning algorithm only change the solutions off-line?

  Parameter change — what parameters change in each phase of the learning cycle, and do they change all at the same time or only a small subset?

Another important distinction in learning agents is whether the learning is done on-line or off-line. On-line learning means that the agent performs its tasks, and can learn or adapt after each event. On-line learning is like 'on-the-job' training and places a severe requirement on the learning algorithm. It must not only be fast but also very stable and fault-tolerant. Other hotly debated issues are whether supervised or unsupervised learning is best. Later we present the ISL and the AOFIS as examples of the unsupervised agent. The general challenges faced by designers of embedded-agents for such an environment were discussed at a recent workshop on Ubiquitous Computing in Domestic Environments [15].

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

24.4.4 Application Level Emergent Behaviour

In pervasive computing systems, the embedded-agent host (frequently an appliance) has a network connection allowing the agents to have a view of their neighbours, thereby facilitating co-ordinated actions from groups of embedded agents. The key difference to isolated appliances is that those participating in groups not only have their individual functionality (as designed by the manufacturer), but they also assume a group functionality that can be something that was not envisaged by the manufacturers. In fact, if there are only weak constraints on association of appliances, it is possible for the user to program unique co-ordinated actions (i.e. unique collective functionality) that was not envisaged by the different manufacturers offering the component appliances. This enables an application level emergent behaviour or functionality (something that, while enabled by the system, was not specified by the system). This naturally gives rise to questions such as the balance between prespecified functionality and emergent functionality, and what or who is responsible for the association between devices and the programming of the basic behaviours. Later in this chapter we discuss various approaches to this challenge. TOP provides an explicit means of directly harnessing user creativity to generate emergent applications while the ISL and AOFIS involve various degrees of user interaction using both supervised and unsupervised learning paradigms to generate emergent application-level functionality.

24.4.5 Machine Level Emergent Behaviour

In the behaviour-based approach to AI, the equivalent to reasoning and planning in traditional AI is produced by arranging for an agent to have a number of competing processes that are vying for control of the agent. The 'sensory context' determines the degree to which any process influences the agent. Thus, as sensing is derived from what is effectively a non-deterministic world, the solutions from this process are equally non-deterministic and result in what is termed 'emergent behaviour' (behaviours or solutions that emerged but were not explicitly programmed). Anything that affects the context can thus have a hand in this machine-level 'emergent behaviour'. For example, the connections (associations) between devices critically affect the sensed data. Thus agent-driven associations, or user driven associations, will be closely associated with emergent behaviour. Emergent behaviour is also sometimes described as emergent solutions. The freedom to make *ad hoc* associations is an important factor in this process, as without them it is difficult to see how emergent functionality could be achieved. At the University of Essex we are researching into what we term promiscuous association — the freedom for agents to form their own associations in as open a way as possible. This approach opens up the possibility of using formally specified ontologies of devices and groups of devices.

It is important to understand that being autonomous and promiscuous (open to making associations with other artefacts) does not imply undirected or unsafe behaviour. Agents can have basic fixed rules built into them that prevent them taking specified actions deemed unsafe.

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

24.4.6 Multi-Agents

The underlying paradigm of all Essex agents is that they are associated with actuators (they are essentially control agents rather than information processing agents). In the underlying agent model, multi-agent operation is supported via three modes. In the first, sensory and actuator parameters are simply made available to other agents. In the second mode, agents make a 'compressed' version of this information (or their internal state) available to the wider network. In a behaviour-based agent, such as the ISL, the compressed data takes the form of which behaviours are active (and to what degree). The general philosophy we have adopted is that data from remote agents is simply treated in the same way as all other sensor data. As with any data, the processing agent decides for itself which information is relevant to any particular decision. Thus, multi-agent processing is implicit to this paradigm, which regards remote agents as simply more sensors (albeit more sophisticated sensors).

We have found that receiving high-level processed information from remote agents, such as 'the iDorm is occupied' is more useful than being given the low level sensor information from the remote agent that gave rise to this higher-level characterisation.

This compressed form both relieves agent processing overheads and reduces network loading. A third approach we have developed is the use of inter-agent communication languages. Standardised agent communication languages (e.g. KQML and FIPA) tend to be too big to use on embedded-computers (many tens of megabytes) and are not well matched in terms of functionality to them. We have generated research that has looked at the problem of developing a lightweight agent communication language and the interested reader is referred to our description of the distributed intelligent building agent language (DIBAL) [19]. Finally, in the home environment (rather than a general unconstrained pervasive environment), because the number of connected appliances is relatively tractable (no more than a few hundred), a widely adopted approach, at a network level, is to fully connect all the appliances, relegating the issue of what appliance will collaborate with any other to the application level. This approach has been successfully applied by the University of Essex group [20, 21].

24.4.7 Knowledge in Rule-Based Agents

One reason we have opted for fuzzy logic rather than neural networks is that the knowledge acquired by the agent is gathered in human linguistic terms. A typical rules set from the iDorm is presented in Fig 24.6. It is made up of simple, if somewhat large *IF THEN ELSE* rule sets. Such rules are intrinsically well structured as they are based on mathematical logic sets.

Meta structures can also be used. For example, at the meta level rule sets can also be characterised according to context such as rule sets for Mr A relating to Context B (e.g. a bedroom). Thus, from such rule sets it is possible to perform meta functions such as deriving the closest rule set for a new user — Ms C — based upon rule sets from others users in the same context.

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

IF **InternalLightLevel** *is* **VVLOW** *AND* **ExternalLightLevel** *is* **VVLOW** *AND* **InternalTemperature** *is* **VVHIGH** *AND* **ExternalTemperature** *is* **MEDIUM** *AND* **ChairPressure** *is* **OFF** *AND* **BedPressure** *is* **ON** *AND* **Hour** *is* **Evening** *THEN* **ACTION_LIGHT1_value** *is* **VHIGH** *AND* **ACTION_Light2_value** *is* **HIGH** *AND* **ACTION_LIGHT3_value** *is* **LOW** *AND* **ACTION_Light4_value** *is* **VVLOW** *AND* **ACTION_Blind_state** *is* **CLOSED** *AND* **ACTION_Bedlight_state** *is* **ON** *AND* **ACTION_DeskLight_state** *is* **OFF** *AND* **ACTION_Heater_state** *is* **OFF** *AND* **ACTION_MSWord_state** *is* **STOPPED** *AND* **ACTION_MSMediaPlayer_state** *is* **RUNNING**

Fig 24.6 Example of rule representation.

24.5 Embedded-Agent-Based Approaches

At the University of Essex we have developed a number of agents that can deal with the problems discussed above. The main approaches we have developed are based on fuzzy logic. Fuzzy logic is particularly appropriate as it can describe inexact (and analogue) parameters using human-readable linguistic rules, offering a framework for representing imprecise and uncertain knowledge. Thus it is well suited to developing control on the basis of inexact sensing, and actuation which, when coupled to behaviour-based agent architectures, can deal with the non-determinism which sometimes characterises human behaviour. We believe this has similarities to the way people make decisions as it uses a mode of approximate reasoning, which allows it to deal with vague and incomplete information. We have shown that fuzzy logic can be applied well to a pervasive computing environment [22-24], such as the iDorm [25, 26], and have developed and tested two fuzzy-based embedded agents in the iDorm, namely the incremental synchronous learning agent [27] and the adaptive on-line fuzzy inference system agent [28, 29]. These agents have been run on commercial and in-house produced hardware. The photograph in Fig 24.7 shows a hardware networked agent platform produced at the University of Essex and used to manage the iDorm pervasive computing community.

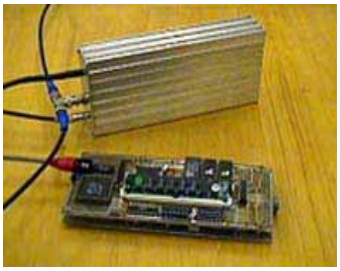

Fig 24.7 Agent prototype.

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

24.5.1 The Incremental Synchronous Learning (ISL) Agent

In general terms the ISL embedded agent[1] work is broadly situated within the behaviour-based architecture work pioneered by Brooks at MIT, consisting of many simple co-operating sub-control units. Our approach differs to other work in that we use fuzzy-logic-based sub-control units, arranging them in a hierarchy (see Fig 24.8) and employing a user-driven technique to learn the fuzzy rules on-line and in real time. It is well known that it is often difficult to determine parameters for fuzzy systems. In most fuzzy systems, the fuzzy rules were determined and tuned through trial and error by human operators. It normally takes many iterations to determine and tune them. As the number of input variables increases (intelligent space agents develop large numbers of rules due to particularisation), the number of rules increases disproportionately, which can cause difficulty in matching and choosing between large numbers of rules. Thus the introduction of a mechanism to learn fuzzy rules was a significant advance. In the ISL agent, we implement each behaviour as a fuzzy process and then use higher level fuzzy process to co-ordinate them. The resultant architecture takes the form of a hierarchical tree structure (as depicted in Fig 24.8). This approach has the following technical advantages:

- it simplifies the design of the embedded agent, reducing the number of rules to be determined (in previous work we have given examples of rules reduction of two orders of magnitude via the use of hierarchies);
- it uses the benefits of fuzzy logic to deal with imprecision and uncertainty;
- it provides a flexible structure where new behaviours can be added (e.g. comfort behaviours) or modified easily;
- it utilises a continuous activation scheme for behaviour co-ordination which provides a smoother response than switched schemata.

The learning process involves the creation of user behaviours. This is done interactively using reinforcement where the controller takes actions and monitors these actions to see if they satisfy the user or not, until a degree of satisfaction is achieved. The behaviours, resident inside the agent, take their input from sensors and appliances and adjust effector and appliance outputs (according to pre-determined, but settable, levels). The complexities of learning and negotiating satisfactory values for multiple users would depend upon having a reliable means of identifying different users.

It is clear that, in order for an appliance-based agent to autonomously particularise its service to an individual, some form of learning is essential [15]. In the ISL, learning takes the form of adapting the 'usage' behaviour rule base, according to the user's actions. To do this we utilise an evolutionary computing mechanism based on a novel hierarchical genetic algorithm (GA) technique which modifies the fuzzy controller rule sets through interaction with the environment and user.

The hub of the GA learning architecture is what we refer to as an Associative Experience Engine [30]. Briefly, each behaviour is a fuzzy logic controller (FLC) that has two parameters that can be modified — a rule base (RB) and its associated

[1] A detailed account of this agent, including the supporting theory and testing, can be found elsewhere [22-24].

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

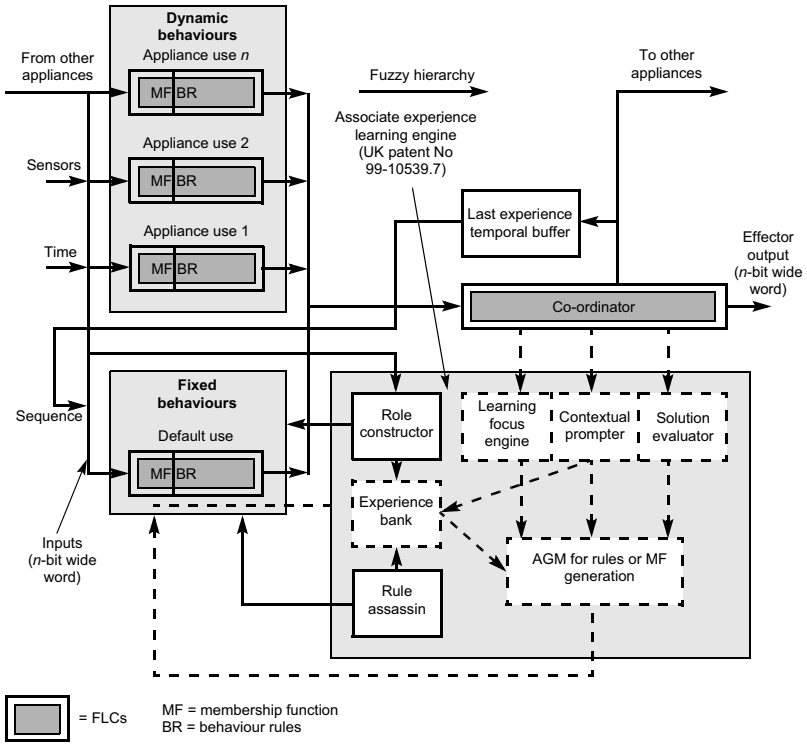

Fig 24.8 ISL embedded-agent architecture.

membership functions (MFs). In our learning we modify the rule-base. The architecture, as adapted for pervasive computing embedded agents, is shown in Fig 24.8. The behaviours receive their inputs from sensors and provide outputs to the actuators via the co-ordinator that weights their effect. When the system fails to have the desired response (e.g. an occupant manually changes an effector setting), the learning cycle begins.

When a learning cycle is initiated, the most active behaviour (i.e. that most responsible for the agent behaviour) is provided to the learning focus from the co-ordinator (the fuzzy engine which weights contributions to the outputs), which uses the information to point at the rule set to be modified (i.e. learnt) or exchanged. Initially, the contextual prompter (which gets a characterisation of the situation, an experience, from the co-ordinator) is used to make comparison to see whether there is a suitable behaviour rule set in the experience bank. If there is a suitable experience, it is used. When the past experiences do not satisfy the occupant's needs we use the best-fit experiences to reduce the search space by pointing to a better starting point, which is the experience with the largest fitness. We then fire an adaptive genetic mechanism (AGM) using adaptive learning parameters to speed the search for new solutions. The AGM is constrained to produce new solutions in a certain range defined by the contextual prompter to avoid the AGM searching

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

options where solutions are unlikely to be found. By using these mechanisms we narrow the AGM search space massively, thus improving its efficiency. After generating new solutions the system tests the new solution and gives it fitness through the solution evaluator. The AGM provides new options via operators such as crossover and mutation until a satisfactory solution is achieved.

The system then remains with this set of active rules (an experience) until the user's behaviour indicates a change of preference (e.g. has developed a new habit), signalled by a manual change to one of the effectors when the learning process described above is repeated. In the case of a new occupant in the room the contextual prompter gets and activates the most suitable rule base from the experience bank or if this proves unsuitable the system re-starts the learning cycle above. The solution evaluator assigns each stored rule base in the experience bank a fitness value. When the experience bank is full, we have to delete some experiences. To assist with this, the rule assassin determines which rules are removed according to their importance (as set by the solution evaluator). The last experience temporal buffer feeds back to the inputs a compressed form of the $n-1$ state, thereby providing a mechanism to deal with temporal sequences.

24.5.2 Adaptive On-line Fuzzy Inference System (AOFIS) Agent

Like the ISL agent, AOFIS is based on fuzzy logic. We utilise an unsupervised data-driven one-pass approach for extracting fuzzy rules and membership functions from data to learn a fuzzy logic controller (FLC) that will model the user's behaviours when using iDorm-based devices. It differs from the ISL in that it not only learns controller rules, but it also learns membership functions (a significant advance on the ISL which has fixed membership functions). The data is collected by monitoring the user's occupation of the iDorm over a period of time. The learnt FLC provides an inference mechanism that produces output control responses based on the current state of the inputs. The AOFIS adaptive FLC will therefore control a pervasive computing community, such as the iDorm, on behalf of the user and will also allow the rules to be adapted on-line as the user's behaviour changes over time. This approach aims to realise the vision of ambient intelligence and support the aims of pervasive computing in the following ways:

- the agent is responsive to the particular needs and preferences of the user;
- the user is always in control and can override the agent at any time;
- the agent learns and controls its environment in a non-intrusive way (although users may be aware of the high-tech interface, they are unaware of the agent's presence);
- the agent uses a simple one pass learning mechanism for learning the user's behaviours, and thus it is not computationally expensive;
- the agent's learnt behaviours can be adapted on-line as a result of changes in the user's behaviour;
- learning is life-long in that agent behaviours can be adapted and extended over a long period of time as a result of changes in the pervasive computing environment.

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

AOFIS involves five phases — monitoring the user's interactions and capturing input/output data associated with their actions, extraction of the fuzzy membership functions from the data, extraction of the fuzzy rules from the recorded data, the agent control, and the life-long learning and adaptation mechanism. The last two phases are control loops that once initiated receive inputs as either monitored sensor changes that produce appropriate output control responses based on the set of learnt rules, or user action requests that cause the learnt rules to be adapted before an appropriate output control response is produced. These five phases are illustrated in Fig 24.9.

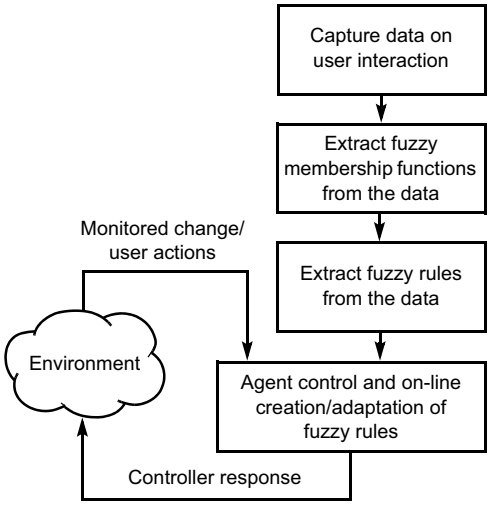

Fig 24.9 Phases of AOFIS.

The agent initially monitors the user's actions in the environment. Whenever the user changes actuator settings, the agent records a 'snapshot'of the current inputs (sensor states) and the outputs (actuator states with the new values of whichever actuators were adjusted by the user). These 'snapshots' are accumulated over a period of time so that the agent observes as much of the user's interactions within the environment as possible. AOFIS learns a descriptive model of the user's behaviours from the data accumulated by the agent. In our experiments in the iDorm we used seven sensors for our inputs and ten actuators for our outputs with a user spending up to three days in the iDorm. The fuzzy rules which are extracted represent local models that map a set of inputs to the set of outputs without the need for formulating any mathematical model. Individual rules can therefore be adapted on-line influencing only specific parts of the descriptive model learnt by the agent.

It is necessary to be able to categorise the accumulated user input/output data into a set of fuzzy membership functions which quantify the raw crisp values of the sensors and actuators into linguistic labels. AOFIS is based on learning the particularised behaviours of the user and therefore requires that these membership functions be defined from the user's input/output data recorded by the agent. A double clustering approach, combining Fuzzy-C-Means (FCM) and hierarchical clustering, is used for extracting fuzzy membership functions from the user data.

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

This is a simple and effective approach where the objective is to build models at a certain level of information granularity that can be quantified in terms of fuzzy sets.

Once the agent has extracted the membership functions and the set of rules from the user input/output data, it has then learnt the FLC that captures the human behaviour. The agent FLC can start controlling the pervasive computing community on behalf of the user. The agent starts to monitor the state of the pervasive community and affect actuators based on its learnt FLC that approximate the particularised preferences of the user. Figure 24.10 illustrates the FLC which consists of a fuzzifier, rule base, fuzzy inference engine and defuzzifier.

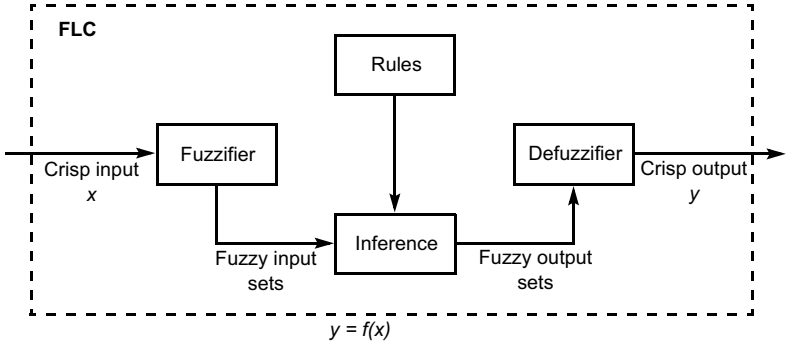

Fig 24.10 AOFIS FLC.

In conformity with the non-intrusive aspect of intelligence [28, 29], whenever the user is not happy with the behaviour of the pervasive computing device or community, the agent's control responses can always be overridden by simply altering the manual control of the system. When this occurs the agent will adapt its rules on-line or add new rules based on the new user preferences. This process incorporates what we term 'learning inertia' where the agent delays adapting its learnt rules until the user preference for changing a particular set of actuator values has re-occurred a number of times. This prevents the agent adapting its rules in response to 'one-off' user actions that do not reflect a marked change in the user's habitual behaviour (this 'learning inertia' parameter is user settable). As rules are adapted it is sensible to preserve old rules so they can be recalled by the agent in the future if they are more appropriate than the current rules. Whenever the user overrides the agent's control outputs and overrides any of the controlled output devices, a snapshot of the state of the environment is recorded and passed to the rule adaptation routine. The AOFIS agent supports the notion of life-long learning in that it adapts its rules as the state of the pervasive community and the user preferences vary over a significantly long period of time. Due to the flexibility of AOFIS, the initially learnt FLC can be easily extended to both adapt existing rules, as well as add new rules. The fuzzy nature of the rules permits them to capture a wide range of values for each input and output parameter. This allows the rules to continue to operate even if there is a gradual change in the environment. If, however, there is a significant change in the environment or the user's activity is no longer captured by the existing rules, then the agent will automatically create new rules that satisfy the current conditions. The agent will therefore unobtrusively and incrementally extend

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

its behaviours which can then be adapted to satisfy a pervasive device and community user.

24.5.3 Benchmarking and Comparative Performance

We have also implemented other soft computing agents, namely genetic programming (GP), the adaptive-neuro fuzzy inference system (ANFIS) and the multi-layer perceptron neural network. The dataset obtained from the iDorm (Fig 24.11) during the AOFIS monitoring phase comprised 408 instances and was randomised into six samples. Each sample was then split into a training and test set consisting of 272 and 136 instances respectively. The performance error for each technique was obtained on the test instances as the root mean squared error which was also scaled to account for the different ranges of the output parameters.

Fig 24.11 User gathering experimental data in the iDorm.

The GP used a population of 200 individuals evolving them over 200 generations. The GP evolved both the rules and the fuzzy sets. Each individual was represented as a tree composed of 'AND' and 'OR' operators as the internal nodes and triangular and trapezoidal membership functions as terminal nodes. The parameters of the membership functions were also evolved in parallel with the structure. The search started with a randomly generated set of rules and parameters, which were then optimised by means of genetic operators. The GP-based approach for optimising an FLC was tested with different numbers of fuzzy sets. In ANFIS, subtractive clustering was used to generate an initial TSK-type fuzzy inference system. Back propagation was used to learn the premise parameters while least square estimation was used to determine the consequent parameters.

An iteration of the learning procedure consisted of two parts, where the first part propagated the input patterns and estimated optimal consequent parameters through an iterative least squares procedure, and the second part used back propagation to modify the antecedent membership functions.

We tested ANFIS with a range of different cluster radii values. The multi-layer perceptron (MLP) back-propagation neural network was tested with different numbers of hidden nodes in a single hidden layer. We tested the AOFIS with different numbers of fuzzy sets and the membership function overlap threshold was set to 0.5 as this gave both a sufficient degree of overlap while also allowing the system to distinguish between the ranges covered by each fuzzy set. Tables 24.1 and 24.2 illustrate the scaled root mean squared error (RMSE) and scaled standard deviation (STD) for each technique averaged over the six randomised samples, and corresponding to the values of the variable parameter tested for each approach. The

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

Table 24.1 Average scaled RMSE.

Average scaled root mean squared error (SRMSE) for six randomised samples of the dataset							
AOFIS		GA		ANFIS		MLP	
Number of fuzzy sets	SRMSE	Number of fuzzy sets	SRMSE	Cluster radii	SRMSE	Number of hidden nodes	SRMSE
2	0.2148	2	0.1235	0.3	1.3269	2	0.2129
3	0.1476	3	0.1156	0.4	0.9229	4	0.1718
4	0.1461	4	0.1189	0.5	0.2582	6	0.1732
5	0.1364	5	0.1106	0.6	0.1661	8	0.1571
6	0.1352	6	0.1210	0.7	0.1669	10	0.1555
7	0.1261	7	0.1193	0.8	0.1418	20	0.1621
8	0.1326	8	0.1173	0.9	0.1213	30	0.1705
9	0.1472	9	0.1202	1.0	0.1157	40	0.1667
10	0.1537	10	0.1235	1.1	0.1201	50	0.1768
11	0.1696	11	0.1110	1.2	0.1168	60	0.1711
12	0.1999	12	0.1201	1.3	0.1131	70	0.1712
13	0.2246	13	0.1169	1.4	0.1131	80	0.1770
14	0.2337	14	0.1120	1.5	0.1118	90	0.1767
15	0.2460	15	0.1089	1.6	0.1130	100	0.1924
16	0.2459	16	0.1225	1.7	0.1115	200	0.2027
17	0.2732	17	0.1146	1.8	0.1137	300	0.2258
18	0.2747	18	0.1188	1.9	0.1182	400	0.2365
19	0.2771	19	0.1159	2.0	0.1189	500	0.2424
20	0.2839	20	0.1143				

Table 24.2 Average scaled STD

Average scaled standard deviation (SSTD) for six randomised samples of the dataset							
AOFIS		GA		ANFIS		MLP	
Number of fuzzy sets	SSTD	Number of fuzzy sets	SSTD	Cluster radii	SSTD	Number of hidden nodes	SSTD
2	0.1896	2	0.1128	0.3	1.2839	2	0.1499
3	0.1350	3	0.1063	0.4	0.9001	4	0.1299
4	0.1354	4	0.1094	0.5	0.2440	6	0.1277
5	0.1277	5	0.1026	0.6	0.1522	8	0.1193
6	0.1280	6	0.1121	0.7	0.1518	10	0.1160
7	0.1200	7	0.1107	0.8	0.1257	20	0.1198
8	0.1266	8	0.1085	0.9	0.1038	30	0.1229
9	0.1409	9	0.1117	1.0	0.0972	40	0.1234
10	0.1472	10	0.1145	1.1	0.1007	50	0.1245
11	0.1626	11	0.1026	1.2	0.0961	60	0.1234
12	0.1912	12	0.1115	1.3	0.0920	70	0.1222
13	0.2133	13	0.1084	1.4	0.0924	80	0.1283
14	0.2218	14	0.1031	1.5	0.0906	90	0.1272
15	0.2323	15	0.1007	1.6	0.0911	100	0.1333
16	0.2318	16	0.1128	1.7	0.0891	200	0.1366
17	0.2557	17	0.1063	1.8	0.0909	300	0.1503
18	0.2568	18	0.1090	1.9	0.0951	400	0.1674
19	0.2588	19	0.1075	2.0	0.0937	500	0.1676
20	0.2646	20	0.1051				

results in Tables 24.1 and 24.2 show that the optimum number of fuzzy sets for AOFIS was 7 and on average AOFIS produced 186 rules. The GP in comparison gave a marginally lower error for 7 fuzzy sets. Both ANFIS and the MLP on average gave a higher error than AOFIS. The ANFIS only learns a multi-input single-output (MISO) FLC and had to be run repeatedly for each output parameter. The FLC

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

produced was therefore only representative of an MISO system. Another restriction with ANFIS was that it generates TSK FLCs, where the consequent parameters are represented as either linear or constant values, rather than linguistic variables as is the case with Mamdani FLCs.

These linguistic variables are very important to understanding the human behaviour. It should be noted that the AOFIS generates multi-input-multi-output (MIMO) Mamdani FLCs representing rules in a more descriptive human-readable form which is advantageous for pervasive computing communities or other ambient intelligent systems, as they deal with people whose behaviours are more easily described in such linguistic terms. The iterative-nature of the GP makes it highly computationally intensive and this also applies to both ANFIS and the MLP which are also iterative-based approaches.

AOFIS is far less computationally intensive due to the one-pass procedure it employs, and is therefore more favourable for an embedded agent. Neither ANFIS nor the GP-based approach can easily be adapted on-line as this would require their internal structures to be re-learnt if either new rules were to be added or existing rules were adapted.

Therefore the AOFIS method is unique in that it can learn a good model of the user's behaviour which can then be adapted on-line in a life-long mode, in a non-intrusive manner, unlike other methods which need to repeat time-consuming learning cycles to adapt the user's behaviour.

Hence, in summary, the AOFIS agent proved to be the best for on-line learning and adaptation; moreover it was computationally less intensive and better suited to on-line learning than the other approaches compared. Finally, at the outset of our work it was not clear how long (if any time at all) it would take for such learning in this type of environment to reach a steady state. Our initial results (see Fig 24.12) indicate this is possible within a day although we would need to conduct experiences over much longer periods to catch other cycles, such as annual climate-related variations.

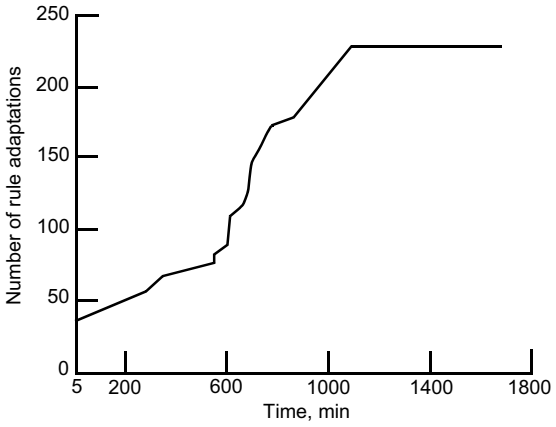

Fig 24.12 Typical learning rate of FLC-based agent.

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

24.6 An End-User Programming-Based Approach

24.6.1 Discussion

While autonomous agents may appeal to many people, their acceptance is not universal. Some lay-people distrust autonomous agents and prefer to exercise direct control over what is being learnt, when it is being learnt and to whom (or what) the information is communicated. These concerns are particularly acute when such technology is in the private space of our homes. Often, end users are given very little, if any, choice in setting up the system to their likings, but rather, they are required to 'surrender their rights' and 'put up with' whatever is provided [31].

Moreover, there are other reasons advanced in support of a more human driven involvement, such as exploiting the creative talents of people by providing them with the means to become designers of their own 'pervasive computing spaces', while at the same time shielding them from unnecessary technical details. To explore this aspect of our inhabited intelligent environment work we have recently opened up a complementary strand of research which we refer to as task-oriented programming (TOP) based on a combination of programming by example (PBE) (sometimes referred to as programming by demonstration — PBD), pioneered by Smith in the mid-seventies [32], learning from the user (LFU), the paradigm Essex University has been developing for many years, and ontologies (the latter mainly drawn from research work on the Semantic Web) [33]. It is based on a vision to put the user at the centre of the system programming experience by exchanging implicit autonomous learning for explicit user driven teaching. In this approach a user defines a community of co-ordinating pervasive devices and then 'programs' it by physically operating the system to mimic the required behaviour, i.e. 'programming by example' [34].

24.6.2 Programming by Example

Programming by example (PBE) was introduced by Smith in the mid-seventies, where the algorithms for the system functionalities were not described abstractly but rather demonstrated in concrete examples [32]. Henry Lieberman later described PBE as '... a software agent that records the interactions between the user and a conventional direct-manipulation interface and writes a program that corresponds to the user's actions', where 'the agent can then generalise the program so that it can work in other simulations similar to, but not necessarily exactly the same as, the example on which it is taught' [35]. Thus, PBE reduces the gap between the user interactions and the delivered program functionality by merging the two tasks. The main area of PBE work has focused on graphical user interfaces running on PCs. For instance PBE has been applied to computer application development [36, 37], computer-aided design (CAD) system [38], children's programs [39-41] and World Wide Web related technologies [42-46]. The underlying principles in PBE are generic and transportable to the pervasive computing world. In addition to the underlying scientific principle, PBE shares the same motivation of empowering lay end users to utilise what would otherwise be prohibitively complex technology.

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

However, to-date PBE has not been applied to programming tangible physical objects, nor any other aspect of pervasive computing. Thus TOP is the first application of PBE to this area.

24.6.3 Task-Oriented Programming

TOP was proposed and developed by Chin in 2003 as a means to address the issues of privacy and creativity in iSpaces [31]. In the TOP approach, the system is explicitly put into a learning mode and taught (by demonstration) how to behave by the lay end user. For example the TV or sitting room light could be made to react to an incoming call on the telephone. Thus the telephone, TV and light co-ordinate their actions to form a new meta (virtual) appliance. The vision goes beyond linking only conventional appliances. For instance if a network capability is added to an appliance, it becomes possible to allow its functional units to be shared with others. Thus in this notion, the audio amplifier in a TV could be made use of by the HiFi system, or vice versa. Consequently, 'virtual appliances' could be created by establishing logical connections between the sub-functions of appliances, creating replicas of traditional appliances, or inventing altogether new appliances. This decomposition of traditional appliances into their atomic functionalities (either physically or logically) and later allowing users to re-compose 'virtual appliances' (nuclear functions), by simply reconnecting these basic atomic functionalities together, is the paradigm we called 'the decomposed appliance' model. The key to creating 'virtual appliances' from decomposed functions is that of making connections between sub-functions so that a closed set of interconnected functions becomes a global set of functions (i.e. it becomes a 'community', or a collective of co-ordinating devices with a meta functionality). To facilitate this it is necessary to have some standard way of describing the functionality of the devices and connections — for TOP we have therefore devised an ontology, dComp (see section 24.6.4). Clearly, this concept of 'community' is not limited to decomposed appliances, but relates to any set of co-ordinated pervasive entities, whatever their functional or physical level (e.g. it could also relate to nano-scale or even macro-scale building-to-building environments). In general, a richer pool of sub-functions will lead to greater combinations or permutations for the user to create new virtual appliances.

As TOP has the notion of working with communities, the system supports setting up communities (if they do not already exist) (see Fig 24.13). Then, by selecting any community that the user wishes to program, a set of co-ordinated actions are taught to the system by simply using the home networked devices in a role-play mode, supported by some on-screen activities An action causes an appliance to generate an associated event, and this event is then used to generate appropriate rules (based on a 'snapshot' of the environment state). To be more general, co-ordinating actions (i.e. tasks) are performed by a community (i.e. one or more devices). A device can be involved in more than one community (i.e. performing one or more actions). The designer (user) interface with TOP is via an editor called 'TOPeditor'. This editor provides a means for:

- setting up/amending communities;
- holding teaching sessions so that tasks can also be taught (via the editor) as well as via physical usage of networked devices.

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

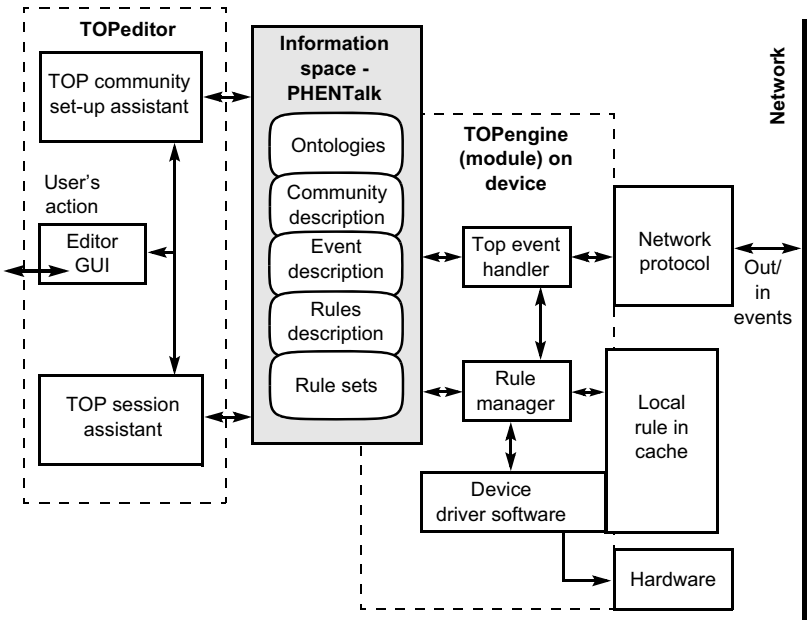

Fig 24.13 The TOP architecture.

The TOP architecture has two distinct modules — a 'TOPeditor' (to program the systems) and a 'TOPengine' (to execute the user generated rules). The TOPeditor has two main components. The first component, a 'TOP community set-up assistant', allows the user to set up groups (communities) of devices that can communicate and co-ordinate their actions to produce some desired meta function (or virtual appliance). The second component, the TOPengine, is a process that runs inside each and every networked device and executes the taught rules. It has three main components:

- a 'TOP event handler' that monitors connected devices, forming rules based on a user's interaction with the networked devices (interactions generating 'events' managed by an underlying UPnP middleware) — user interactions can be direct (e.g. activating a control) or indirect (e.g. activating a telephone by dialling in from another phone);
- the second component, a 'rule manager', manages the addition and removal of rules from memory, which includes removing dormant rules to make space for newer rules, and checking for duplication or conflicts;
- the third module, 'local rule cache', acts as a temporary rule buffer while rules are being built by the user (i.e. while the user is still designing and experimenting with creating community functionality).

To facilitate the information to be used within and beyond the community, data needs to be standardised so that it can be understood by all other parties in the network. For this aspect of work, the semantics in the TOP dComp ontology

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

supports information interoperability between applications, providing a common machine 'understanding' knowledge framework.

24.6.4 The dComp Ontology

TOP leverages ontology semantics as the core vocabulary for its information space, generating ontology-based rule sets when a user demonstrates their desired tasks to the system in a 'teaching' session. As explained in the previous section, these rule sets are then interpreted and executed by a back-end execution engine — the 'TOPengine'. As ontology allows information to be conceptually specified in an explicit way by providing definitions associated with names of entities in a universe of discourse (e.g. classes, relations, functions, or other objects) that are in both machine- and human-usable format. Thus, in more practical terms, ontology describes things such as what a device name means, and provides formal axioms that constrain the form and interpretation of these terms. Most ontology tools support descriptions of behaviour based on rules — hence an ontology-based approach is well suited to the challenges TOP faces.

We have chosen to base dComp around the OWL language as it is more expressive than RDF or RDF-S (i.e. provides additional formal semantic vocabularies allowing us to embed more information into our ontology) and is widely used (especially for the Semantic Web), with numerous supporting tools such as Jena [47] and inference engines such as RACER [48], F-OWL [49], Construct [50]. In order to realise our vision, a set of explicitly well-defined vocabularies (i.e. an ontology) is needed to model not just the basic concept of decomposed devices, but also the communities they form, the services they provide, the rules and policies they follow, the resultant actions that they take, and of course the people who inhabit the environment along with their individual preferences — dComp provides these properties. Wherever possible we have sought to build on existing work. The SOUPA ontology (from Ubicomp) is aimed at pervasive computing but lacks support for key TOP mechanisms such as community, decomposed functions and co-ordinating actions (to produce higher level meta-functionality) [51]. In addition, the current SOUPA standard has only limited support for the concept of pervasive home UPnP-based devices (on which TOP depends). However, SOUPA has a well-defined method of supporting notions of action, person, policy and time, which dComp has adopted. Thus most of the innovation in dComp relates to the ontology of decomposition and community — hence the name 'Decomposed Community Programming' (dComp).

The following is a summarised walk-through of the full dComp specification which is described more fully on-line[2] and in other papers [47]. To avoid any confusion in terminology, henceforth we refer to the dComp ontology as 'the ontology' whereas the documents that describe a certain concept of entities (e.g. device, services, community) that exist in the dComp environment are referred to as 'ontology documents'. Ontology also describes a set of properties and relationships that are associated with these concepts, along with the restrictions they may have. The current version (v.1.1) of the dComp ontology consists of 10 classes (see Fig 24.14).

[2] dComp ontology can be retrieved from: http://iieg.essex.ac.uk/dcomp/ont/dev/2004/05/

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

DCOMPDevice Class	**DCOMPHardware Class**	**DCOMPService Class**	**Rule Class**	**Policy Class**
DCOMPDevice	Hardware	DCOMPService	Rule	Policy
MobileDevice	CPU	LightsAndFittingsService	UnchangeableRule	Mode
StaticDevice	Memory	LightService	PersistentRule	
NomadicDevice	DisplayOutput	SwitchService	NonPersistentRule	**Time Class**
Light	DisplayScreenProperty	TelephoneService	Preceding	
Switch	AudioOutput	AlarmService	Device	**DCOMPperson Class**
Telephone	AudioOutputProperty	TemperatureService		
Alarm	Tuner	EntertainmentService		**Action Class**
Blind	Amplifier	AudioService	**Preference Class**	Action
Heater	**DCOMPCommunity Class**	VideoService	Preference	PermittedAction
FileRepository	Community	FollowMeService	SituationCondition	ForbiddenAction
DisplayDevice	NotJointCommunity	SetTopBoxService	CommunityPreference	Recipient
AudioDevice	PersistentCommunity	StateVariable		TargetAction
SetTopBox	TransitoryCommunity	TOPService		
Characteristic	CommunityDevice			
DeviceInfo				

Fig 24.14 dComp ontology (v1.1).

Device Class

The main class called 'DCOMPDevice' provides a generic description of any devices. Currently DCOMPDevice has ten sub-classes including both nuclear (traditional appliances) and atomic (decomposed) devices and remains the subject of ongoing development. The roles of most sub-classes are obvious from their names. Those which might not be obvious include 'DeviceInfo' for devices sharing some UPnP descriptions, 'Characteristic' for different mobility characteristics, and Relationships which are defined by using the OWL object property[3] and are:

- hasDeviceInfo;
- hasHardwareProperty;
- hasDCOMPService;
- hasCharacteristic.

The main elements of a typical DCOMPDevice expression are shown in Fig 24.15.

Hardware Class

The abstract class, DCOMPHardware, generalises all hardware that exists in a DCOMPDevice and, in the current version, has eight sub-classes along with associated properties — CPU, Memory, DisplayOutput, DisplayInput, AudioOutput, AudioInput, Amplifier and Tuner. In order for the DCOMPDevices to work together, every DCOMPDevice on the dComp network offers services. These services are modelled by a class called 'DCOMPService' which currently contains three sub-classes, namely TOPService, LightsAndFittingsService and EntertainmentService. Each contains sub-services — for example, the EntertainmentService class includes

[3]Object property denotes relations between instances of two classes (see owl:ObjectProperty).

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006


```
<device:AudioDevice rdf:ID="TestDevice12">
<device:hasDeviceInfo>
<device:DeviceInfo>
<device:friendlyName>TestDevice12</device:friendlyName>
<device:DeviceUUID>0</device:DeviceUUID>
<device:DeviceType>urn:schemas-upnp-org:TestDevice12:1</device:DeviceType>
<device:DeviceModelURL>http://TestDevice12URL/</device:DeviceModelURL>
<device:DeviceModelNumber rdf:datatype="&xsd;double">0.0</device:DeviceModelNumber>
</device:DeviceInfo>
</device:hasDeviceInfo>
<hw:componentOf>
<hw:RAM rdf:about="#JCTestMemory2"/>
</hw:componentOf>
<serv:hasDCOMPService>
<!-- can have more than 1 service -->
<serv:AudioService rdf:about="#JCAudioService01"/>
</serv:hasDCOMPService>
<!-- 2nd service -->
<serv:hasDCOMPService>
<serv:AudioService rdf:about="#JCAudioService02"/>
</serv:hasDCOMPService>
<!-- 3rd service -->
<serv:hasDCOMPService>
<serv:AudioService rdf:about="#JCAudioService03"/>
</serv:hasDCOMPService>
```

Fig 24.15 Typical display device expression.

AudioService, VideoService, FileRepositoryService, SetTopBoxService and FollowMeService. The LightsAndFittingsService and EntertainmentService are mutually distinct (i.e. in mathematical terms, they do not belong to a same set). These characteristics are modelled by declaring the classes to be disjointWith[4] each other. Every service in the dComp environment is identified by a property called 'serviceID' and a class called 'StateVariable' (to represent UPnP values). The StateVariable class has three properties, namely 'name', 'value' and 'evented'. The relationship between a DCOMPService and the StateVariable is linked by an object property called 'hasStateVariable'. The relationship between a DCOMPDevice and DCOMPService is coupled by an object property called 'hasDCOMPService'.

Community Class

As dComp needs to support the notion of community (a collective), there is a class called DCOMPCommunity. In the current implementation, we model three types of community, namely SoloCommunity (for those devices not yet part of a community), PersistentCommunity (for communities with a degree of permanency), and TransitoryCommunity (for communities with a short lifetime). A DCOMPDevice can join one or more communities (a community must have at least one device). Relationship between a DCOMPDevice and a DCOMPCommunity, is described using an object TransitiveProperty[5] called 'inTheCommunityOf'. A class called 'CommunityDevice' is introduced to represent all the devices in a community. These devices are identified by their deviceUUID identification. The relationship between a Community and CommunityDevice is linked by another object

[4] DisjointWith asserts that the class extensions of the two class descriptions involved have no individuals in common.

[5] TransitiveProperty denotes if a device X is in the community of C and the community C is a member of Community P, then the device X is also a member of community P.

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

TransitiveProperty called 'hasCommunityDevice'. Communities in dComp are formed by a user; thus, each community has an owner. The properties of Communities are — community ID, communityName, communityDescription and timestamp. The relationship between a community and its owner is linked by an object type property, called 'hasOwner'. An example of the main elements in a dComp TV community is given in Fig 24.16.

```
<com:TransitoryCommunity rdf:ID="JCTV">
<com:communityID>Tran-JCTV</com:communityID>
<com:communityName>JC TV</com:communityName>
<com:communityDescription>The first JC testing TV</com:communityDescription>
<com:timeStamp rdf:datatype="&xsd;dateTime">2004-09-06T19:43:08+01:00</com:timeStamp>
<com:hasOwner>
<person:Person rdf:about="#JC"/>
</com:hasOwner>
<com:hasCommunityDevice>
<com:CommunityDevice rdf:about="#JCMonitorCRT17"/>
</com:hasCommunityDevice>
<com:hasCommunityDevice>
<com:CommunityDevice rdf:about="#JCAudioMMS223"/>
</com:hasCommunityDevice>
<com:hasCommunityDevice>
<com:CommunityDevice rdf:about="#JC:NetGem442"/>
</com:hasCommunityDevice>
</com:TransitoryCommunity>
```

Fig 24.16 Typical TV community definition.

Rules Class

Rules are needed in TOP for co-ordinating community actions and are supported by a class called 'Rules' which models three types of rule, namely UnchangeableRules (rules that cannot be changed), PersistentRules (rules that infrequently change), and NonPersistentRules (rules that frequently change). These rules are mutually distinct and are declared to be complementOf[6] each other. Rules have properties — ruleID and ruleDescription and an object property called 'hasRuleOwner' to link to the owner. (Note that the rule and community owners may be different people.) A class called 'Preceding' is used to represent a set of triggers that cause the co-ordinating actions to be executed. The devices in the Preceding class are identified by their deviceUUID, and the service they offer. Finally an object property called 'hasAction' binds the relationship between Rules and Actions. The main elements of a Rule Definition are given in Fig 24.17.

Action, Person, Policy and Time Class

Wherever possible we have sought to build on existing ontology work. SOUPA provides a suitable DCOMPperson, Policy and Time ontology and thus these have been adopted in dComp[7]. The Action ontology document has, to some extent, been influenced by the SOUPA Action ontology. The class 'Action' represents the set of actions defined by the rules. As with SOUPA, we have two class types of action, namely PermittedAction and ForbiddenAction. The Action class in dComp is the

[6] ComplementOf denotes all individuals from the domain of discourse that do not belong to a certain class.

[7] For further information, refer to their Web site [51].

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

union of these two action classes; every co-ordinating action has its target devices. A class called 'Recipient' models target devices, which represents a set of target devices where actions take place. The members of Recipient are identified by their deviceUUID and the serviceID. Actions for the recipient are called 'TargetAction' which has two properties, namely actionName (the name of the action) and targetValue (the value for the action to be taken). A typical statement 'when the phone rings, mute the TV' could be expressed as in Fig 24.18.

Preference Class

As the name implies, DCOMPPreference describes the preferences a person has within any given set of options. In dComp, preferences are referred as 'situated preferences', which is similar to Vastenburg's 'situated profile' concept where he

```
<NonPersistentRules rdf:ID="Rule1">
 <rule:ruleID rdf:datatype="xsd;int">00001</rule:ruleID>
 <rule:ruleDescription>Test Rule 1</rule:ruleDescription>
 <com:communityID>Tran-JCTV</com:communityID>
 <rule:hasRuleOwner>
  <person:Person rdf:about="#JC"/>
 </rule:hasRuleOwner>
<rule:hasPreceding>
 <!-- can have more than 1 device -->
 <rule:Device>
  <dComp:DeviceUUID>uuid:Telephone01</dComp:DeviceUUID>
  <serv:hasDCOMPService>
  <!-- a device can provide more than 1 service-->
  <serv:TelephoneService>
  <serv:serviceID> Telephone</serv:serviceID>
   <serv:hasStateVariable>
       <!-- a service can have more than 1 value of state variable-->
       <serv.name>state variable 1</servname>
       <serv:value>RINGING</SERV:VALUE>
   </serv:hasState Variable>
  </serv:TelephoneService>
  </serv:hasDCOMPService>
 </rule:Device>
</rule:hasPreceding>
<rule:hasAction>
 <act:PermittedAction rdf:about="#TestAction"/>
</rule:hasAction>
</NonPersistentRules>
```

Fig 24.17 Main elements of rule definition.

```
<act:PermittedAction rdf:ID="TestAction">
 <act:actionName>Test action</act:actionName>
 <act:hasRecipient>
<device:DeviceUUID>UUID:PHLAudioMMS223</device:DeviceUUID>
  <serv:serviceID>AudioMMS223</serv:serviceID>
 </act:hasRecipient>
 <act:hasTargetAction>
  <act:actionName>Mute</act:actionName>
  <act:targetValue>Mute</act:targetValue>
 </act:hasTargetAction>
</act:PermittedAction>
```

Fig 24.18 Main elements of an action (muting the TV).

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

uses situation as a framework for user profile so that the values of the profile are relative to situations [52]. The 'Preference' class represents a set of situated preferences of a person for his community. This Preference class has a subclass called 'CommunityPreference' and an associated property called 'communityID'. To model 'person A prefers X, depending on the situation conditions of Y', another class called 'SituatedConditions' is defined which represents the set of situated conditions that the person's preferences depended on. Although a person is allowed to define his own 'SituatedConditions', dComp explicitly defines a list of pre-set situated conditions so that it forms a default template that a person can use. The Preference class has a close relationship to the Person class. To bind this relationship, an object property called 'hasPreference' is used, which links the domain of Person to the range of Preference. The relationship between the Preference class and SituationConditions class is linked by another object property called: 'hasCondition'. The main elements of a Situated Condition are given in Fig 24.19.

```
<owl1:Class rdf:ID="SituationalCondition">
 <rdfs:label>SituationalCondition</rdfs:label>
 </owl:Class>
 <SituationalCondition rdf:ID="DuringTheWorkdays"/>
 <SituationalCondition rdf:ID="DuringThe Weekends"/>
 <SituationalCondition rdf:ID="WhileOutOfTown"/>
 <SituationalCondition rdf:ID="WorkingFromHome"/>
 <SituationalCondition rdf:ID="FriendsVisiting"/>
 <SituationalCondition rdf:ID="FamilyVisiting"/>
 <SituationalCondition rdf:ID="OnHoliday"/>
 <SituationalCondition rdf:ID="WhenComeHomeFromWork"/>
 <SituationalCondition rdf:ID="WhenComeHomeFromSchool"/>
 <SituationalCondition rdf:ID="WhenAtMyOffice"/>
 <SituationalCondition rdf:ID="WhenDining"/>
 <SituationalCondition rdf:ID="WhenHavingLunch"/>
 <SituationalCondition rdf:ID="WhenHavingBreakfast"/>
 <SituationalCondition rdf:ID="WhenEasting"/>
 <SituationalCondition rdf:ID="WhenPlayingComputerGames"/>
 <SituationalCondition rdf:ID="WhenWatchingTV"/>
 <SituationalCondition rdf:ID="AtNight"/>
 <SituationalCondition rdf:ID="InTheMorning"/>
 <SituationalCondition rdf:ID="AtLunchTime"/>
 <SituationalCondition rdf:ID="AtTeaTime"/>
 <SituationalCondition rdf:ID="Alone"/>
 <SituationalCondition rdf:ID="WhenAlarmGoesOff"/>
 <SituationalCondition rdf:ID="WhenSmokeAlarmGoesOff"/>
```

Fig 24.19 Main elements of a situated condition.

24.6.5 dComp Performance

To assess the performance of dComp we compared two sets of device descriptions — the first description was structured in typical XML-based 'all-in-one' format, while the second was decomposed into smaller segments (i.e. broken up into hardware and service information), each segment being 'linked' back to the device. Both descriptions were written in OWL. For each set, we used two different quantities of devices in the test (3 and 32). A common query with five conditions was used for the test, with each test being run fifty times. The test was conducted using a WindowsXP, 2.08 GHz, 512 RAM machine. Four sets of tests were completed:

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

- 3 device descriptions in 'all-in-one' format;
- 3 device descriptions in 'decomposed' format;
- 32 device descriptions in 'all-in-one' format;
- 32 device descriptions in 'decomposed' format.

A representative example of our tests is provided in Fig 24.20. As can be seen we found that the decomposed device description out-performed the compact devices description for smaller domains with fewer devices. On average, queries took half the time that 'all-in-one' format descriptions took. Although we had been concerned that decomposed descriptions might not fair as well for larger domains, because of increased link following, we found that this was not the case, as the system performed as well as the 'all-in-one' descriptions, while bringing the advantages of decomposition described earlier. This we attribute to additional link-processing being counterbalanced by the processing benefits of smaller, better focused descriptions. For larger domains we found that the performance of decomposed versus the compact descriptions remained roughly the same. Both TOP and dComp represent new directions in our research and hold great promise for solving the problems of providing user creativity and privacy.

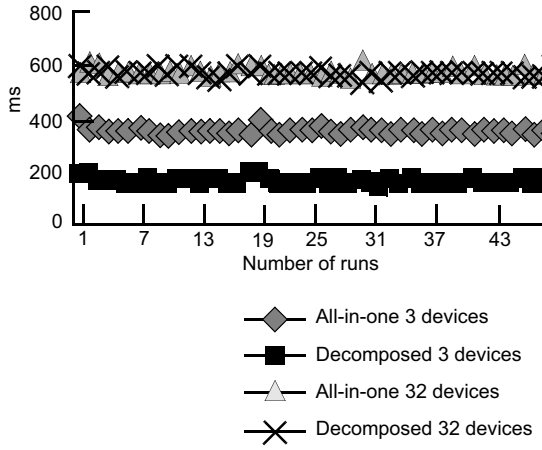

Fig 24.20 A typical dComp performance test.

24.7 Summary and Future Directions

Both the ISL and AOFIS provide life-long learning and adaptation for pervasive devices and communities. Both techniques were evaluated by arranging for users to live in and use the iDorm for periods of up to five days. Both techniques performed well in handling human behaviour (with all the uncertainties involved), and in dealing with complex sensors, actuators and control. The agents operated in a non-

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

intrusive implicit manner allowing the user to continue operating the pervasive computing device or community in a normal way while the agents learn controllers that satisfy the user's required behaviour.

In contrast, the TOP approach deliberately seeks to involve the user in the learning phase, providing explicit control of what and when the agent learns. In support of TOP we devised the ontology dComp which directly supports the concept of community, and collectives of decomposed devices, together with co-ordinating actions to create meta-group functionalities. We were motivated to produce such an ontology to serve our longer term research goals which aim to explore the development of end-user tools to allow lay people to 'program' groups of co-ordinated pervasive computing devices, so as to become designers of their own environments. In dComp, the notion of decomposition extends to device descriptions which are decomposed to map to separate sub-capabilities, each linked to other related descriptions. Thus, decomposed device descriptions do not necessarily have to reside in the same place, a consequence of dComp's roots being in the Semantic Web ontology, which provides firewall protection for the physical location of data servers, facilitating information retrieval via a hyperlink. dComp device descriptions can be shared or reasoned about by other applications on the network, while queries can be directed to a specific service rather than the whole device.

Contrasting these two approaches will allow us to evaluate the arguments for and against increased agent autonomy and determine when and where each is appropriate. In all the approaches, the underlying science is based on methods that are practical to implement in embedded computers.

Our work is taking a number of directions. Firstly, we are continuing to try to develop and experiment with new types of autonomous intelligent embedded agents. For example, we have projects underway looking at new type-2 fuzzy logic-based agents and new types of neuro-fuzzy agents. We are also mindful of the role that mood and emotions play in making decisions and have begun a project that is seeking to enrich the decision space of agents by adding sensed data on emotions. We have also embarked on two projects concerned with investigating developing agents at a nano-scale — one project is looking at nano agents in fluids, the other as part of smart surfaces. Our end-user programming work (TOP and ontologies) is at an early stage but the initial results as reported in this chapter are encouraging. We anticipate future systems will require a mix of both autonomous-agent approaches (perhaps dominating low levels) and end-user programming methods (perhaps dominating higher levels). We hope our work will go some way to resolving where and when either method is most appropriate, perhaps exploring the notion of the end user determining the levels of autonomy that the communities of devices use. Finally, to gather more realistic and meaningful results for all our research into pervasive environments, we need better data and so, with SRIF support, we have embarked on the construction of a new purpose built test-bed for pervasive computing and iSpace work called the iDorm-2. The iDorm-2 is a full-size domestic flat built from scratch to facilitate experimentation with pervasive computing technology. Apart from being equipped with the latest pervasive computing appliances, and having been constructed to facilitate easy experimentation, the major advantage of the iDorm-2 is that we will be able to get much longer periods of experimentation as people will be able to stay in the environment for weeks and months. Thus we look forward to being able to report more interesting and useful results when this facility comes on line during 2005.

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

References

1. Clarke G, Callaghan V and Pounds-Cornish A. Intelligent Habitats and The Future: The Interaction of People, Agents and Environmental Artefacts. 4S/EASST Conference on Technoscience, Citizenship and Culture in the 21st Century, Vienna, September 2000.

2. Colley M et al. Intelligent Inhabited Environments: Co-operative Robotics and Buildings. 32nd International Symposium on Robotics (ISR 2001), Seoul, Korea, April 2001.

3. Chin J S Y and Callaghan V. Embedded-Internet Devices: A Means of Realizing the Pervasive Computing Vision. IADIS International Conference, Algarve, Portugal, November 2003.

4. [Facts 03]

5. [GSM 01]

6. [Metcalfe 01]

7. PHEN — http://iieg.essex.ac.uk/phen

8. TINI — http://www.ibutton.com/TINI/

9. Imsys — http://www.imsys.se/

10. JStik — http://jstik.systronix.com/

11. Mini ITX — http://www.mini-itx.com/

12. Callaghan V, Clarke G and Pounds-Cornish A. Buildings as Intelligent Autonomous Systems: A Model for Integrating Personal and Building Agents. The 6th International Conference on Intelligent Autonomous Systems (IAS-6), Venice, Italy, 2000.

13. [Mozer 98]

14. Callaghan V et al. A Soft-Computing DAI Architecture for Intelligent Buildings. Journal of Studies in Fuzziness and Soft Computing on Soft Computing Agents, Physica-Verlag-Springer, 2001.

15. Callaghan V et al. Embedded Intelligence: Research Issues for Ubiquitous Computing. Proceedings of the Ubiquitous Computing in Domestic Environments Conference, Nottingham, September 2001.

16. Brooks R. Intelligence Without Representation. Artificial Intelligence, 1991:47:139-159.

17. Hagras H et al. A Hierarchical Fuzzy Genetic Agent Architecture for Intelligent Buildings Sensing and Control. RASC 2000 — International Conference on Recent Advances in Soft Computing, Leicester, UK, June 2000.

18. Hagras H et al. A Hierarchical Fuzzy Genetic Multi-Agent Architecture for Intelligent Buildings Learning, Adaptation and Control. International Journal of Information Sciences, August 2001.

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

19. Cayci F, Callaghan V and Clarke G. DIBAL — A Distributed Intelligent Building Agent Language. The 6th International Conference on Information Systems Analysis and Synthesis (ISAS 2000), Orlando, July 2000.

20. Duman H, Hagras H A K and Callaghan V. A Soft-Computing Based Approach to Intelligent Association in an Agent-based Ambient-intelligence Environment. 4th International Conference on Recent Advances in Soft Computing, Nottingham, 2002.

21. Duman H et al. Intelligent Association in Agent-based Ubiquitous Computing Environments. Proceedings of the 2002 International Conference on Control, Automation and System (ICCAS'02), Muju, Korea, October 2002.

22. Hagras H A K et al. Incremental Synchronous Learning for Embedded-Agents Operating in Ubiquitous Computing Environments. In: Soft Computing Agents: A New Perspective for Dynamic Information Systems. IOS Press, 2002.

23. Hagras H A K et al. A Fuzzy Incremental Synchronous Learning Technique for Embedded Agents Learning and Control in Intelligent Inhabited Environments. Proceedings of the 2002 IEEE International Conference on Fuzzy Systems, Hawaii, 2002:139-145.

24. Hagras H A K et al. Online Learning and Adaptation for Intelligent Embedded Agents Operating in Domestic Environments. In: Maravall D and Changjiu Zhou D R (editors). Fusion of Soft Computing and Hard Computing for Autonomous Robotic Systems. Physica-Verlag, 2002.

25. Holmes A, Duman H and Pounds-Cornish A. The iDORM: Gateway to Heterogeneous Networking Environments. International ITEA Workshop on Virtual Home Environments, Paderborn, Germany, February 2002.

26. Pounds-Cornish A and Holmes A. The iDorm — a Practical Deployment of Grid Technology. 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002), Berlin, Germany, May 2002.

27. Hagras H A K et al. A Fuzzy Logic Based Embedded Agent Approach to Ambient Intelligence in Ubiquitous Computing Environments. IEEE Intelligent Systems Journal, 2004.

28. Doctor F, Hagras H and Callaghan V. A Type-2 Fuzzy Embedded Agent for Ubiquitous Computing Environments. In the Proceedings of the IEEE International Conference on Fuzzy Systems, Budapest, Hungary, July 2004.

29. Doctor F, Hagras H and Callaghan V. An Adaptive Fuzzy Learning Mechanism for Intelligent Agents in Ubiquitous Computing Environments. Proceedings of the 2004 World Automation Congress, Seville, Spain, June 2004.

30. Genetic-Fuzzy Controller, UK Patent No 99 10539.7, May 1999.

31. Chin J S Y et al. Pervasive Computing and Urban Development: Issues for the Individual and Society. UN Second World Urban International Conference on The Role of Cities in an Information Age, Barcelona, Spain, September 2004.

32. Smith D C. Pygmalion: A Computer Program to Model and Stimulate Creative Thought. Basel, Stuttgart, Birkhauser Verlag, 1977.

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

33. Design Issues — http://www.w3.org/DesignIssues/Semantic.html

34. Chin J S Y et al. An Ontology Based Approach to Representation and Decomposition In Pervasive Home Computing Environments. Submitted to Pervasive 05.

35. Lieberman H. Your Wish is My Command: Programing by Example. Morgan Kaufmann Press, 2001.

36. Myers B A. Creating User Interfaces Using Programming by Example, Visual Programming, and Constraints. ACM Transactions on Programming Languages and Systems (TOPLAS), April 1990:12:2:143-177.

37. Halbert D C. SmallStar: Programming by Demonstration in the Desktop Metaphor. In: Watch What I DO. MIT Press, 1993.

38. Bring Programming by Demonstration to CAD Users — http://www.lisi.ensma.fr/ihm/index-en.php

39. Stagecast Creator — http://www.stagecast.com/creator.html

40. AgentSheets — http://agentsheets.com/products/index.html

41. ToonTalk — http://www.toontalk.com/

42. Sugiura A and Koseki Y. Internet Scrapbook: Automating Web Browsing Tasks by Demonstration. Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology, San Francisco, California, United States, 1998:9-18.

43. Bauer M et al. Programming by Example: Programming by Demonstration for Information Agents. Communications of the ACM, March 2000:43:3:98-103.

44. McDaniel R. Demonstrating the Hidden Features that Make an Application Work. In: Lieberman H (editor). Your Wish is My Command: Programming by Example. Morgan Kaufmann Press, 2001:163-174.

45. Lieberman H et al. Training Agents to Recognize Text by Example. Proceedings of the Third Annual Conference on Autonomous Agents, Seattle, Washington, United States, 1999:116-122.

46. Blackwell A F and Hague R. Designing a Program Language for Home Automation. Proceedings PP1G, 2001:85-103.

47. Jena — http://jena.sourceforge.net/

48. Haarslev V and Moller R. Description of the RACER system and its application. Proceedings International Workshop on Description Logics, 2001.

49. Zou Y, Chen H and Finin T. F-OWL: An Inference Engine for Semantic Web. Proceedings of the Third NASA-Goddard/IEEE Workshop on Formal Approaches to Agent-Based Systems, April 2004.

50. Network Inference — http://www.networkinference.com/products/construct_business.html

51. SOUPA — http://pervasive.semanticweb.org/soupa-2004-06.html

From Chapter 24 Springer-Verlag book; The Application of Pervasive ICT Series: Computer Communications and Networks, Steventon, Alan; Wright, Steve (Eds.) 2006, XVIII, 438 p. 162 illus., Softcover ISBN: 1-84628-002-8. January 2006

52. [Chin 04b]

53. Vastenburg M. SitMod: A Tool for Modelling and Communicating Situations. Second International Conference, PERVASIVE 2004, Vienna Austria, April 2004.