6 Beyond Systems: Ubiquitous Computing

In this chapter, we discuss how EM principles can potentially be applied to everyday practical computing as it may be in the future, where some systems can never be formalised by developers. The system concept is formed upon its use in a situation. Following Weiser [Wei91] we adopt the term ‘Ubiquitous Computing’ or in short ‘ubicomp’ to refer to an era where people will use a variety of computer-based devices to support everyday life activities. We firstly identify a variety of researches related to ubicomp. We discuss and summarise their shared visions. We argue that these visions are hindered by the lack of a conceptual framework to encapsulate the complexity and new requirements of ubicomp. In particular, little research has so far been conducted to develop a conceptual framework that explicitly supports both design and use of ubicomp devices. We argue that having a coherent conceptual framework is very significant for the conceptual integrity of ubicomp systems, and that this is fundamental to the success of ubicomp. We introduce a new conceptual framework based on EM principles and tools and illustrate this with examples. We shall discuss challenges involved in realising the framework. Finally, we shall describe some related research work and make comparisons with our proposed framework.

6.1 Visions of the future computing environment

Vannevar Bush, in his 1945 article “As We May Think”, envisaged a device that can manage and disseminate results of research [Bus45]. Baecker et al. [Bae95, p35] view Bush as the first person to see beyond the scientific use of the computer to its use as a “fundamental tool for transforming human thought and human creative activity”. Now, nearly 60 years later, the development of computer technologies has led to the realisation of Bush’s dream. A new vision of the role of computers has evolved. Computers are getting more and more pervasive. In his 2000 article “As We May Live”, Gibbs reports on research into ubicomp that applies computer technologies in our everyday life, such as a Georgia Tech’s four-bedroom house where there are more than 60 computers, 25 video cameras and 40 cabinet sensors [Gib00]. The vision has

shifted from mainly concerning the way computers support us to *think* to a broader concern for the way computers can support us in *living* (cf. [Dau00, Gor97]). Devices with computing power are moving off the desktop into everyday items (cf. [Sch00a]).

Since Weiser's seminal paper [Wei91], research interests in ubicomp have grown tremendously. Amongst these, the most representative research titles include ubiquitous computing, invisible computing, disappearing computing, sentient computing and augmented reality computing. We shall briefly review these five research areas. This review will be the basis for our discussion of how EM principles can be applied to ubicomp in the rest of this chapter. All five research areas share prominent common themes, but each has its own distinctive emphasis.

Ubiquitous computing. Ubiquitous computing is also sometimes referred to as *pervasive computing* (e.g. [Ark99, Old99]). The term ubiquitous computing or ubicomp was coined with Weiser and colleagues at Xerox PARC in the late 1980s. Weiser promotes a new way of thinking about computer: "one that takes into account the natural human environment and allows the computer themselves to vanish into the background" [Wei91]. The motivating idea in ubicomp is to make computing power available through the physical environment invisibly. It has been viewed as the Third Wave of computing [Fol02]. The First Wave was many people per computer (mainframe). The Second Wave was one person per computer (personal computer). The Third Ware is characterised by many computers per person. The initial research areas identified by Weiser included new interaction devices, power consumption and wireless connectivity [Wei93]. More recent areas of interest include natural interfaces, context-aware applications, and automated capture and access [Abo00]. The goal of developing natural interfaces is to "support common forms of human expression and leverage more of our implicit actions in the world". Context-aware applications need to sense the environment and adapt the computation according to the use situation. These applications also need to provide facilities for users to capture and access live experiences.

Invisible computing. The concept of invisible computing, introduced by Norman [Nor99], is primarily concerned with how ubicomp technologies can be best integrated into everyday life. The idea of information appliances is central to invisible computing. Norman argues that general-purpose personal computers are difficult to use because they are technology-centred products that are inherently complex. The solution is to develop information appliances that are small, task-focused devices in place of big, complex, general-purpose personal computers. This is to design an

information appliance to fit the task so well that the device "becomes a part of the task, feeling like a natural extension of the work, a natural extension of the person" [Nor99]. One distinctive feature of information appliances is their ability to 'communicate' among themselves and share relevant information. Norman suggests human-centred product development with a cross-disciplinary team of experts in marketing, engineering, and user experience. He promotes Contextual Design that includes six tasks: talk to specific customers while they work, interpret the data in a cross-functional team; consolidate data across multiple customers; invent solutions grounded in user work practice; structure the system to support this new work practice; iterate with customers through paper mock-ups; and design the implementation object model or code structure.

Disappearing computing. Disappearing computing is a European initiative on research and development of future computing. Its mission is "to see how information technology can be diffused into everyday objects and settings, and to see how this can lead to new ways of supporting and enhancing people's lives that go above and beyond what is possible with the computer today" [Dis01]. Though the overall goal of disappearing computing is similar to that of other ubicomp research, its specific research strategy involves three sub-goals of particular interest in connection with this thesis. These are:

- creating artefacts that have the attributes of openness and connectivity;
- promoting emerging functionality through the collaboration of collections of artefacts;
- designing artefacts with the emphasis on people's experience of them.

Sentient computing. Sentient computing is a collaborative project between the AT&T Laboratories and University of Cambridge [Sen02]. Its emphasis is on developing and exploiting technologies to give computers access to the state of their environment. The project started from the development of an ultra-sonic indoor location system. The system can provide the locations of tagged objects or people to an accuracy of about 3cm throughout a 10000 square foot building. The distinguishing feature of sentient computing is its use of sensors and resource status data to maintain a model of the real world which is shared between users and applications. One representative application enables a networked scanner to perform a selection from a list of functions presented in the form of a poster. A user can use a tagged object to point at one of the functions on the poster. This in turn triggers the scanner to perform the function. In this case, the system maintains a model of the real

world that incorporates the geometric relationship between the poster and the tagged object. This model is a communication medium between the scanner application and its user. In effect, “the whole world is a user interface” [Sen01]. The goal of sentient computing is to make applications more responsive and useful by observing and reacting to the physical world [Hop99]. Research is based on three major themes: developing sensor technology, experimenting with application devices, and constructing platforms that connect sensors and devices together.

Augmented reality. Research on augmented reality aims to superimpose virtual objects upon, or compose virtual objects with, the real world. One way of augmenting reality is to overlay computer-generated graphics onto the real world. But augmented reality is not limited to sight – it might be applied to all senses. The motivation for augmented reality is to “enhance a user’s perception of and interaction with the real world. The virtual objects display information that the user cannot directly detect with his own sense” [Azu97]. A typical augmented reality system consists of three components: a head-mounted display, a tracking system, and a wearable computer [Bon02]. The head-mounted display allows us to see text and graphics generated by computers. The tracking system senses the location of a user’s head and eyes, and maintains the correct relationship between virtual objects and real world surroundings with reference to the user’s movement. The wearable computer provides portable, hands-free computational power to drive the whole system [Nap97]. Applications of augmented reality include medical visualisation, maintenance and repair, annotation, robot path planning, entertainment, and military aircraft navigation and targeting [Azu97].

Because of its potentially radical impact on everyday life, research on ubicomp has attracted many critics. A major common concern in critiques of ubicomp is that:

- ubicomp is driven by technology
- insufficient account is being taken of the human perspective on what is desirable in personal and social terms.

We are acknowledging that ubicomp is not necessarily a good thing in every respect. However, EM is offering an approach that promotes high levels of human engagement in the design and use of technology. This can make it easier for designers and users to develop ubicomp applications in a sensitive way.

6.2 Assessing the visions

Although the future development of ubicomp is difficult to predict, the approaches reviewed in the last section do reflect the same dominant emphasis in respect of four key issues: the roles to be played by *automation*, *visibility*, *connectivity* and *adaptation*. All the approaches discussed above aspire to full automation, hiding the technology from the users, indiscriminate interconnection of devices and systems that are self-adaptive. This thesis emphasises a complementary perspective: the need to keep humans in the loop; to encourage user engagement; to promote understanding and control over the interactions amongst devices; and to allow user customisation of the ubicomp environment.

6.2.1 Automation

One of the common but inadequate visions of ubicomp is having ubicomp systems that require nearly no human intervention. Negroponte advocates the use of "intelligent agents" as digital butlers that do all the work for you while you take it easy [Neg96]. Joseph describes this is as "the top of the IT agenda" [Jos02]. Tennenhouse, a vice president in the Intel Corp, advocates "getting the human out of the interactive loop" [Ten00]. This vision is only an industrial hype. Full automation is not plausible for ubicomp. The main reason is that the ubicomp environment is the environment we live in – where activities are situated and exceptions are the norm. Since we cannot prescribe the ubicomp environment, human intelligence has to be involved in solving ubicomp problems. Even the authors with visions for full automation seem to agree that there is a need for the involvement of intelligence. Joseph [Jos02] envisages that "computers will be intelligent enough to manage, configure, tune, repair, and adjust themselves to varying circumstances to handle the workload exposed to them efficiently". Tennenhouse [Ten00] wants to automate the software creation process in ubicomp by generating software from specifications and constraints. Such visions presume that we can automate the management and specification tasks that seem to require human intelligence.

Undeniably, many people dream of sitting back and relaxing and allowing machine servants to help them to do all their work. This dream has become one of the driving motivations of ubicomp development. However, we cannot desire automation blindly for every device and aspect of ubicomp. Automation introduces problems of predictability and accountability. Edwards [Edw01] asks: "how will the

occupant-users adapt to the idea that their home has suddenly reached a level of complexity at which it becomes unpredictable?" To paraphrase Langheinrich et al. [Lan02], "in order to lower the demands on human intervention in … a dynamic world … [we require] the concept of *delegation of control*, where we put automated processes in control of otherwise boring routines, yet provide *accountability mechanisms* that allow us to understand complicated control flows".

In the ubicomp environment, creating and supervising the automatic processes should be part of the users' role. We need to have *the human in the loop* and aim not to replace but to enhance and complement human abilities.

6.2.2 Visibility

Most of the visions for a ubicomp environment explicitly mention that computers should be invisible in the future (e.g. [Wei91, Nor99, Dis02]). The idea is that if we could somehow make computers vanish into the background, the complexity and frustration of using computers nowadays would disappear. In Norman's terms, to hide a technology is to hide the infrastructure of it [Nor99]. He envisages a world where information appliances with infrastructure hidden in the background largely replace conventional personal computers. This view has invited some criticism. Odlyzko [Odl99] believes that "[information appliances] will not lessen the perception of an exasperating electronic environment. The interaction of the coffee pot, the car, the smart fridge, and the networked camera will create a new layer of complexity", in which it creates new frustration. Langheinrich et al. warn that invisibility may lead to unpredictability: "… the ideal of the invisible, altogether unostentatious computer that silently hides in the background, might complicate or even impede the predictability of the system" [Lan02].

Visibility poses a dilemma. On one hand, it is desirable to hide the infrastructure of computer technology from its users, because that might just make the system easier to use and comprehend. A common view is that most users seem to have no interest in how a technology works so long as it does work. On the other hand, when things go wrong, as they often will in the case of computer technology, hidden infrastructure might hinder the possibility of fixing the problem promptly and safely.

In fact, sometimes the idea that "infrastructure should be invisible" is the fundamental cause of frustrations. For example, the latest versions of the Microsoft Windows operating system (e.g. Windows XP) hide file extensions from the user by

default. There is an increasing number of people who do not know what file extensions are for, which might be a good thing because often it is the applications that mostly care about them. However, some of the file extensions can be shared by more than one application. For example, a ".txt" file can be used by Notepad, Wordpad, and MS Word. Frustration arises when a user wants to open a ".txt" file using MS Word – on double-clicking the ".txt" file, Notepad pops up every time but not MS Word! The problem then becomes: what should be visible and what should not? Unfortunately, the answer will depend on the individual user and situation.

Hjelm sees some analogy between the development of radio and development of computer technology [Hje01]. The development of radio went through three design phases: the archaic, the suppressed and the utopian. In the archaic phase, the radio was a new invention that was intrusive in a home environment and required an expert to use it. In the suppressed phase (because the product was not widely accepted), commercial applications that involved hiding the unfamiliar radio in big bulky but familiar objects such as grandfather clocks were explored. In the utopian phase, the radio was transformed into the compact, usable, and portable forms now in wide use. The development of computer technology might now be viewed as entering the suppressed phase, where people are embedding computers into every imaginable everyday object.

Streitz [Sto01] argues that causing the computer to disappear is only the first step towards achieving the final goal of "coherent experiences". He adds "… coherent experience is the result of the combination of macro affordances (e.g. physical shape and form factor) and certain micro affordances (e.g. tactile characteristics of the artefact's interface) in combination with the software providing appropriate interaction affordances". We believe that this shift of emphasis to coherent experience is very important to the development of ubicomp. After all, it is *users' engagement* with the ubicomp environment that governs its success. To appreciate the true meaning of invisibility we should ask questions about users' engagement in addition to the more commonly asked questions about ways to hide infrastructure. Therefore, on the basis that exposing and understanding a technology is the first step towards making it conceptually invisible, it might be good for users to know and understand more about the infrastructure of the ubicomp environment. This thinking has an important implication: it leads us to place a conscious emphasis on the design of infrastructures with conceptual integrity that the user can understand easily.

6.2.3 Connectivity

High connectivity is another feature of ubicomp. Norman describes “a distinguishing feature of information appliances is the ability to share information among themselves” [Nor99]. In ubicomp, each person is surrounded by hundreds of wirelessly interconnected computers [Wei93]. Through connectivity, a collection of artefacts can act together and produce “new behaviour and new functionality” [Dis02]. Current technology is certainly capable of making this vision come true. Technologies like Bluetooth [Blu02], a short-range, low-power radio frequency technology, already promise to standardise wireless communications.

Connectivity has become part of the common language of ubicomp – so common that people often forget to justify or even think about reasons for the connections. What “new behaviour and new functionality” that connectivity supports is yet to be discovered. In fact, sometimes connecting everything to everything else is not a good thing. Connectivity can cause new complexity and frustration for ubicomp [Old99]. In [Luc99], Lucky amplifies on the potential frustrations: “My refrigerator… would refuse to open at certain hours of the day, having talked to my bathroom scales”; “My car is no longer the friend I once knew. If I exceed the speed limit, it reports me, and if I try to park illegally, it refuses to turn off or to let me open the door”.

In this context, the key issue for the user is knowing the purpose of the connections, and being able to *understand and control* them at will at any time. Edwards et al. [Edw01] point out that we need new models of connectivity for users to control, use, and debug the devices that are interacting with one another in the environment. Questions like “How can I tell how my devices are interacting? What are my devices interacting with, and how do they choose?” [Edw01] should be easy to answer in the future ubicomp environment.

6.2.4 Adaptation

In the ubicomp environment, requirements are unsettled. Users’ needs change over time, so that a ubicomp system should facilitate dynamic adaptation to various situations. Research on ubicomp usually associates adaptation with context-awareness of applications (e.g. [Abo00, Dey01, Lae01]). Abowd et al. [Abo00] point out that “ubicomp applications need to be context-aware, adapting their behaviour based on information sensed from the physical and computational environment”. The definitions of the term ‘context’ given in the literature vary but a

generic definition can be found in [Dey01a]:

> *"Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."*

Because of the aspiration to develop fully automated systems discussed earlier, the typical research focus is on capturing context by using sensor technologies. Capturing context through sensors only works when we can define the set of possible contexts needed in advance. However, in a ubicomp environment what is relevant to the interaction between a user and an application cannot be fully specified, and usually relevance changes over time. Sometimes, what is relevant is very subjective to the user. Inferences made through the preset sensors may not be accurate. As Edwards et al. [Edw01] observe "… [simple sensing] may report that I am present in a room when, instead, I have simply left my active badge on the desk".

A complementary way of trying to fulfil the adaptation requirement of ubicomp is through 'user modelling'. Traditionally, user modelling is about constructing an explicit profile of properties and preferences of the user in the system. The profile is used as the basis for adaptation and personalisation of the system. There are two approaches to user modelling: *adaptive* and *adaptable* [Fis00, Kul00]. In the adaptive approach, the system dynamically adapts itself to the current task and the current user. In the adaptable approach, the system gives substantial support to allow the user to change the functionality of the system.

Systems like GUIDE (a tourist guide system [Che01]) and PDS (a personal daily system [Byu01]) use both context-aware and user modelling approaches. Combining user modelling with context-awareness in an application improves the system's adaptation capability to some extent. However, there is no way for a system to take full account of all the preferences of the user by just maintaining an explicit representation of the properties of the user. In fact, even the user might not know his or her preferences in respect of a system. Consider one of the scenarios described in [Byu01] for PDS is "When a user passes by a theatre, the PDS can notify the user that the theatre is playing the user's favourite movie." In this case, the location together with a film preference of the user triggers the notification. The question is: who is to specify this behaviour of the system? If it is to be the system, we have the issue of properly predicting the user's state of mind; if it is to be the user, we have the issue of

adequately supporting the user's need to specify the behaviour. Whichever is the answer, however, it is clear that context-awareness and user model maintenance alone are not sufficient for system adaptation. Even in this simple situation, we need mechanisms to *customise* the system to suit individual needs. The system should be able to understand the user's preference and have some reflective capability concerning the way in which these preferences are expressed within itself. The user should likewise be able to understand his or her preferences and relevant functionality of the system. Evolvability of the system comes from mutual understanding, which in turn comes from openness to interaction and customisation.

Figure 6.1 summarises the ideas of this section.

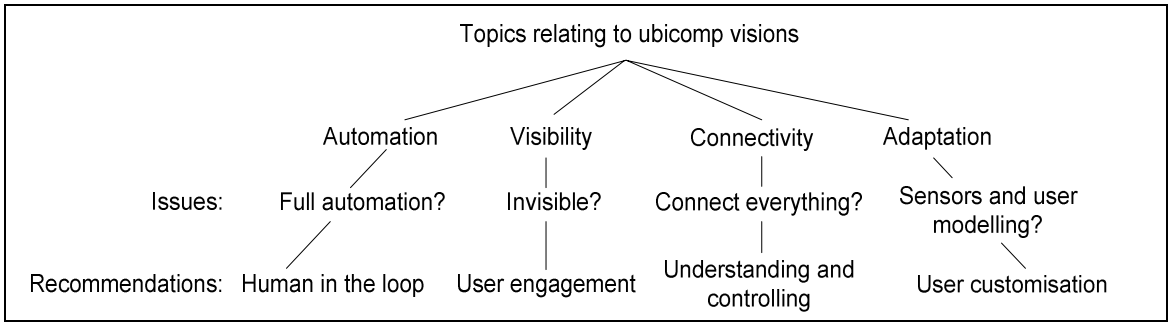

Figure 6.1: Topics, issues and recommendations relating to common ubicomp visions

6.3 A new conceptual framework (SICOD)

In this section, we shall discuss the potential application of EM principles to ubicomp and propose a new conceptual framework in which the issues mentioned in the last section can be more effectively addressed. The potential contribution of EM research to ubicomp becomes apparent when we consider the properties of a ubicomp environment. These include:

- The concurrent nature of a ubicomp environment – EM principles are based on a commonsense way of construing phenomena (chapter 2).
- The importance of context – In contrast to classical approaches to programming, EM gives prominent emphasis to modelling state and situation (chapter 2). Treating contexts as states is identified as a key issue in the ubicomp literature (e.g. [Dey01a, Rah01]).
- Unforeseeable user requirements – EM principles can be used as a heuristic way towards human problem solving (chapter 4).

- Dynamic and unpredictable integration of different devices – EM principles assist integration (cf. [Bey00a]) and help to maintain conceptual integrity (chapter 5).

The EM conceptual framework for ubicomp can be described as a framework for "controlling devices through building EM models". We call these special-purpose models Interactive Control Models (ICMs). In contrast to other conceptual frameworks (which will be described in section 6.5 below), the EM conceptual framework aims to:

- make the infrastructure of ubicomp more *visible* to the users
- provide principles to help users to maintain the conceptual integrity of their views of ubicomp systems.

We shall describe the EM conceptual framework – to be called 'soft interfaces for the control of devices (SICOD)' – in detail, and illustrate its potential use with a ubicomp example. An ICM typically consists of a set of Interactive Device Models (IDMs) and an Interactive Situation Model (ISM). The use of EM principles in the construction of IDMs has been discussed in some detail in previous papers (e.g. [Bey01b, chapter 5 in Run02]). The development and use of ISMs has been a common theme in recent EM research (cf. [Sun99a, Bey99, Bey00c, Bey01b]). In developing ISMs for a ubicomp environment, we propose novel methods for constructing ISMs from IDMs that are aimed at the end-user.

The left of Figure 6.2 shows an ICM with three IDMs (depicted by pentagons) linked to an ISM (depicted by a circle) by dependencies. The whole ubicomp environment can contain a network of ICMs (see the right of Figure 6.2). Notice that an IDM can be shared by more than one ISM. This reflects the fact that many devices are shared resources in a ubicomp environment.

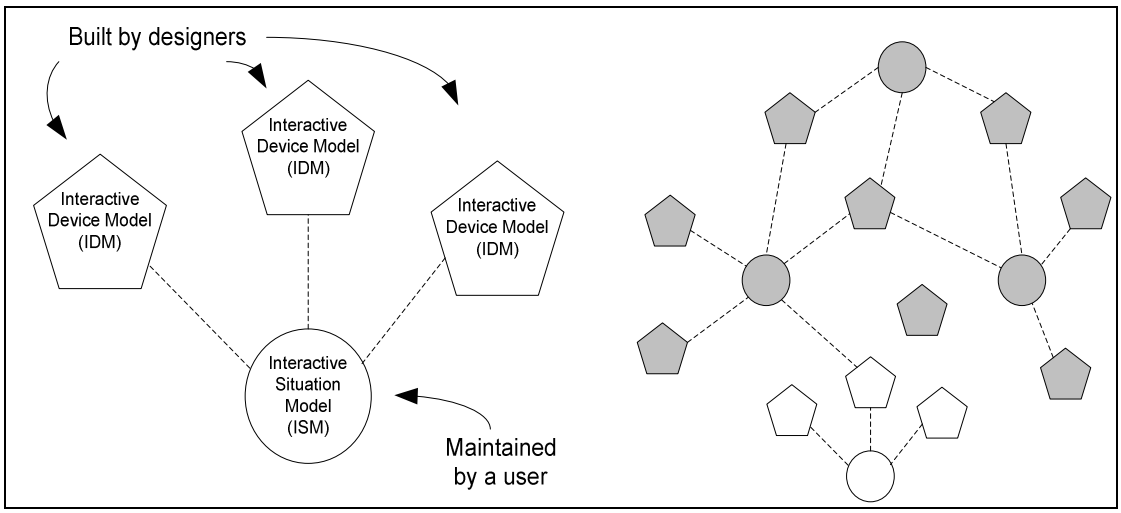

Figure 6.2: An Interactive Control Model (left) and ubicomp environment with many interacting ICMs (right)

Both IDMs and ISMs are built using EM principles but the differences between them are summarised as follows:

IDM	**ISM**
Built by designers.	Built and maintained by users.
Corresponds to a particular device for generic application.	Corresponds to a particular situation for individual use.
Assists users to gain a conceptual understanding of the device.	Assists users to configure devices through creating definitions to establish dependencies between the states of IDMs and users' situated observation.

IDMs are relatively stable models which are built by the device designers. Each device has an IDM. ISMs are models created by users of devices. An ISM links a set of IDMs through dependencies.

To illustrate our conceptual framework, we consider a ubicomp scenario similar to that introduced by Huang et al. in [Hua99]. The scenario is as follows:

> *A user sets up a model to control the stock of a particular drink in her fridge, in this case, canned cola. Four computer-based devices are involved: a fridge, a personal Global Positioning System (GPS), a retail store information device, and a clock. The fridge maintains a count of how many cans of cola there are in it (possibly through some*

kind of object tagging and detection technology); the personal GPS gives the current location of the user; the store information device gives information about the location and opening hours (possibly through a web enabled device connected to the store's homepage); the clock gives the current time and date. The user is holding a party on Friday and has to make sure that there is enough cola in the fridge. She wants the system to remind her to buy the drinks when she is near the store. If she does not go near the store before Friday, the system will remind her to buy the drinks on Friday.

One possible realisation of this scenario within the SICOD framework is depicted in Figure 6.3a. The definitions involved are displayed in Figure 6.3b. These two Figures are complementary representations. Figure 6.3a depicts the interface for the end-user to specify the definitions in Figure 6.3b. A prototype implementation of such an interface will be described in chapter 8.

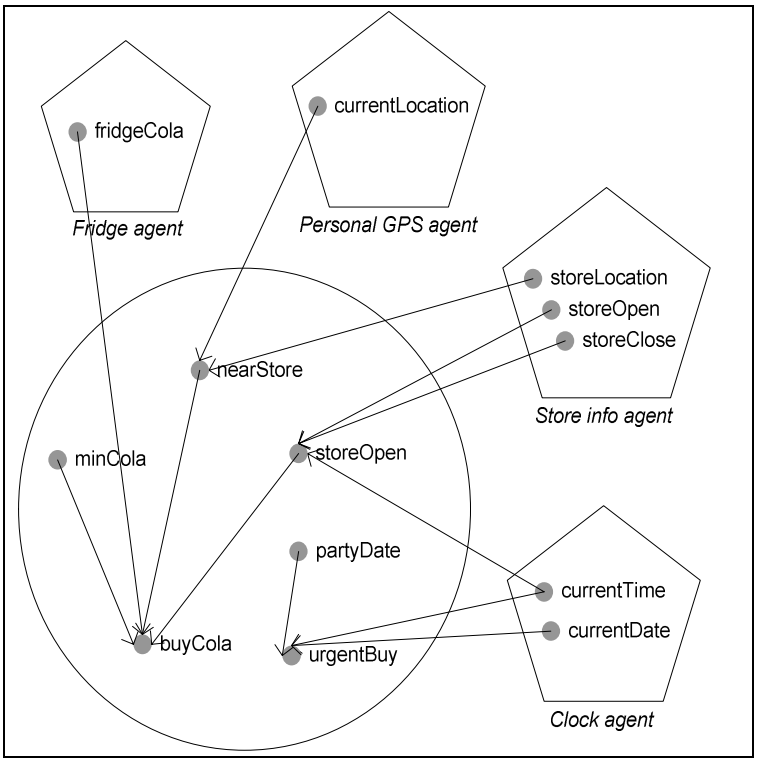

Figure 6.3a: An ICM of a particular drink stock control

IDMs
storeOpen is 8
storeClose is 20
storeLocation is 1234
currentTime is 10
currentDate is 20020815
currentLocation is 1234
fridgeCola is 2

ISM
partyDate is 20020820
minCola is 10
storeOpen is storeOpen < currentTime < storeClose
nearStore is storeLocation = currentLocation
buyCola is minCola > fridgeCola and storeOpen = True and nearStore = True
urgentBuy is minCola > fridgeCola and partyDate = currentDate

Figure 6.3b: Definitions of ICM of the drink stock control

The 13 definitions in this ICM can be divided into two groups: fixed and changeable definitions[3]. Fixed definitions are definitions provided by IDMs, for example, the user cannot change the definition of `storeOpen` (store opening time) because this definition is determined by the store. In this case, each IDM corresponds to an agent. The changeable definitions are the definitions in the ISM that express the user's special requirements of the drink stock control system. The meanings of the definitions are quite obvious when we look at the definitions in detail in the Figure 6.3b. Without going into detail about each definition, we draw attention to two important definitions: `buyCola` and `urgentBuy`. The `buyCola` definition specifies that if the stock of cola in the fridge is below the minimum amount and the user is near the store within store open hours, the system can remind her to buy the drinks. The `urgentBuy` is especially for the party schedule on Friday – so that if there is not enough cola in the fridge on the day of the party, the system will issue a warning.

Note that all the definitions contained in the IDMs in Figure 6.3a are fixed and so for reference only. These IDMs can be thought of as models of sensors linked to the real world. However, an IDM can also contain definitions for actuators that act on the real world. One example is depicted in Figure 6.3c below. Figure 6.3c extends Figure 6.3a by adding an alarm device. This device's IDM contains only one definition, `alarmOn`, where the right-hand-side of the definition can be changed by the user. The simple behaviour of the alarm device is that it generates a tone whenever the

[3] Note that in an ICM variable assignments are represented as constant definitions. For example, we write "fridgeCola is 2" instead of "fridgeCola = 2".

value of `alarmOn` is set to be `True`. We can define `alarmOn` as `buyCola` to specify that the system should remind the user to buy the drinks by generating a tone. This example shows how a device can be configured through its IDM by redefining an observable in the ISM.

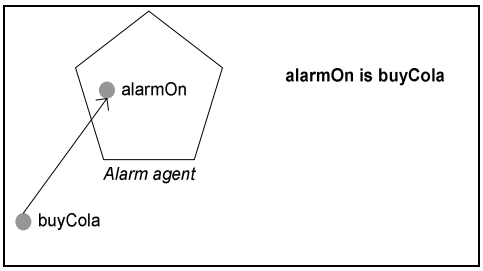

Figure 6.3c: An alarm device extension

By looking at the above application, we can identify the following advantages of the SICOD framework:

- An IDM assists the understanding of how a device works by representing the characteristic and persistent dependencies between its observables. This allows more effective communication of the designer's conceptual model of how a device works to the user.

- ICMs help users to maintain conceptual integrity of the system. For example, it is easy to find the answers to 'why' questions – cause and effect is clear and accessible through navigation of the dependency graph of an ICM (e.g. if the value of `buyCola` is `True`, the user can investigate why this is so by following the dependency links). Since conceptual integrity is a subjective matter, it is doubtful whether the designer of the system can effectively prescribe a view with conceptual integrity for the user. For this reason, we instead need a conceptual framework that helps the user to maintain conceptual integrity. Unlike a traditional window-based GUI, the SICOD framework enables users to have a global conception of system state.

- Uses of the system are highly customisable. In a typical ubicomp environment, there are no fixed boundaries for the system. The system emerges when we build an ICM to link devices together. In fact, the system is created by the user and therefore, high flexibility is guaranteed.

- The sharing of devices is mediated naturally by the framework. The values of the

observables in an IDM can be used by many users concurrently. Particular definitions in an IDM may be changeable and may also be restricted to reflect the fact that they can be changed by only one user at a time. For example, this applies to the definition of `alarmOn` in Figure 6.3c.

- Contexts in the ubicomp environment are explicitly represented as states within the SICOD framework. This is in line with the practice of representing contexts as states to be found in much ubicomp literature. For examples, Dey writes that "… a collection of states can be described as a situation." [Dey01a]; Rahlff et al. defines personal context as "a snapshot of the state of the most important situational parameters: personal identification, time, location, task at hand, nearby objects, nearby people, etc." [Rah01]. Furthermore, by using definitions, we can explicitly specify the relationship between states.

- The framework supports rapid prototyping of a particular use situation. As new requirements come from the users, the best people to prototype the system are the users themselves through the building of ISMs. The designer's responsibilities are to build the functions to support the definitive notation and the sensing technology to support the automatic update of observables.

- The user can use an ISM for a system to adapt the reliable behaviours of IDMs so that they reflect the current situation. For instance, in the context of the drink stock control, a user can configure the ISM so that the notification that the supply of cola is exhausted is suspended during the night.

Apart from the advantages listed above, the SICOD framework also addresses the four important issues discussed in connection with visions for ubicomp in the last section. We shall now discuss these in turn.

6.3.1 Human in the loop

As we discussed in section 6.2.1, full automation is over-hyped in the visions of ubicomp and we need to put the human in the loop. This point can be illustrated by considering the difficulties in obtaining contextual states automatically. The states of digital devices are the easiest to obtain automatically. States of the environment, such as room temperature, illumination intensity and noise level are harder to obtain because they depend on sensor technology. What can be reliably sensed by current technology is limited to very primitive contextual state. States such as the orientation

of an arbitrary physical object, the meaning of a sign and the topic of a conversation are very difficult for devices to detect automatically. It is doubtful whether all such states, which involve human perception and interpretation, can be reliably sensed by computers automatically. The ICM of the cola stock control scenario is a good example – most of the behaviour of the system is determined by the ISM, not by the IDMs.

Most activities in our everyday life are situated. Situated activities mostly contain actions that involve conscious reference to the context and the choice of course of action [Sun99a]. For this reason, ISMs in the SICOD framework play an important role in capturing context-awareness. An ISM maintained by the user of the system represents a particular use case of a system of devices that cannot be prescribed and therefore, the ISM itself cannot be automatically built by the system (cf. circumscription of use cases in UML).

The SICOD framework supports problem-solving in a ubicomp environment that is based on *intelligence captured through practical experience* – a precept that human agents tacitly use to solve problems encountered in the real world [Bey94a, Sun99a]. It supports experimentation, discovery and exploration of an environment prior to the identification of desired reliable behaviour. Automation comes later when patterns of interaction become reliable and a system emerges. However, familiar and reliable patterns cannot take account of dynamical changes in the ubicomp environment. Humans should be involved in constant revising of the ICM to adapt to new requirements and new contexts.

6.3.2 User engagement

We argued in section 6.2.2 that true invisibility comes not hiding infrastructure but from the user's engagement in the primary activities of interest. Sometimes, it is appropriate to make infrastructure visible to the user.

The SICOD framework provides a direct way of modelling the observation of the system and its environment by the user. In particular, just like cells in a spreadsheet, observables associated with an ICM are all task-oriented (the term used by Nardi in [Nar93]) – they represent states that can be perceived and observed by the user; they represent entities that are of interest to the user for the particular task or situation. The user can engage with the task at hand more easily using a spreadsheet

than another conventional programming paradigm [Nar93]. An ICM inherits this advantage of the spreadsheet.

The modelling of observation provided by the SICOD framework is very useful in a ubicomp environment. Since the designer cannot prescribe all the observables in a user's mind in advance, it is left to the user to describe the observables using an ISM. After a period of time, the management of some of these observables might follow such commonplace patterns that the designer can prescribe them in systems for other users.

In the SICOD framework, states are represented explicitly, and interactions are direct. This allows users to define what is to be observed, and allows users to engage in setting up the devices based on their subjective experience (cf. the computer as instrument discussion in [Bey01a]).

6.3.3 Understanding and controlling the connectivity

Where connectivity is concerned, we have emphasised user understanding and control of the interconnections between devices. In the SICOD framework, this is addressed by the notions of agency and dependency.

Within the framework, what to connect to what is entirely up to the user. With dependency graphs, a user is able to get a visual understanding of the interconnections. This helps the user to maintain a clear conceptual model of the communications and devices involved. The explicit representation of dependency helps to make the system traceable, so that, for instance, communication between two devices exists only if there is a dependency link. This helps to ensure that the user can tell why and how the devices are interacting each other.

The notion of agency supports a user's natural commonsense attribution of state change. For example, with reference to the cola stock scenario, a user can refer to the store location definition but cannot change the definition; the fridge agent is responsible for counting the colas. This is a natural application of LSD analysis introduced in chapter 2.

6.3.4 User customisation

We argued in section 6.2.4 that conventional context-awareness and user modelling

techniques are not sufficient to meet the system adaptation requirement for a ubicomp environment. We also need ways for the user to customise the system. Ideally, these should be flexible and easy to use. Usually there are trade-offs between flexibility and ease of use so that if a way is flexible it is usually not easy to use and *vice versa* (cf. [Odl99]). However, the SICOD framework can arguably provide ways to customise the system that are both flexible and easy to use.

To illustrate this point consider a central heating controller, called Balmoral, based on a real-life model described by Green [Gre99]. Figure 6.4 shows the control panel and a summary of instructions on how to use the controls.

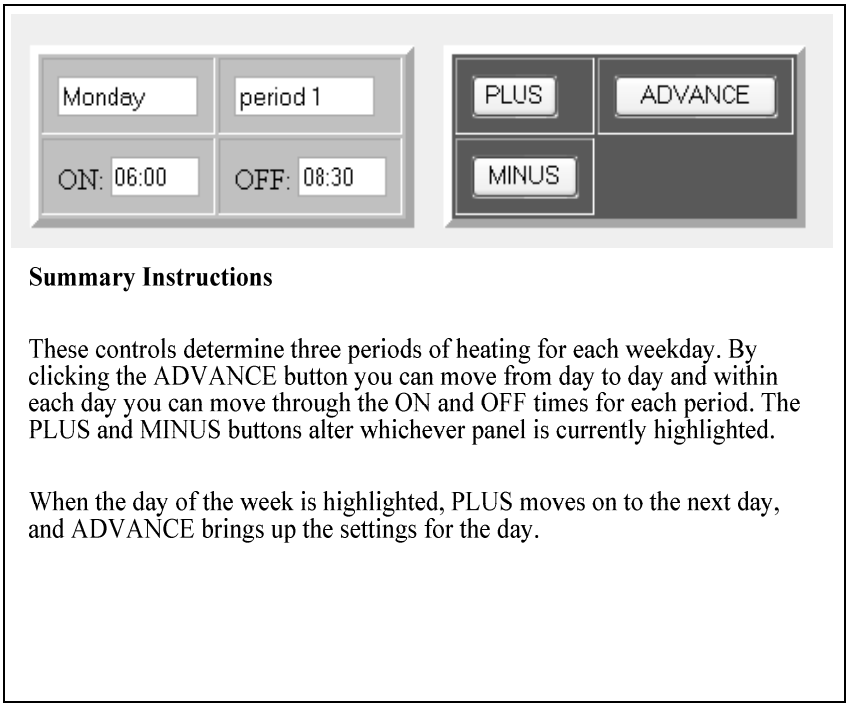

Figure 6.4: A layout of a central heating control panel and its instructions adapted from [Gre99]. On this control panel, there is an LCD on the left and there are buttons on the right.

By pressing buttons on this control panel, a user can set up three periods of heating for each weekday. The operation of the control panel is highly dependent on the mode switching button called 'ADVANCE'. Mode switching buttons like this also exist in most programmable VCRs, washing machines, digital watches and desk clocks. They are also popular in conventional windows-based GUIs. Mode switching makes a system difficult to comprehend because it demands users to switch the perception of the system accordingly. With a conventional 'press-button' interface, however, it is usually unavoidable because of the physical constraints on the number of buttons that we can put on a control panel. For example, it would be inconvenient to

have a separate pair of 'PLUS' and 'MINUS' buttons for every weekday and every heating period. We shall refer to this kind of interface as a 'hard interface'.

In designing a hard interface, we not only have to prescribe the functionality of the system but also the possible observables and their interpretation by the user in the situations of use. This is a potential barrier to providing a system view that has conceptual integrity for the user (e.g. consider the different roles that the 'PLUS' and 'MINUS' play according to the current mode of operation). It also affects the flexibility of the resulting system. For example, the central heating interface only allows the user to enter 3 heating periods for every weekday – a rather arbitrary prescription imposed by the designers.

The SICOD framework provides a different way to configure the system. We can regard an ICM as a 'soft interface' to a system of ubicomp devices. Applying the SICOD framework, an ICM of central heating control would be like the one shown in the Figure 6.5a. The corresponding definitions are shown in the Figure 6.5b.

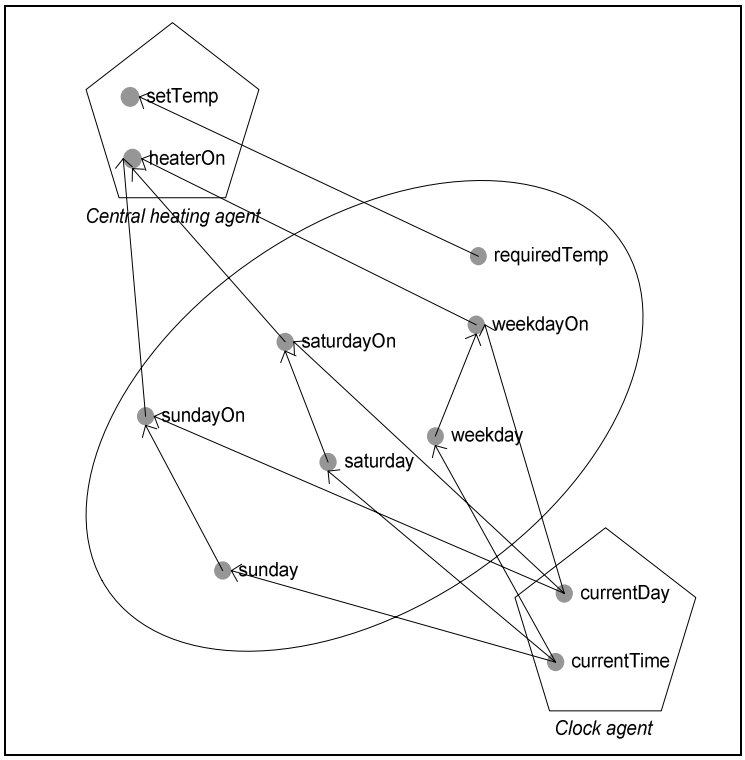

Figure 6.5a: An ICM for central heating control

IDM
setTemp is requiredTemp
heaterOn is sundayOn or saturdayOn or weekdayOn
currentDay is 4
currentTime is 10

ISM
requiredTemp is 20
sunday is 9 < currentTime < 22
saturday is 9 < currentTime < 12 or 16 < currentTime < 22
weekday is 7 < currentTime < 8 or 6 < currentTime < 22
sundayOn is sunday and currentDay = 7
saturdayOn is saturday and currentDay = 6
weekdayOn is weekday and 1 <= currentDay <= 5

Figure 6.5b: Definitions of ICM of the central heating control

The observables sunday, saturday and weekday specify the user's configuration of the heating period. The observables `sundayOn`, `saturdayOn` and `weekdayOn` evaluate to `True` only if they can match their own day and heating period with the current time and day information provided by the `Clock` agent. The central heating will be turned on only if `heaterOn` evaluates to `True`.

With the ICM, extension of the system becomes very easy. For example, suppose that the user has bought sensors to detect if there is any person in the house. The central heating ICM can make use of this information to allow more efficient use of central heating – the user can change the `heaterOn` definition to:

```
heaterOn is (sundayOn or saturdayOn or weekdayOn) and houseNotEmpty
```

A conventional central heating interface will surely have difficulty in embracing this requirement without replacing the whole control panel and developing mechanisms to link to the sensors.

The SICOD framework improves both flexibility and ease of use of the resulting system:

- the system becomes more flexible as its ICM is open to change and extension;

.

- the system becomes easier to use as the SICOD framework provides a simple interface through which to customise devices. The user needs only to learn the underlying definitive notation to be able to control a variety of

different devices.

The idea of allowing the user to customise most parts of the ubicomp system is an alternative approach to the adaptation through auto-learning described in [Byu01]. Byun et al. propose several scenarios in which a ubicomp system can learn patterns of use. One of them is “If a user participates in a meeting at 10 am every fourth Monday, the PDS [their proposed system] might learn this behaviour and suggest that the user might have to go to a meeting today (the fourth Monday) at 10 am. However, a more sophisticated level of learning would enable the system to realize when such a notification is inappropriate, for example when the user is on holiday.” [Scenario D in Byu01]. Simple problems of user customisation become complex problems of artificial intelligence. Even if we do eventually have systems which are smart enough to act on behalf of users, system predictability will become an issue.

In discussing possible extension to Context Toolkit (see [Dey01a] and section 6.5), Dey describes the struggle involved in negotiating the tradeoff between supporting a complex situation and providing a simple method for describing a situation. He adds that “while designers who have domain-specific expertise can determine part of the solution [to a ubicomp problem], they will obviously not think of everything that is needed to support individual users. It is the end user who is in the best position to further specialize context-aware application to meet their individual needs.” [Dey01a]. The application of the SICOD framework is potentially a simple way to allow users to represent situations by building ISMs.

The SICOD framework transforms the role of the designer from ‘prescribing use situations’ to ‘developing reusable functionalities that allow users to specify use situations by themselves’. An analogy can be made here with the spreadsheet framework, in which designers provide a library of domain-specific functions but the actual use of these functions is for spreadsheet users with their specific tasks to determine.

With the SICOD framework, users can develop their understanding of use situations specific to them. This understanding can be animated and visualised by building ICMs. With current technologies, we can envisage users making use of Personal Data Assistants (PDAs) to build and maintain ICMs anytime and anywhere. The new tool introduced in Chapter 8, for example, can be used for this purpose. The result is a ubicomp system of devices without a fixed system boundary, capable of better adaptation to use situations and open to evolution.

6.4 Challenges for realising the SICOD framework

In this section, we identify some of the challenges that will have to be met in realizing the SICOD framework. The aim of this section is to identify the key issues for further research rather than to provide or propose immediate solutions.

Synchronisation of the external states with an ICM. There are two related questions. The first concerns keeping the virtual model up-to-date with the real-world situation. For instance, in the drink stock control example, how can the cola count information be updated in the ICM immediately after the action of buying? The second concerns the conversion of real-world analogue states into digital states. This conversion is handled by sensor technologies, but the choice of how frequently this conversion occurs (i.e. the update rate) can affect the responsiveness and integrity of the system (cf. precision, granularity and accuracy of sensor data discussed in [Hon01]). The key question is: how can a user comprehend and manage the conversion process?

Interface for maintaining an ICM. We need simple means for users to create and modify ICMs. This issue is one of the themes of this thesis. The development of WING, EME and DMT (described in later chapters) is targeted towards simplifying the model building activities. The assumption in current EM tools development is that a flat screen-like display and a keyboard is available to the user. However, we could build interfaces of other kinds when new display technologies have evolved. For example, if we combine 3D hologram technology (e.g. [Fre02]) with precision location systems such as the one at AT&T Lab [Sen02], we can create an interface by generating 3D objects and interacting with them. In this way, it might be possible to build an ICM by physically moving virtual 3D objects.

Scalability of ICMs. In some ubicomp scenarios, we are expecting hundreds of ICMs connecting thousands of devices. To implement the SICOD framework on such a big scale it would be necessary to solve problems of reliability and efficiency.

Development of suitable terminology. Introducing the SICOD framework also introduces many terms that will be unfamiliar to end-users. While they are appropriate for academic discussions, terms such as IDM, ISM, ICM and LSD are not easily interpreted by users. We need to develop more user friendly terms that still

make the underlying concepts clear.

Integrating with existing technology. So far, we have concentrated on applying the SICOD framework 'through-and-through', expecting every device to be designed with an IDM. In the real situation, we shall have to consider an environment which includes other devices that were not designed with the SICOD framework in mind. We surely cannot throw away all current devices and replace them with new devices designed for the SICOD framework.

Safety of customisations. One good thing about traditional hard interfaces is that they prevent users from doing things that are dangerous. The open nature of ICMs might allow users to configure the system so that it exhibits dangerous behaviours. For example, imagine the consequences of connecting a car navigation system to the wrong map.

Security of private observables. Sharing observables between ICMs might sometimes be desirable. However, ICMs might also contain personal information (e.g. credit card numbers) that we might not want to share with others. Methods for attaching scopes and privileges to observables are needed so that we can specify and distinguish these sensitive observables.

Intangible interface. ICMs potentially offer a better conceptual model of the system than a traditional hard interface at the expense of sacrificing physical affordance. For example, in some contexts, the use of an ICM might be an inadequate replacement for the physical buttons on a device; it would be inappropriate to replace all the channel buttons on a TV remote controller with an ICM.

Naming conventions for the observables. How can we make sure that we are referring to the central heating system in our house and not the one next door? We shall need to standardise the naming conventions for observables. One possible solution would be to use conventions similar to URLs for the Internet.

Distributed dependency maintenance. It is relatively easy to implement a centralised model of dependency maintenance. Definitions are stored in one place. The proper order of evaluation of definitions can be determined by classical topological sort algorithms. In a distributed ubicomp environment, where definitions are scattered amongst different devices, the order of evaluation becomes undeterminable and updates of definitions become concurrent. In his thesis, Sun

proposed a client and server model of distributed dependency maintenance [Sun99a]. The result is a distributed prototype version of TkEden called DTkEden. DTkEden is a good tool for studying the issues involved in distributed dependency maintenance.

6.5 Related work

In this section, we describe other related work in the ubicomp research community. In the past five years, the lack of guiding principles for ubicomp development has been directly addressed by many research groups around the world. As a result, many 'toolkits', 'models' and 'APIs' have been developed. Although researchers have used different terms and banners for their work, at some level of abstraction, each has introduced a set of guidelines on how a ubicomp environment should be implemented. We shall refer to each set of guidelines as 'conceptual framework' or simply a 'framework' for ubicomp. In this section, we briefly review five such conceptual frameworks and compare then with the SICOD framework.

6.5.1 Toolkit framework

The framework introduced in the Context Toolkit [Dey01b] is based on generalising the idea of traditional GUI toolkits. Just as GUI toolkits separate interface concerns from program development, the Context Toolkit framework tries to separate concerns between context acquisition and the use of context in an application. There are five basic software components: context widgets, interpreters, aggregators, services and discoverers. Context widgets hide the specifics of the input devices being used from an application. Their role is similar to that of device drivers. Interpreters convert low-level context data into high-level context information. Aggregators combine context information. Services execute actions based on context information. Discoverers are responsible for maintaining a registry of other software components. Interaction between components is implemented through message callbacks. The main aim of the research is to provide "concepts that make context-aware computing easier to comprehend for application designers and developers" [Dey01b]. Little consideration has been given to users of ubicomp systems. The purpose of interpreters is to provide automatic interpretation of context data. This demands that the designer prescribes the interpretation of context, which is not easy in the dynamic environment of ubicomp.

6.5.2 Layer framework

Tandler [Tan01] tries to separate the concerns of a ubicomp application into five layers. These layers share five data models: the interaction model, the physical model, the user-interface model, the tool model, and the document model. Each layer represents a level of programming abstraction – the module layer contains tailored functionality for specific applications; the generic layer contains common functionality across applications; the model layer contains definitions of the five data models; the core layer provides hybrid implementation of the underlying infrastructure for communications, event handling, device and sensor management, automatic dependency detection and update, etc. An interesting feature of this framework is the use of a declarative description to ensure that the dependency between visualisations and attributes of shared objects is automatically maintained. This framework is specific to OO programming.

6.5.3 Middleware framework

Hong and Landay [Hon01] advocate building middleware similar to the middleware of the Internet to provide communication services for ubicomp applications. In their framework, each application is responsible for implementing standardised communication data formats and protocols. Although this adds complexity, each application can be more independent. The advantages of this framework are the same as the advantages of the Internet infrastructure – it gives freedom in choosing hardware, operating system and programming language.

6.5.4 Blackboard framework

The Blackboard framework uses the blackboard metaphor [Win01]. A blackboard is a communication centre where all communication between applications takes place. Applications can post messages on the blackboard and subscribe to particular classes of message from the blackboard. A blackboard consists of two components: an event heap and a context memory. The event heap maintains short-term message storage. The context memory is a database that provides long-term message storage. The framework is data-centric rather that process-centric. Centralised communication provides opportunities for system integration [Win01].

6.5.5 Trigger-based framework

Huang et al. [Hua99] introduced a framework for ubicomp based on extending the concept of triggers originating from databases. A trigger is a constraint-action pair. The action of a trigger will be executed when the constraint is satisfied. The framework provides automatic translation of a high-level task that is input by the user (e.g. buying a drink) into triggers that are maintained by the system. The main disadvantage of heavy reliance on triggers is that it may lead to state changes that are not easy to comprehend and anticipate. The system behaviour becomes unpredictable.

6.5.6 Overall comparisons

All five conceptual frameworks adopt an engineering approach to ubicomp application – engineers design and implement a ubicomp application; users buy the application and use it according to the user manual. However, as we have discussed throughout this chapter, because our real life environment is very dynamic and user requirements are always changing from situation to situation, the problems of design and use of ubicomp applications cannot be addressed separately. Sometimes, users can also find themselves in the designer's role. It seems unlikely that an engineering approach can address the issues of automation, visibility, connectivity and adaptation discussed in section 6.2 satisfactorily.

The five frameworks discussed in this section are technology-centred. The EM SICOD framework is human-centred. Our emphasis is on using concepts that make the resulting system comprehensible not only by the developers but also by the users. Most of the five frameworks only provide good concepts for the developer to develop ubicomp applications, limiting the scope for flexibility to the design phase of the development. The SICOD framework has the potential to extend system flexibility to users, allowing them to design and customise their own ubicomp environment.

6.6 Summary

In this chapter, we have discussed many visions for ubicomp and identified key issues associated with them. We have proposed a conceptual framework (SICOD) based on EM principles. The potential advantages of the SICOD framework have been illustrated using simple examples. Some of the challenges to be met in realising the framework have also been identified. Finally, we have compared the SICOD framework with other related ubicomp research.