More principled design of pervasive computing systems

Simon Dobson

Department of Computer Science, Trinity College, Dublin IE
simon.dobson@cs.tcd.ie

Paddy Nixon

Department of Information and System Sciences, University of Strathclyde, Glasgow UK
paddy@cis.strath.ac.uk

Abstract. Pervasive computing systems are interactive systems in the large, whose behaviour must adapt to the user's changing tasks and environment using different interface modalities and devices. Since the system adapts to its changing environment, it is vital that there are close links between the structure of the environment and the corresponding structured behavioural changes. We conjecture that predictability in pervasive computing arises from having a close, structured and easily-grasped relationship between the context and the behavioural change that context engenders. In current systems this relationship is not explicitly articulated but instead exists implicitly in the system's reaction to events. Our aim is to capture the relationship in a way that can be used to both *analyse* pervasive computing systems and aid their *design*. Moreover, some applications will have a wide range of behaviours; others will vary less, or more subtly. The point is not so much **what a system does** as how what it does **varies with context**. In this paper we address the principles and semantics that underpin truly pervasive systems.

1 Introduction

Pervasive computing involves building interactive systems that react to a wide variety of non-standard user cues. Unlike a traditional system whose behaviour may be proved correct in an environmentally-neutral state space, a pervasive system's behaviour is intended to change along with its environments. Examples include location-based services, business workflows and healthcare support, gaming, and composite access control policies.

Building pervasive computing systems currently revolves around one of two paradigms: (a) *event-handling systems*, where behaviour is specified in terms of reactions to events; and (b) *model-based systems*, in which rules are applied over a shared context model. The former leads to fragmented application logic which is difficult to reason about (in the formal and informal senses); the latter leaves a large number of rules whose interactions must be analysed, a situation known to be quite fragile. In addition, the majority of these approaches are premised on snapshot views of the environmental state.

A truly pervasive system requires the ability to reason about behaviours beyond their construction, both individually and in composition with other behaviours. This is rendered almost impossible when a system's reaction to context is articulated only as code, is scattered across the entire application, and presents largely arbitrary functional changes.

From a user perspective the design of pervasive computing systems is almost completely about interaction design. It is vitally important that users can (in the forward direction) predict when and how pervasive systems will adapt, and (in the reverse direction) can perceive why a particular adaptation has occurred. The hypothesis for our current work is that **predictability in pervasive computing arises from having a close, structured and easily-grasped relationship between the context and the behavioural change that context engenders**. In current systems this relationship is not explicitly articulated but instead exists implicitly in the system's reaction to events. Our aim is to capture the relationship in a way that can be used to both *analyse* pervasive computing systems and aid their *design*.

In this paper we describe our rationale for taking a more principled approach to the design of context-aware pervasive computing systems and outline a system that encourages such an approach, focusing on its impact on interaction. Section 2 presents a brief overview of pervasive computing, focusing on the difficulties in composing applications predictably. Section 3 explores pervasive computing from first principles to articulate the underlying motivations and factors influencing system behaviour. Section 4 describes a more principled design approach base on these factors and how they impact the interface functionality of systems, while section 5 concludes with some open questions for the future.

2 Pervasive computing

Pervasive computing can broadly be defined as *calm* technology that delivers the correct service to the correct user, at the correct place and time, and in the correct format for the environment[1]. Context, viewed alongside this definition, is all the information necessary to make a useful decision in the face of real-world complexity. More specifically, context is central to the development of several related trends in computing: the increasing pervasiveness of computational devices in the environment, the mobility of users, the connectivity of mobile users' portable devices and the availability to applications of relevant information about the situation of use, especially that based on data from physical sensors.

2.1 Context

Historically, the use of *context* grew from roots in linguistics [2]. The term was first extended from implying inference from surrounding text to mean a framework for communication based on shared experience [3]. The importance of a symbolic structure for understanding was embraced in other fields such as [4,5,6] and subsequently developed from a purely syntactic or symbolic basis to incorporate elements of action, interaction and perception.

[7] divides context into two broad classes: *primary* context is derived directly from sensors or information sources, while *secondary* context is inferred in some sense from the primary context. A typical example is when GPS co-ordinates (primary context) are converted into a named space (secondary context) through a look-up process (inference).

More recently, in the setting of pervasive computing, **context awareness** was at first defined by example, with an emphasis on location, identity and spatial relationships [8,9]. This has since been elaborated to incorporate more general elements of the environment or situation. Such definitions are, however, difficult to apply operationally and modern definitions [10] generalize the term to cover "any information that can be used to characterize situation". Current work in the field addresses issues including:

- developing new technologies and infrastructure elements, such as sensors, middleware, communication infrastructures to support the capture, storage, management and use of context.
- increasing our understanding of form, structure and representation of context;
- increasing our understanding of the societal impact of these new technologies and approaches and directing their application;

A more detailed retrospective of the academic history of context can be found in [10,11].

For this paper we conjecture that as we move away from the *define by example* notions of context there is an increasing demand to establish the foundational models for context. For pervasive computing systems there remains two fundamental problems. Firstly, the centrality of context to the progress in the field of pervasive computing demands new views on the theoretical underpinnings of context. For example there is no widely accepted operational theory or formal definition of context. There is also an immediate problem of providing to application developers ways in which they can describe the context needs of their applications in manner that is orthogonal to the application or business logic of the application. The programming primitives, frameworks, and tools are still in their infancy.

3 The semantics of a context-aware system

3.1 What *is* context?

By **context** we mean the environment in which an application is executing. This might include the identity of a user, their location, the locations of other users, the device they are using, the information, task workflows they are involved in, their goals, strategies and so forth.

The intention of making a system context-aware is to allow the detailed behaviour of the application to adapt to context while keeping the overall behaviour constant: a messaging application always delivers messages, but may deliver messages

differently in different contexts. Interface modality [12] may not be purely a device issue: a system might adapt its mode of interaction on the same device for different circumstances (such as going from vision to voice on a handheld), or might choose to switch devices while maintaining the same interaction style (such as making use of a wall screens instead of a PDA for form input).

Context is not monolithic: a given context may be composed of a number of different facets. Moreover the facets available may change between different executions of a context-aware application, for example when a new location system is installed. This implies that context-aware systems have defaults for "missing" contextual parameters, and that there is some mechanism for making new parameters "useful" to a wide range of applications. We do not, for example, want a context-aware system to be tied to a particular kind of location system, but want the location systems available at run-time to be leveraged to their fullest extent. This is essential for incremental, open deployment.

3.2 Behaviour

As stated above, the *gross* behaviour of an application should remain the same - sorting algorithms remain sorting algorithms in whatever context they execute. However, the *detailed* behaviour may change with context - the sorting criteria, for example - and it is this detail, and the way **behaviour varies**, that we are seeking to capture when talking about the semantics of context-aware systems.

One way to view this is as follows. Behaviour can be captured as a function from inputs to outputs, with some of the inputs being captured during execution. Context provides additional inputs describing the environment in which the function is being evaluated. Two invocations of the same function with the same (external) inputs may result in different behaviours because of changes in context.

We can therefore regard contextual variation as changing the contextual inputs to an underlying "ordinary" function. In what follows, when we refer to "behaviour" and "behavioural change"we mean this change in parameterisation rather than an explicit change in (the code of) the function being provided. (There is no loss of generality here as the parameter might encode a function description being passed to a universal evaluator.) From an implementation perspective this makes explicit the context on which the function's detailed behaviour depends.

3.3 Design

While much of the research on pervasive computing has its roots in the programming language and distributed systems communities, the chief design task is clearly one of interfacing - creating systems that are usable as part of a larger real-world activity. Moreover, the design task is both multimodal and dynamic.

Some pervasive computing systems will be unimodal, using a single device and interaction structure. However it is widely accepted that many will be multimodal, utilising a range of different devices across the lifetime of the interaction. This includes multiple users with different constraints.

If we consider the ability to deploy context-aware applications into a shared space, we must also deal with the interactions between these applications. This may involve negative aspects such as sharing device capabilities between applications, prioritising different (and possibly conflicting) decisions. However, there are also significant potentially positive aspects including the case where one application provides context for another that might not otherwise have been obtainable.

3.4 Behaviour variation

Some applications will have a wide range of behaviours; others will vary less, or more subtly. The point is not so much **what a system does** as how what it does **varies with context**.

Much of computer science has been devoted to the notion of *correctness* - that is, to ensuring that a system has a single behaviour, and that this is the behaviour the user wants. Context-aware systems attack the underlying assumption of a single behaviour that can be articulated, replacing it with the view that behaviour *should* change in different circumstances.

Arbitrary behavioural changes would be incomprehensible to users, and would make systems completely unusable. However, single behaviour is equally unattractive in that it prevents a system adapting to context. There is therefore a spectrum in the behavioural variation we are willing to accept (figure 1). In building a pervasive computing system we are looking for the "sweet spot" between adaptability and comprehensibility. However, this still leaves the issue of deciding *how* behaviour should change and *when* changes should occur.

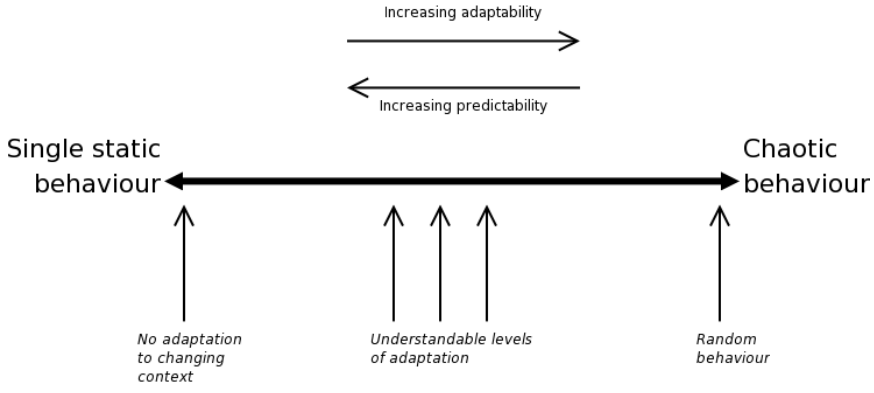


Fig. 1. The spectrum of behavioural variation.

An adaptive system adapts *to* something, and presumably adaptation happens when that something changes. Actually this turns out to be a little simplistic - adaptation may happen before or after a change - but the principle is valid. Since we are

discussing context-aware systems, we can reasonably expect a system to adapt to changes in its context.

However, not all changes in context are significant or simple. A location-based service's behaviour will not typically be different at *every* different location, so not all location cues cause changes. Similarly location may not in itself be enough to define the system's behaviour without contributions from other aspects of context.

3.5 Describing the semantics

We might regard context as having a "shape" over which the system operates. The shape is multidimensional, defined by the various contextual parameters. The shape will also have identifiable "significant" points or areas that will have meaning to the user of the application, being perceived either as points where behaviour could (or should) change, or as areas in which behaviour could (or should) remain the same.

Not only do the significant points in the context define *when* behaviour can change, for a given application they will in many cases essentially define *what* new behaviour will be selected. To take a concrete example of a service providing tourist information, we expect the information being served both to change as we move and to remain relevant to the location we are in. The interface's adaptive behaviour of the system must therefore be closely related to the external world if that adaptation is to be intuitive.

This leads to our defining observation about developing a semantics for context-aware pervasive computing: in order for a pervasive computing system to be predictable to users, **the relationship between context and behaviour must be two-way and (largely) symmetric**. An application's behavioural variation should emerge "naturally" from the context that causes it to adapt, and that variation mandates that certain structures be visible in the model of context being used. It might only adapt to large-grained changes, placing it at the static end of figure 1; alternatively it may adapt to fine-grained changes, placing it at the dynamic end. The point is that the application's position in the spectrum is not selected *a priori* but emerges naturally from the shape of its context. If a context has a fine-grained structure it will support a highly adaptable application; conversely a highly adaptive application needs fine-grained context.

An *application*, in this view, consists of four elements:

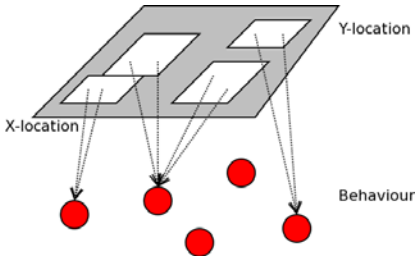
1. A baseline behaviour parameterised by a context
2. The context space with its significant points and shapes defined
3. The behavioural space with its own structures
4. A mapping matching changes in context to corresponding changes in behaviour

The first element is a standard program with adaptation hooks, and perhaps significant control structures for concurrency control and consistency maintenance. The third element describes the parameters used to control the program's adaptation. The second element describes the context expected by the application and the points at which this context forces or precludes adaptation. The fourth element describes the way in which the context adapts the program, matching significant changes in context to changes in behaviour.

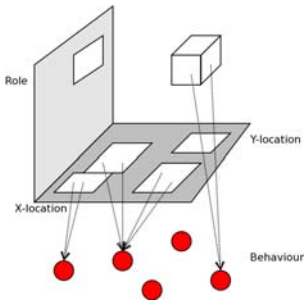
The issue of correctness reappears in another guise: instead of ensuring that a single behaviour is *implemented* correctly (and that the correct behaviour is implemented), we now need also to ensure that the behaviour *varies* correctly. The problem is not as bad as it might appear, however: if the underlying function is correct then the behaviour will be correct in *some* sense for each possible contextual parameter. The issue is one of the *appropriateness* of selecting a detailed behaviour in particular circumstances.

3.6 Towards more principled design

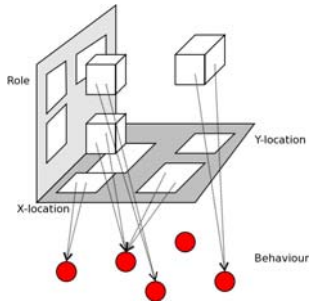
Making a function context-dependent essentially adds extra parameters to its definition. However, adding extra parameters in principle allows these additional degrees of freedom to affect the function's behaviour in arbitrary ways - a situation that is probably more general than is consistent with predictable variation. The challenge, then, is to provide additional parameters in such a way that their impact on the function's behaviour is constrained to be predictable, and follows (in some sense) the structure of the context.



(a) Location-dependent behaviour



(b) Adding role



(c) Different roles in the same location

Fig. 2. Context dependence as parameter selection.

The essence of this problem is shown in figure 2. Figure 2(a) shows a function whose behaviour (the lower circles) depends on the location in which it is executed (the plane). Different regions of the plane map to the same behaviour, so the function observed by the user will be the same as they move within this region. Change in behaviour will only be observed when they move between regions.

Adding a extra contextual parameter, such as the person's role, adds another dimension to the behavioural space[12]. The behaviour may not vary in some locations for a change in role (figure 2(b)); alternatively there may be a change for some roles in some locations (figure 2(c)).

We claimed above that behaviour should only change "on cue" from context. This suggests that the change in role needs to be clear in the interface.

From a design perspective, it would also be attractive for the changed behaviour to depend structurally on the role and location: rather than making the change arbitrary, it should emerge naturally from the parameter space. This has three major advantages:

1. It simplifies the development of the adaptive controls by placing all adaptation functions in a single sub-system
2. It simplifies the development of the adaptive components by making the parameter space clearly defined and explicitly articulated
3. It provides a "closed form" of the system's context-aware behaviour for analysis

4 A mathematical model of principled design

The discussion above leads us to consider a model in which primary context conditions and constrains secondary context and behaviour. Formalising this notion leads to a semantics of context-aware systems.

We have adopted category theory as our semantic framework, for three reasons:

1. it is naturally extensible, so we can deal with an extensible collection of contextual parameters;
2. many of the well-known categorical structures suggest, at least intuitively, that they may be useful in structuring context awareness; and
3. our eventual goal is to develop programming abstractions for pervasive computing systems, and category theory's extensive use in language semantics may make this step easier.

However, our presentation here requires no understanding of the detailed mathematics of category theory: we focus here on the structural features of the approach and how it impacts the design and analysis of interface functionality. We refer the interested reader to [13] for a fuller treatment.

[12] Of course role is usually more complicated than this diagram suggests, but it will suffice for the purposes of illustration.

4.1 Modelling primary and secondary context

A **category** is a generalisation of the familiar approach of sets and functions between them. A category consists of a collection of **objects** and **arrows** between them. The most familiar category is the category of sets whose objects are sets and whose arrows are total functions between them. The arrows are constrained to be compositional and associative, and each object has an identity arrow.

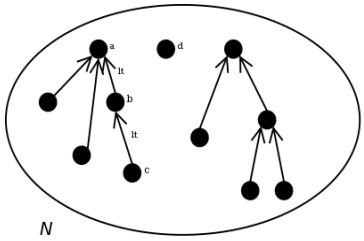


Fig. 3. Pointed structure within an object.

To each individual contextual parameter we assign an object in the category (*e.g.* a set) denoting the values the parameter can take. In a location system based on individual named spaces, for example, the "location" parameter would be represented by an object *N* whose points (elements in the case of a set) are the space names.
In many cases the elements of a parameter are themselves structured. A typical example (which occurs repeatedly) is a parameter structured as a partial order, pointed set or lattice, where each element can be "included" in at most one other (figure 3). For named spaces there is an arrow from the parameter object to itself, taking each space to its containing space or to itself if it is a "top" space. By repeatedly applying this operation we can navigate from a space up its container hierarchy. In figure 3 this means that the inclusion morphism *lt* takes space *c* to space *b*, space *b* to space *a*, and spaces *a* and *d* to themselves (we have omitted these arrows for clarity).

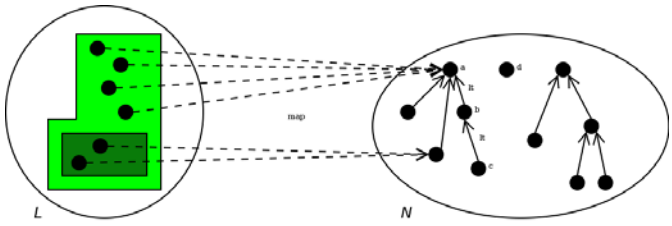


Fig. 4. Deriving secondary context.

Named spaces are probably secondary context, derived from a lower-level location system such as GPS. GPS can be modelled as an object *L* of GPS co-ordinate pairs. An obvious contextual constraint is the mapping between a GPS location and the

named space containing it. We can represent this as an arrow *map:* $L \rightarrow N$ capturing the "map" (figure 4). It is important to realise that this is a *semantic* characterisation of what would implementationally be a lookup operation, the details which can be abstracted in the analysis.

Figure 4 makes clear the structural relationship between the two parameters; A region of L maps to an element of N in such a way that elements of the containing region in L must map to an element of N containing the original element. *map* is constrained to reflect the structure of one object in another, and it is this correspondence that preserves meaning in the interface.

4.2 Context as behaviour

Current context-aware systems are not uniform, in the sense that much of a system's behaviour is conditioned by information not held in a single context model. For the purposes of analysis it is simpler to regard context in the wider sense as the sole arbiter of behaviour: the system is functional with respect to its context. (We regard this as a sound implementation strategy too.)

The easiest way to accomplish this to include the "real" parameters to the external behaviour in the context. For a simple example, consider a wireless document system which delivers a set of documents depending on the user's location. The corpus of documents being managed can be represented as a contextual parameter (object) D whose elements are possible sub-sets of documents being served related by set inclusion.

We may now define an arrow *serve:* $N \rightarrow D$ which selects the set of documents to be served by the document system in each location. Although this arrow does not define behaviour in the normal sense of describing exactly what will happen, it *does* describe how the parameter passed to that behaviour will vary. We may therefore to some extent treat D as a proxy for the behaviour of the system and study how this "behaviour" changes with context.

4.3 Analysing the structure of behaviour

Even in this simple model there are a number of questions we may ask of the system. Key to these is an understanding of the way in which *different* contexts select the *same* behaviour. Using figure 4 as an example, there are a number of points in L that map to the same element of N. This is captured by the categorical notion of a **fibre**: given an element a of N the fibre of *map* lying over a is the sub-object of L that maps to a under *map*. Similarly the fibres of *serve* above represent the spaces in which the system will serve the same set of documents.

The significance of fibres is that they capture both those contexts in which the system will behave the same and the points at which that behaviour changes.

4.4 Compound context and behaviour

One of the advantages of category theory is that it has several strong notions of composition that can be used to create complex concepts by construction. A good example of this is the use of products of context and behaviour.

If C and D are contexts (objects) we can create a product context $C \times D$ whose elements are ordered pairs of elements from C and D respectively. Moreover there is an arrow between an element (i, x) and (j, y) if there is an arrow on C from i to j and an arrow on D from x to y.

Such products represent the compound state of the system: If we take N and another context P of people's identities, the compound context $P \times N$ represents a person in a named space. We can use this product contexts to contextualise behaviour in the normal way, by specifying an arrow *serve'*: $P \times N \rightarrow D$ defining how the documents available vary with identity and place. The risk here is that such behaviour will be arbitrary, in that there is no necessary relationship between the way behaviour changes with identity and the way behaviour changes with identity *and* location. In many cases we may wish to ensure that such a relationship is preserved.

If we have arrows *serveto:* $P \rightarrow D$ and *servein:* $N \rightarrow D$ we can model this by *constructing* the arrow *serve'* from the two more elementary arrows, in such a way that *serve'* preserves some of their features. For example, we might constrain *serve'* so that it always serves a set of documents that includes the set identified by *serveto* – location context may *broaden* the behaviour but always maintains the behaviour of *serveto* as a "core". Conversely we might force *serve'* to never serve a larger set of documents than permitted by *serveto* – the underlying arrow specifies the "extent" of the behaviour. A third possibility is that location "adds nothing" to the behaviour, when *serve* defines the same behaviour as *serveto*. Similar arguments apply to *servein*.

These constructions allow us to potentially specify the constraints on complex behaviours in terms of simpler behaviours. This is important both for tackling the complexity of the system and ensuring its consistency. A user of *serve'* that preserves *serveto* as a core, for example, will be able to form a mental model in which (a) they can rely on a certain minimum behaviour everywhere, and (b) their location may add significant new documents. This consistency is vital to the usability of the system, and can be made a direct consequence of its categorical model.

Similar techniques can be used when contextualising a product context, where (for example) two behaviours B_1 and B_2 are combined to form a compound behaviour $B_1 \times B_2$ that specifies two aspects of the system independently. Again, composition of underlying arrows can be used to constrain the way in which behaviour varies.

4.5 Composition and conflict analysis

Pervasive computing almost implies dynamic composition, in that we expect mobile systems to be carried around by users and to "discover" resources as they move. This brings positive and negative possibilities: new capabilities may become available very easily, but systems may interact in undesired ways. A major challenge for analysis is to detect such conflicts.

In certain simple cases we can both detect conflicts and identify "safe" zones when two systems are composed. Suppose we have two systems with the same context and behaviour, described by two arrows $f,g : C \to D$: for the wireless document server these might be the public and private document servers. If we run both systems together, we may ask whether they will both serve the same document set for a given user and location. A categorical construction called an **equaliser** captures the sub-object C' of C in which f and g behave the same. If we can ensure that the system will remain in this region C', the systems may be composed safely; if it strays outside then the two systems diverge. Another possibility is to force g (for example) to serve as a core or extent of f.

In both cases the composition of systems is captured cleanly within the categorical model, and can be analysed using standard techniques. This may in turn lead to improved implementation techniques.

4.6 Designing "graspable" systems

Systems analysis, while important, is in many ways less interesting than systems design: we want to develop pervasive computing systems that are *usable and predictable by design*, using a model that both aids in this process and in the analysis of the results.

The fibre structure of arrows provides a powerful technique for designing systems as well as analysing them. Suppose we want to design our wireless document server so that it serves a set d_1 of documents in those places in the vicinity of a place n_1, and another set d_2 in the vicinity of n_2. If we constructed this system from scratch we would need to ensure that it responded to location events in the correct manner - an arduous testing process.

However, we can observe that the system behaves the same within a fibre - changes in context that remain within a fibre do not affect the behaviour. We need only ensure that all the places around n_1 lie in the fibre of d_1 to be convinced that the system will behave as required.

From a user perspective, in order to be predictable a change in behaviour must be accompanied by a perceptible change in the context that "makes sense" for the application at hand. Changes in behaviour occur when context moves between fibres. If we ensure that these changes correspond to external contextual cues that will convey the need for behavioural change to the user, then the user will be able to develop an appropriate mental model of the way in which the behaviour changes in response to context. The cues in the outside world are reflected exactly in the fibre structure of the model.

We claimed in section 4 that, in order for a pervasive computing system to be comprehensible, the relationship between context and behaviour needed to be largely symmetrical. It is this matching of fibre structure to external cues that captures this symmetry, either constructively (for design) or analytically (for analysis).

Although the matching of cues to fibre transitions is application-dependent and generally external to the model, it is sometimes possible to capture the cues within the structure of the category. If, for example, we can identify the context points at which behaviour should change, we can often identify the "internal" points where it should

remain the same, corresponding to the fibre over the desired behaviour. These regions - sub-objects of the overall context - can have their behaviour described individually, with the "full" behaviour coming by composition in a way that will detect many conflicts automatically. This means that a user-centred design that identifies the adaptation points in the environment can be used directly to construct a mathematical description of the system being constructed, carrying usability concerns directly into the system model.

5 Conclusion

We have motivated using a more principled approach to the design and development of context-aware pervasive computing systems, and presented a formal approach that captures some of the essential driving forces in a natural and compositional way. We have shown how certain aspects of usability and predictability in the requirements for a pervasive computing system can be given a formal realisation within a system model suitable for use as a basis for analysis and design.

Perhaps more than any other potentially mainstream technology, pervasive computing requires that we take an automated approach to system composition and variation - the alternative would constrain deployment to constellations of devices and information sources that could be described *a priori*. This in turn means that we need to be able to state very precisely the way in which system behaviour varies. This is the point at which our work diverges from that in the ambient calculus[14] or bigraphs[15] - two very prominent and influential formal treatments of mobile systems - in that we sacrifice the precise characterisation of system behaviour in favour of broad-brush analysis. We also do not privilege location, regarding it as just one of the possible contextual parameters to be studied.

The obvious counter in this formulation is that the baseline behaviour needs to encapsulate all possible adaptations, which are then selected by context. While this is correct to an extent, we should differentiate between the abstract semantic model of a context-aware application and its concrete realisation. One would not necessarily pass context as a parameter to a function: it might be preferable to allow the function to access a shared context model, and provide some templated mechanism for this model to affect its behaviour. There are, however, serious engineering problems to be overcome in developing a programming model under this model.

Although we have not investigated it in this paper, a design approach such as we propose needs to be backed by an engineering methodology. In particular we have largely elided the way in which a designer would decide on the correct formulation for context and behaviour, or check that his choices relate correctly to the users' perceptions of the system. While traditional analysis and design methods can help address these problems, there is also a need to deploy detailed usability evaluations - possibly modified for pervasive computing - to inform the feedback loop. This is a subject that is outside our expertise but that we would be keen to explore further.

It seems unlikely that the techniques described are sufficient to address the full range of context-aware behaviours, so there is a major open question in the applicability of the techniques to real-world applications - something we are

investigating at present. We are also addressing the limitation of the model to "immediate" context, where only the current situation (and not the past or possible future) affect behaviour. However, we believe that "closed form" expressions of context awareness are a key enabler for building the next generation of complex pervasive computing systems.

References

1. Weiser, M..The computer for the 21st century. Scientific American (1991)
2. Winograd, T. Architecture for context. Human Computer Interaction **16** (1994) 85-90
3. Minsky, M. A Framework for Representing Knowledge. In The Psychology of Computer Vision. McGraw Hill (1975)
4. Brooks, R. A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation **2** (1986)
5. A.Draper, B., Collins, R.T., Brolio, J., Hansen, A.R., Riseman, E.M. The schema system. International Journal of Computer Vision **2** (1989)
6. Bajcsy, R. Active perception. Proceedings of the IEEE **1** (1988) 996-1006
7. Salber, D., Dey, A., Abowd, G. The Context Toolkit: aiding the development of context-enabled applications. In Proceedings of the ACM Conference on Computer-Human Interaction, CHI'99. (1999) 434-441
8. Ward, A., Jones, A., Hopper, A. A new location technique for the active office. IEEE Personal Comunications **4** (1997) 42-27
9. Rodden, T., K.Cheverest, Davies, K., Dix, A. Exploiting context in HCI design for mobile systems. In Workshop on Human Computer Interaction with Mobile Devices. (1998)
10. Dey, A. Understanding and using context. Personal and Ubiquitous Computing **5** (2001) 4-7
11. Crowley, L., Coutaz, J., Rey, G., Reignier, P. Perceptual components for context aware computing. In Proceedings of Ubicomp 2002. (2002)
12. Calvary, G., Coutaz, J., Thevenin, D. A unifying reference framework for the development of plastic user interfaces. In Proceedings of EHCI'01. Volume 2254 of Lecture Notes in Computer Science., Springer Verlag (2001)
13. Dobson, S., Nixon, P. Towards a semantics of pervasive computing (just the category theory). Technical report, Department of Computer Science, Trinity College Dublin (To appear)
14. Cardelli, L., Gordon, A. Mobile ambients. In Nivat, M., ed. Foundations of software science and computational structures. Volume 1378 of LNCS. Springer Verlag (1998)
15. Jensen, O.H., Milner, R. Bigraphs and mobile processes. Technical Report UCAM-CL-TR-570, University of Cambridge Computer Laboratory (2003)

Discussion

[Nick Graham] This is a semantic framework that is instantiated over a specific application. This seems to require the modeller to anticipate the possible contexts or compositions that may arise.

> [Simon Dobson] This is less a problem than with other approaches. In effect, we can define compositions without having to specify what kinds of things are being composed. This is sufficiently rich to allow interesting analyses.

There are a small set of composition operators that seem to recur frequently: although we have to select which operator to use when we encounter a new contextual parameter, we often don't need to know its details to do something meaningful.

[Helmut Stiegler] Category theory is all about commutative diagrams. You did not show any such examples, in which you can apply such diagrams. Do you have some ?

[Simon Dobson] Yes, we have them used. I suppressed them here on purpose. You will be able to find them in a technical report.

[Gerrit van Der Veer] How do the notions of "conflict" and "problem" relate to the framework ?

[Simon Dobson] These notions are not automatically specified, but have to be stated explicitly in order to reason about them.