**Simon Dobson and Paddy Nixon**

Department of Computer Science
Trinity College, Dublin IE

Department of Computing and
Information Sciences, University of
Strathclyde, Glasgow UK

simon.dobson@cs.tcd.ie
paddy.nixon@cis.strath.ac.uk

# More principled design of pervasive computing systems

## Introduction

Pervasive computing involves building systems that will respond to a range of novel cues

Our current work is targeted at *improving the analysis and design of pervasive computing applications*

1. Formal models of context and behaviour
2. Map between them to structure adaptation
3. Develop insight for new programming structures

This talk introduces the early work on the first two, and points towards the future of the last

## Talk overview

1. Challenges from pervasive computing
2. Back to first principles
3. The start of a mathematical model of behavioural variation
4. Where we are and where we're going

## Pervasiveness and context

Deliver the correct service to the correct user at the correct place and time, and in the correct format for the environment [Weiser, 1991]

Must reason about behaviours beyond construction

Context is the complete environment of a behaviour, understood symbolically

- *Primary* context from sensors, used to infer *secondary* context
- No widely-accepted operational or representational theory
- No general ways to integrate context cleanly into applications

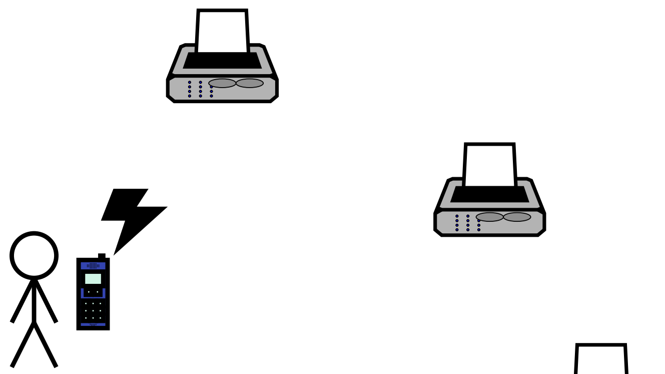
## Current application approaches

Two basic models

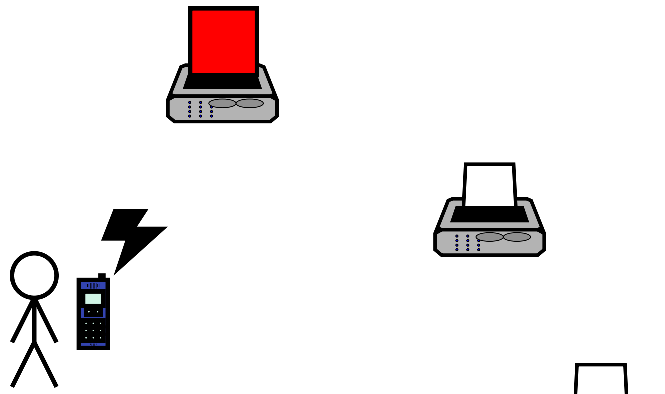
- Event-handling systems – behaviour specified as responses to events
  - Fragmented logic, semantically opaque, difficult to combine
- Model-based systems – rules applied to a shared context model
  - More difficult to scale, difficult to combine

Neither explicitly captures the *structure* of the environment and the *dependence* that variation has on it

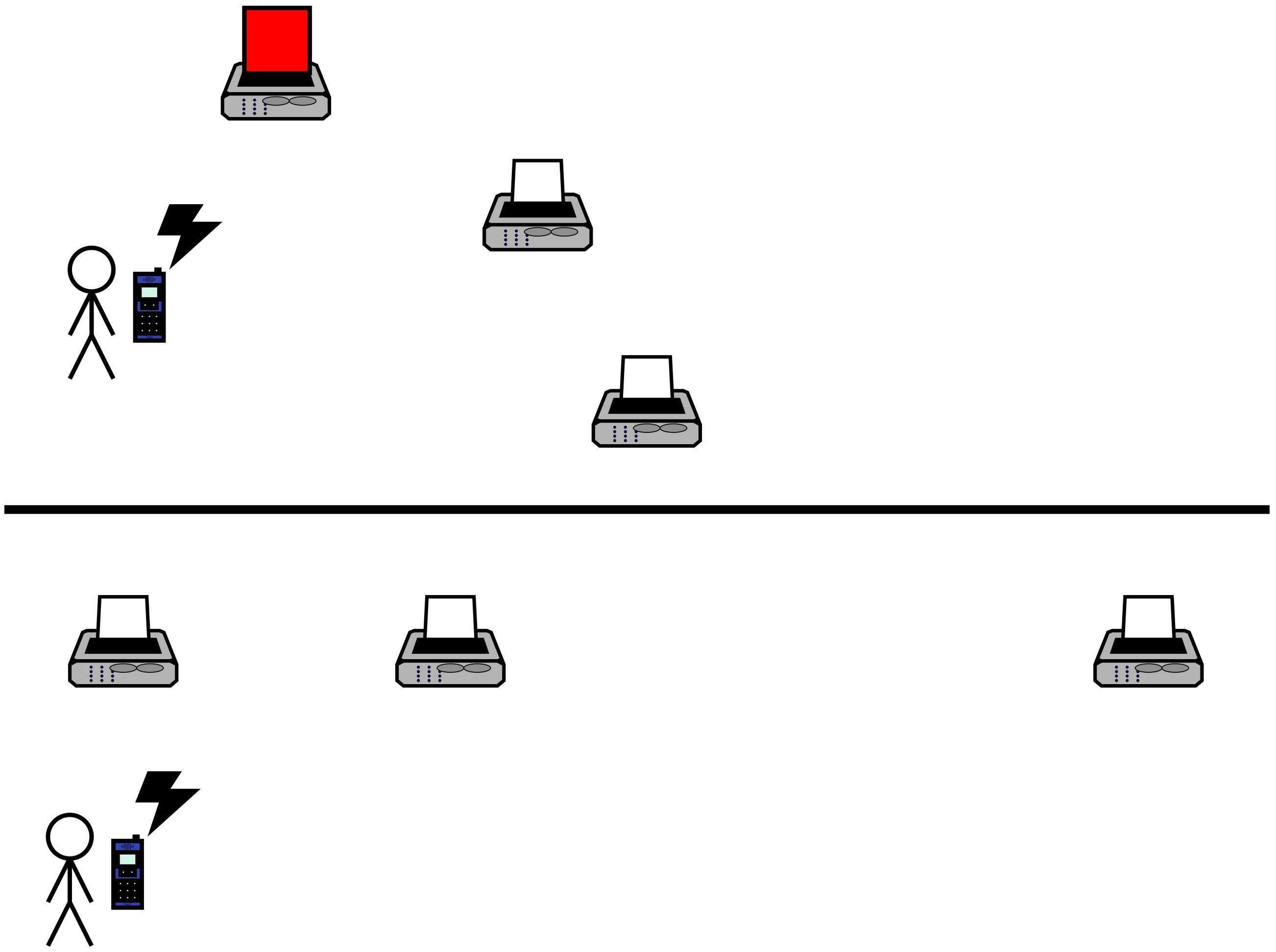
## Why composition is hard



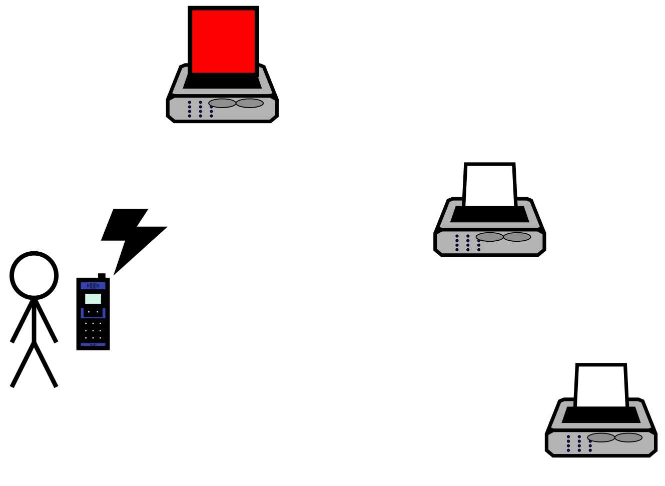
## Why composition is hard

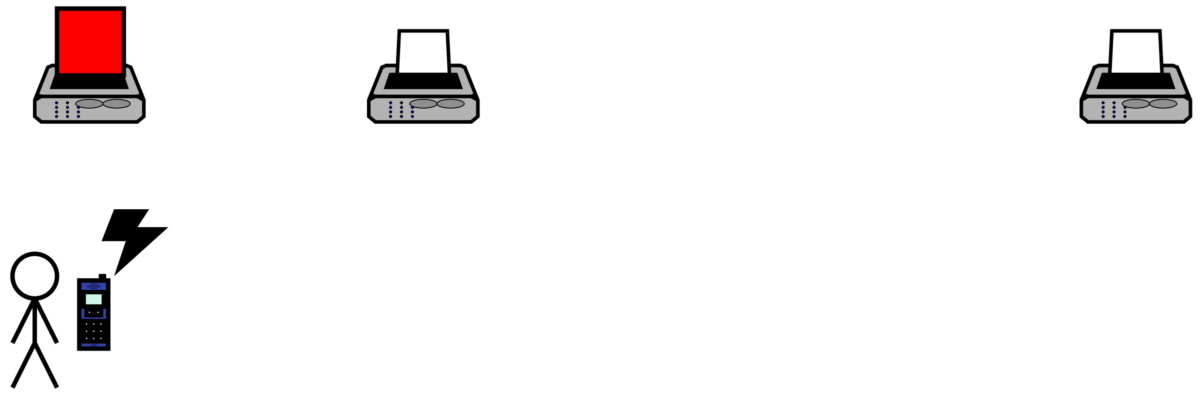


# Why composition is hard

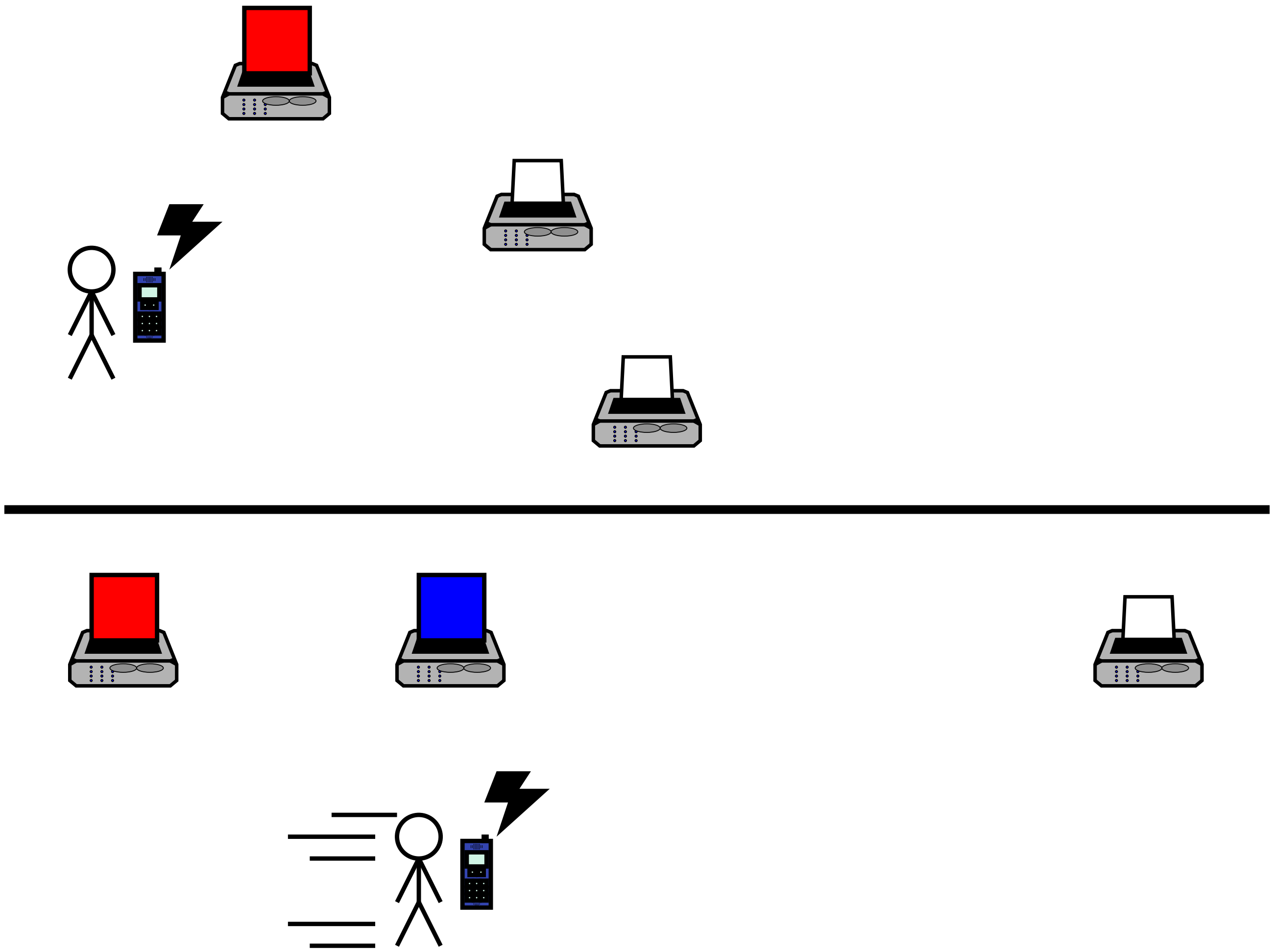


## Why composition is hard

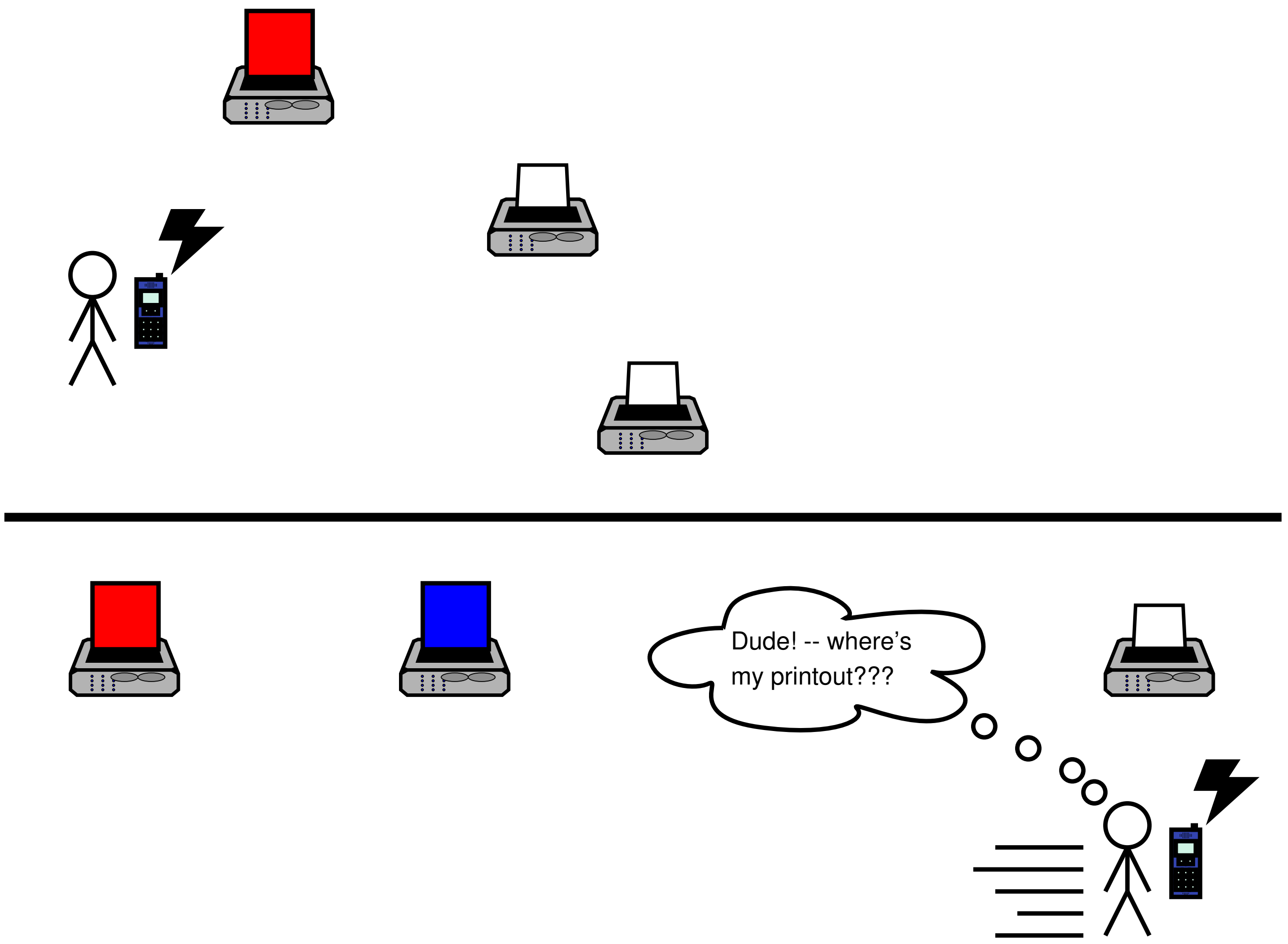




## Why composition is hard



## Why composition is hard



## Semantic essentials

Behaviours that are correct as *points* may give rise to a behaviour that is incorrect as a *process*

- Sequential composition of two correct behaviours is not necessarily correct
- Optimal solution depends on *entire* context and may change as more detail emerges

The key point is *stability*: if we perturb a system's context, how do its adaptations change?

- Keep the system stable enough to be grasped but dynamic enough to be meaningfully adaptive

## Hypothesis

Predictability in adaptive systems comes when there is a clear, structured relationship between the contextual and behavioural spaces

Put another way:

- A user should be able to anticipate changes in behaviour by looking at approaching changes in the environment; or
- Having observed a change, there should be an explanation in terms of the context that caused it; and
- The behaviour should be continuous with respect to the users' perceptions of the task

## Environment and change – 1

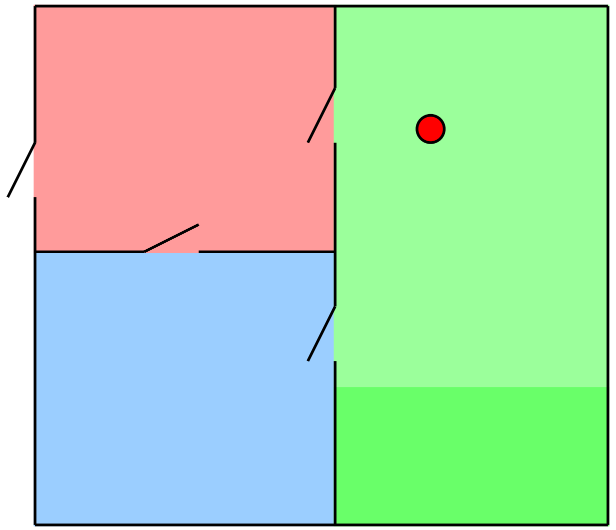
Pervasive systems are inherently compositional

- Can’t pre-specify all the relevant context
- Can’t avoid multiple systems in a space
- . . . and don’t want to anyway

## Environment and change – 1

Pervasive systems are inherently compositional

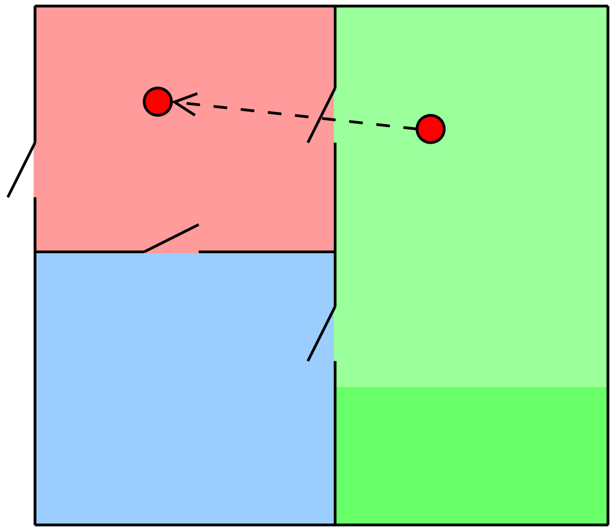
- Can't pre-specify all the relevant context
- Can't avoid multiple systems in a space
- . . . and don't want to anyway



## Environment and change – 1

Pervasive systems are inherently compositional

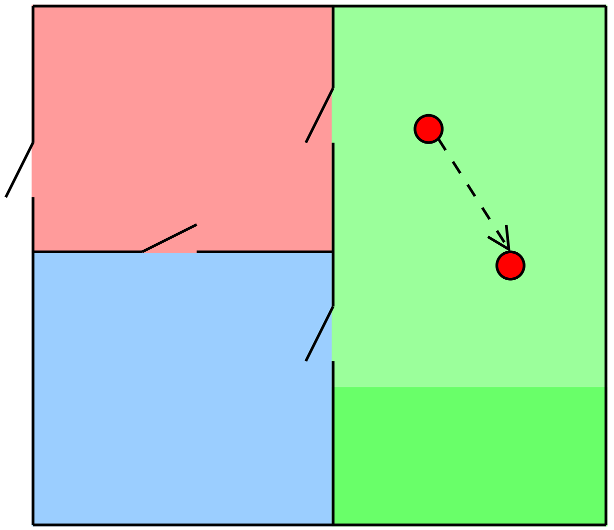
- Can’t pre-specify all the relevant context
- Can’t avoid multiple systems in a space
- . . . and don’t want to anyway



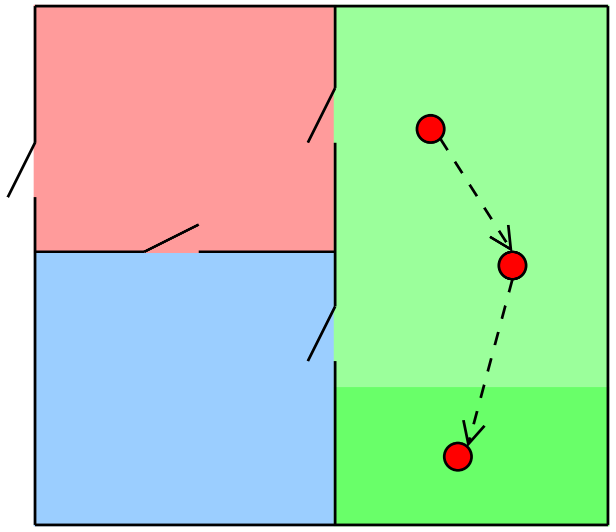
## Environment and change – 1

Pervasive systems are inherently compositional

- Can't pre-specify all the relevant context
- Can't avoid multiple systems in a space
- . . . and don't want to anyway



# Environment and change – 1



Pervasive systems are inherently compositional

- Can't pre-specify all the relevant context
- Can't avoid multiple systems in a space
- . . . and don't want to anyway

There is a natural symmetry between the structure of the environment and the structure of application behaviours over that environment

## Environment and change – 2

If we take a wide enough view of the situation, *every* behavioural change maps to some *perceptible* change in the environment

- Physical factors
- Also non-physical – two sides of a room handled differently

## Environment and change – 2

If we take a wide enough view of the situation, *every* behavioural change maps to some *perceptible* change in the environment

- Physical factors
- Also non-physical – two sides of a room handled differently

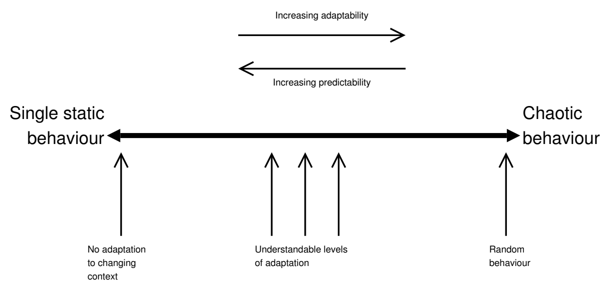
Put another way, there are *"seams"* in the context

- Not seamless – crossing a boundary is significant
- . . . but things stay the same within a boundary

Less interested in what a system does than in how what it does changes with context – semantics with a broad brush . . .

## Composite meaning

An application's place in the spectrum of adaptations emerges naturally from the "shape" of its context


Increasing adaptability
Increasing predictability
Single static behaviour
Chaotic behaviour
No adaptation to changing context
Understandable levels of adaptation
Random behaviour

We can find four fragments of the overall meaning:

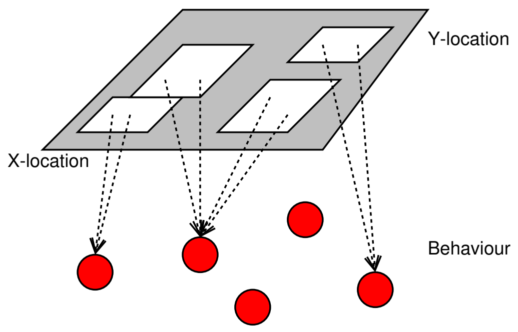
1. The "baseline" behaviour
2. The context space, with structure
3. The behavioural space, also with structures
4. A mapping from changes in context to corresponding changes in behaviour

## Towards more principled design

The challenge is to describe how the baseline behaviour varies, in a way that composes properly as context is elaborated

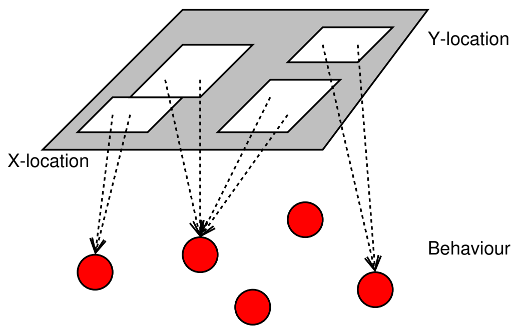
## Towards more principled design

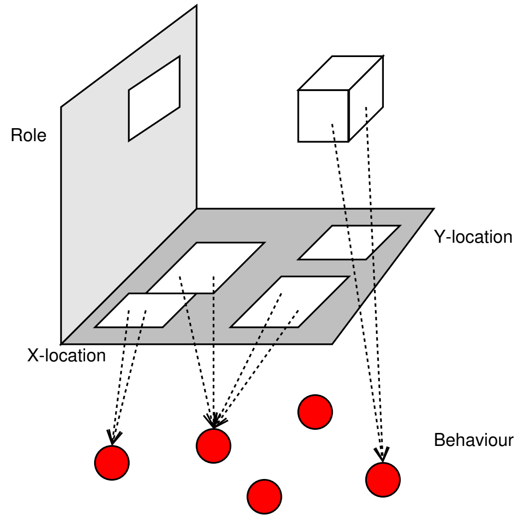
The challenge is to describe how the baseline behaviour varies, in a way that composes properly as context is elaborated



## Towards more principled design

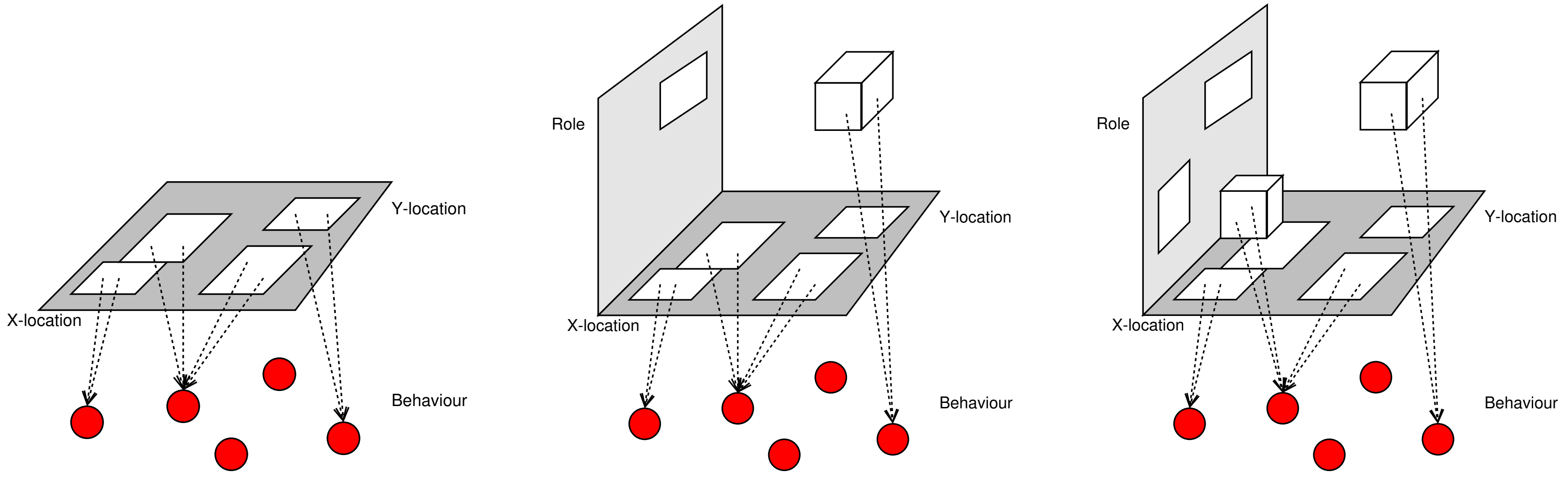
The challenge is to describe how the baseline behaviour varies, in a way that composes properly as context is elaborated





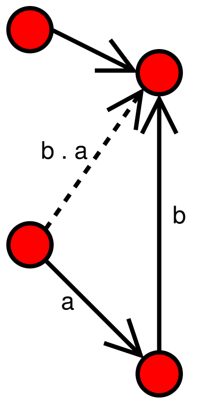
## Towards more principled design

The challenge is to describe how the baseline behaviour varies, in a way that composes properly as context is elaborated



Articulate the complete environment and the application's responses to it

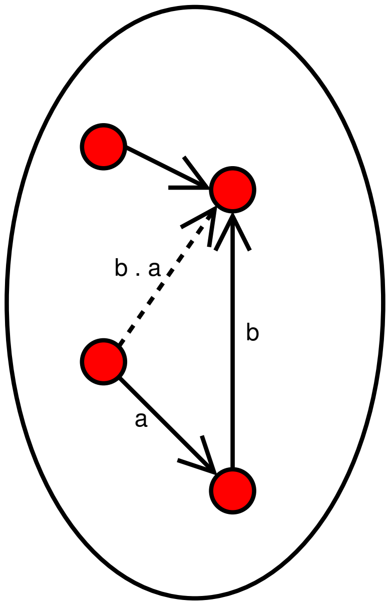
# Category theory on one slide



Object - an abstract value, like a member of a set

Arrow - mapping between objects, with composition and identity

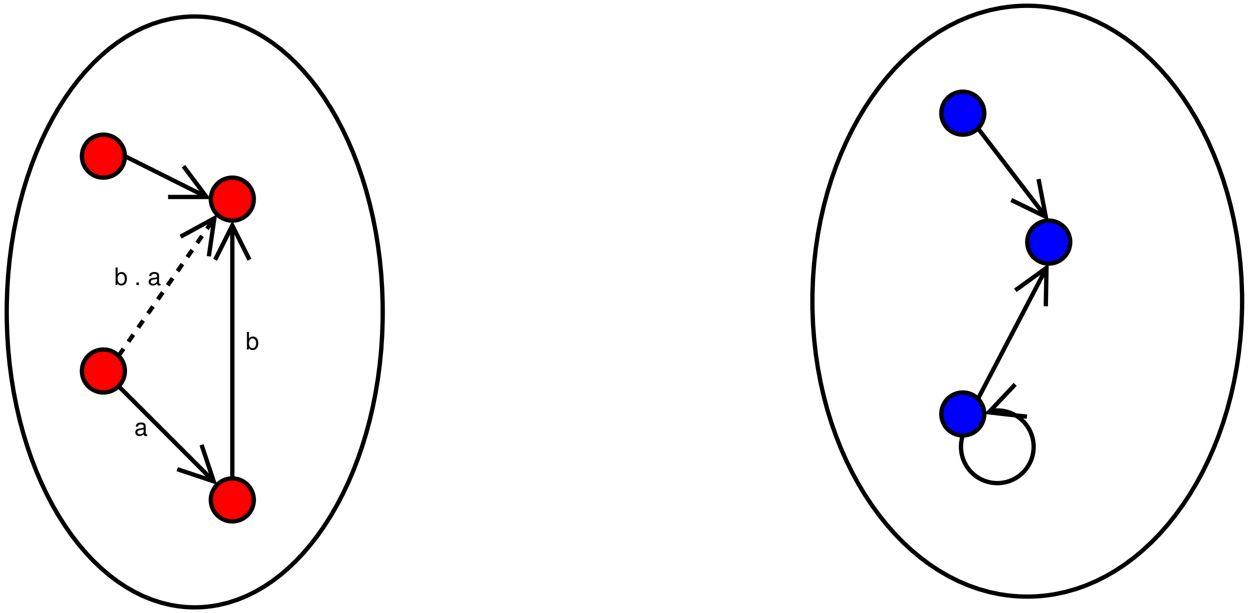
# Category theory on one slide



Object - an abstract value, like a member of a set

Arrow - mapping between objects, with composition and identity

# Category theory on one slide

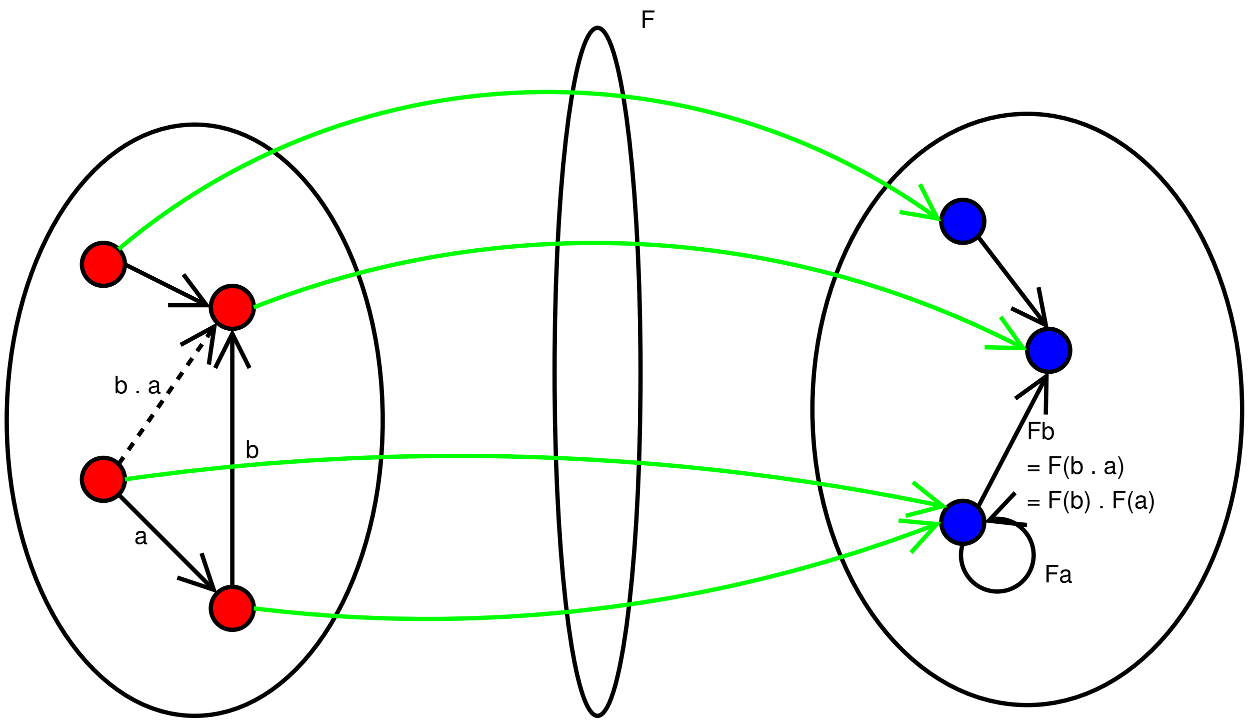


Object - an abstract value, like a member of a set

Arrow - mapping between objects, with composition and identity

Category - collection of objects and arrows

# Category theory on one slide



Object - an abstract value, like a member of a set

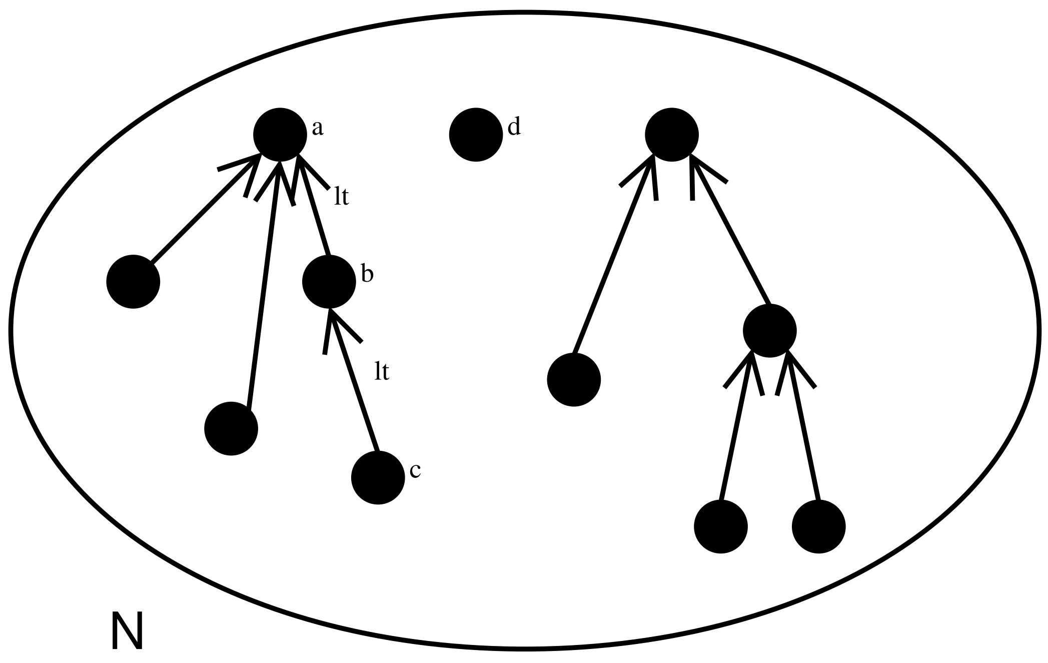
Arrow - mapping between objects, with composition and identity

Category - collection of objects and arrows

Functor - mapping between categories preserving identities composition of arrows

## Context as category – 1

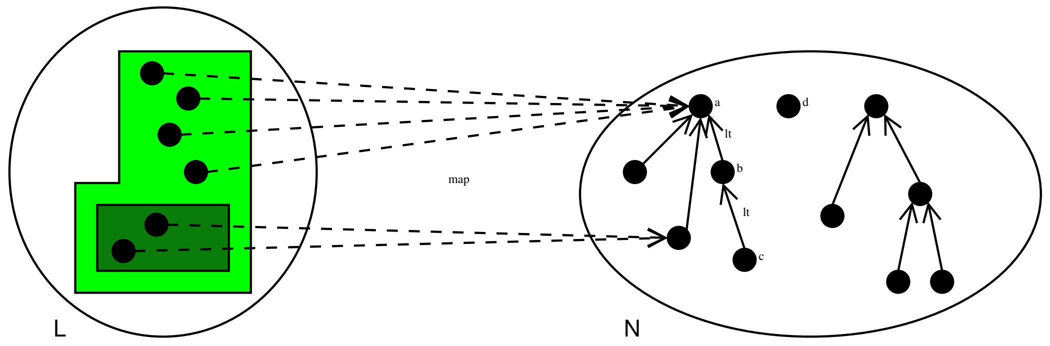
By representing context as a category we capture the structure inherent in it in the patterns of arrows



Often simple, *e.g.* containment of named spaces

## Context as category – 2

Secondary context can be represented as a functor from primary, *e.g.* from GPS locations to named spaces



Using categories gives us the necessary consistency

- For example the secondary context has to respect any structure in the primary context – for better or worse

## Context as behaviour

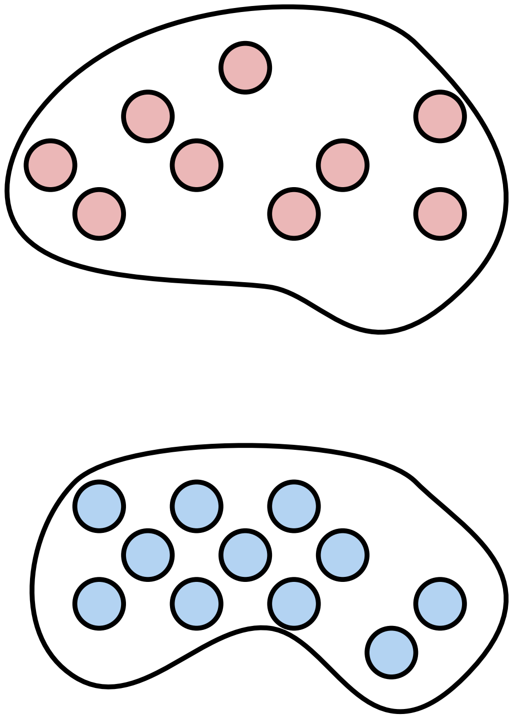
As well as representing context as a category, we can represent the details of behaviour as categories too

- An object parameterising the base behaviour
- In fact, behaviour variation *is* context – a secondary derived by a (probably complex) functor

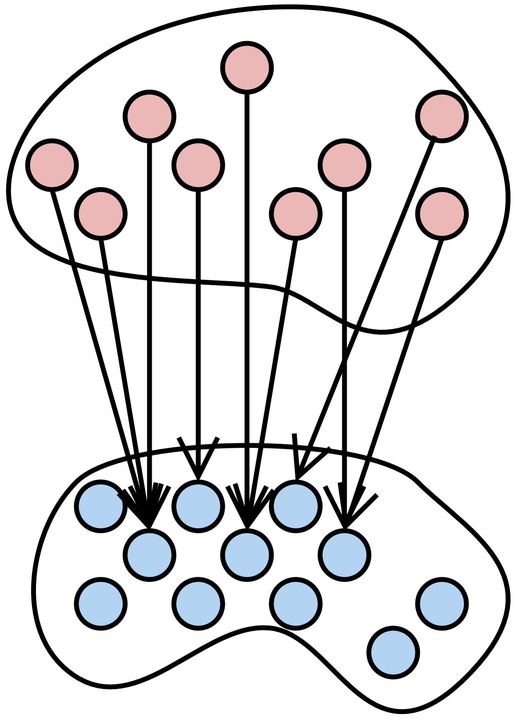
For example, a wireless document system

- Baseline behaviour is the document service
- Adaptation is the set of documents served

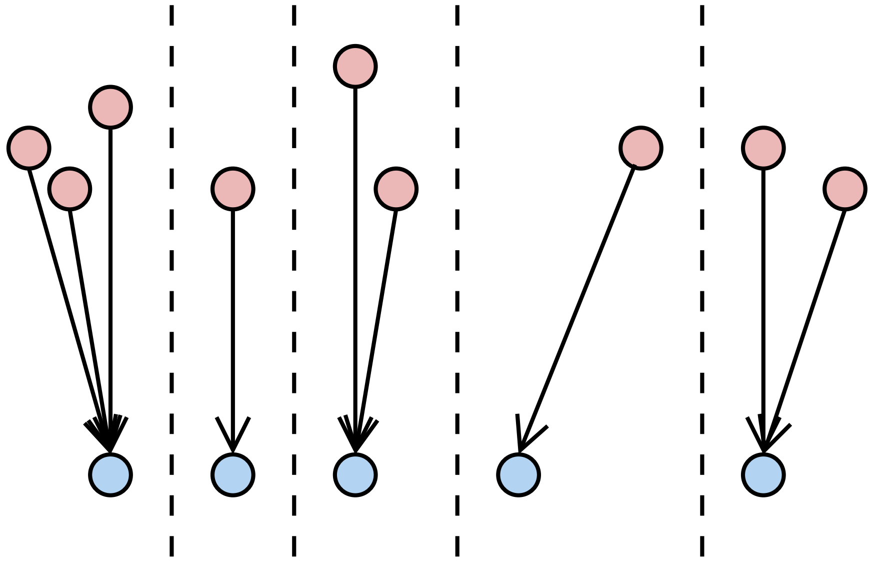
## Fibres



## Fibres



## Fibres



The fibre structure captures the global variation of the system

- The contexts in which it behaves the same
- The points at which it changes

## Change in the right place

Even so simple a model can answer some interesting questions

- Which different contexts select the same behaviour?
- Conversely, at what points will behaviour change?

Align the fibre transitions with the contextual changes that are appropriate for the application

- Formalise “different context”

*We can use the same techniques “forwards” for design and “backwards” for analysis*

## Composition – 1

Returning to the wireless document server, we might have two different variations

- By user – what documents can I see?
- By location – what can I see here?

Need to separate these concerns where possible, and compose them to get the desired behaviour

## Composition – 2

We can formalise three behaviours categorically:

1. No change in behaviour (context-independent)

## Composition – 2

We can formalise three behaviours categorically:

1. No change in behaviour (context-independent)
2. Make sure a user always sees a “core” of documents – perhaps more in some locations

## Composition – 2

We can formalise three behaviours categorically:

1. No change in behaviour (context-independent)
2. Make sure a user always sees a “core” of documents – perhaps more in some locations
3. Make sure a user never sees more documents than the “extent” of the document base – less in some places

Help form mental models, *e.g.* some documents are always available

Direct consequence of categorical treatment

## Conflict analysis

Service discovery implies dynamic conflicts between behaviours

In some simple cases we can identify “safe zones”

- The context in which the two systems behave the same
- If we ensure they stay in this zone, there will be no perceptible conflict

Either force safety (design) or detect conflict (analysis)

*This uses two categorical notions of sub-object and equaliser*

## Status and plans

Motivated by the need to *engineer* pervasive systems, not just build them

The start of a semantics for adaptation and variation in pervasive computing systems (and beyond)

- Design – get the changes right
- Analysis – why a change was wrong

Looking at improving analysis and embedding the notions directly into programming constructs

*Many of these ideas give rise to interesting new type theories and language constructs*

## Four things to take away

1. Articulate both context and behaviour – these are the semantic essentials
2. A more “topological” model provides stronger formal notions of continuity of behaviour
3. Support closed-form analyses, not pointwise adaptations that may not compose
4. It may be possible to mirror many interface concerns at a direct semantic level