**Simon Dobson**

Systems Research Group
Department of Computer Science
University College
Belfield, Dublin IE

simon.dobson@ucd.ie

# Hybridising events and knowledge in an infrastructure for context-adaptive systems

Department of Computer Science
Roinn na Ríomheolaíochta

# Overview

## Context-aware systems

- Context – situation of a system, understood symbolically
- Adaptation – change in behavioural details that respects an overall "core" underlying behaviour

## Current programming models are typically event-based

- React to sensor observations, user actions etc

## We claim that such direct connections can prove problematic in larger systems

- Recognise the limits of events
- …and hybridise them with a more knowledge-based approach to improve responsiveness

# Context and programming

Some core problems of context-adaptive systems

a) What adaptations do I make in what situations?
b) How do I know when these situations have occurred (or are occurring)?
c) How do I know the adaptations I make are “correct”?

The first is an issue of design, the third of verification, and the second of contextual reasoning

- How is sensed information translated into situations?

A typical pervasive (or other context-adaptive) system will use some form of instrumentation to sense its environment and trigger adaptations in its behaviour

# Event systems

## Event systems are very common

- Formalised as process algebra, used extensively in distributed systems (publish-and-subscribe, message-oriented middleware, …)
- Highly scalable from a systems perspective
- Can be tricky to program in the face of complex interactions

## Natural to build context-aware systems on top of messaging infrastructures

- Sensors and other contextors provide events
- …which are consumed by applications
- Leverage existing, scalable platforms

*Generalised context capture is discussed by Coutaz and Rey*

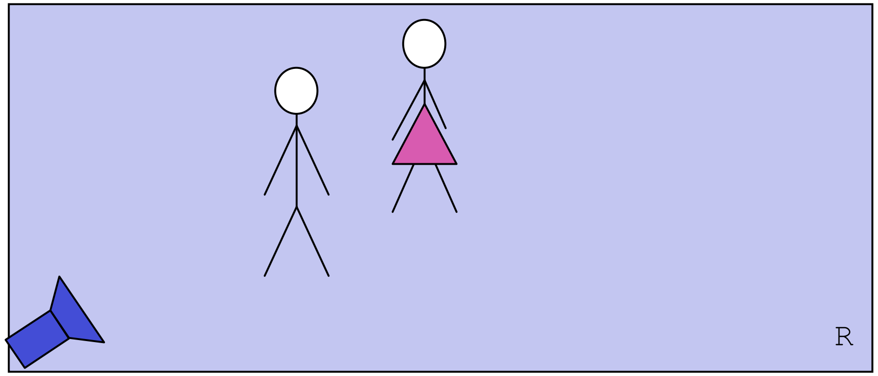
*A good example is the "sentient objects" model of the EU CORTEX project*

## Represent situations by "compound" events

- and- or or-combinations of simple events

*For example the work of Hayton et al*

# Adapting to events




Time

enter(A,R)
enter(B,R)

Each observation gives rise to a simple event, published to all subscribers

<enter(A,R) … enter(B,R)>
or
<enter(B,R) … enter(A,R)>

Compound event recogniser

trigger

Adaptation

The compound event is sent to the behaviour to trigger adaptation

# Hidden assumptions

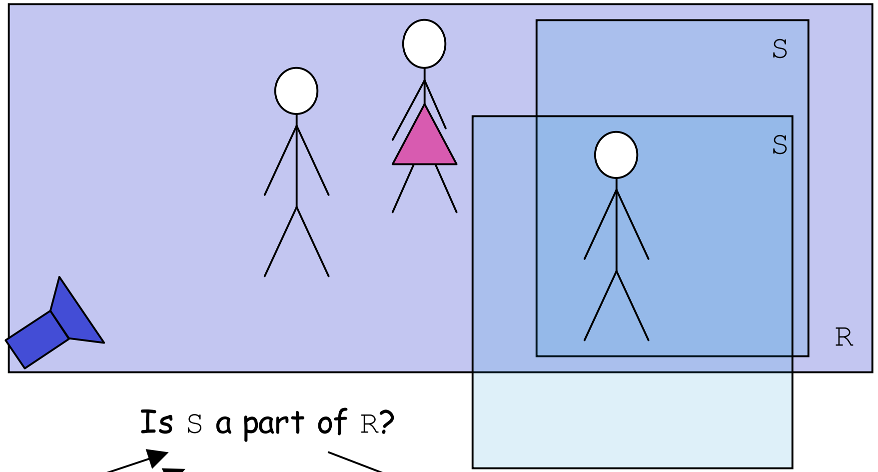
The simplicity of this model rests on two key assumptions

- The semantic independence of events
- The accuracy of information

If *either* of these assumptions is broken, the event system deteriorates

Unfortunately, in pervasive systems, they *both* are

# No cancellation – 1




Is S a part of R?

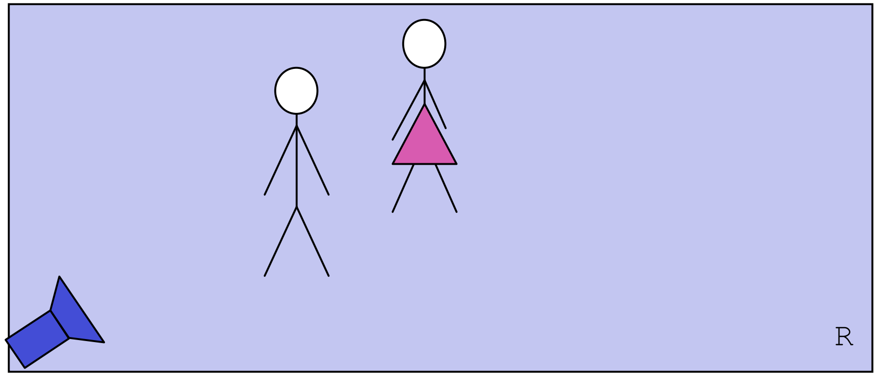
enter(A,R)
enter(A,S)
enter(B,R)

Or is S a separate space, or an overlapping space, or…

<enter(A,R) … enter(B,R)>
or
<enter(B,R) … enter(A,R)>

Without knowing the spatial relationships we can't decide whether to fire the adaptation, so the event system suddenly needs detailed contextual knowledge

# No cancellation – 2




enter(A,R)

leave(A,R)

enter(B,R)

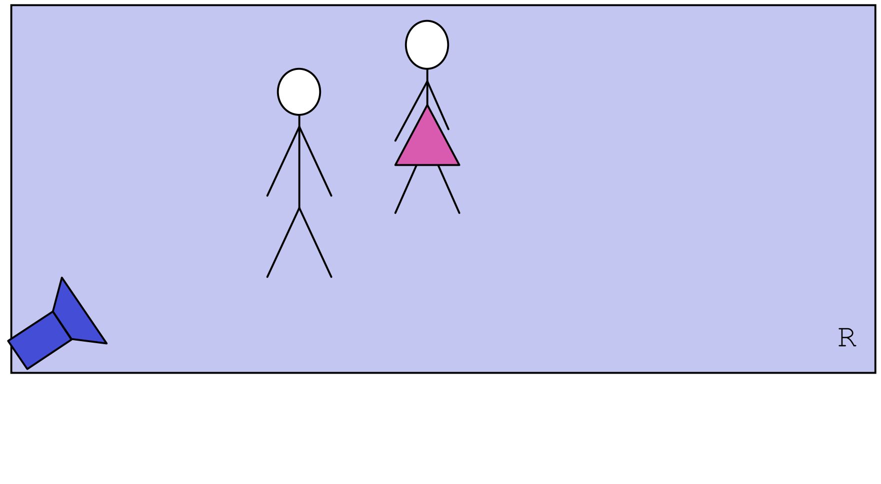
If we add new event types, we need to guarantee that these new types don't interact semantically with the existing types

<enter(A,R) … enter(B,R)>
or
<enter(B,R) … enter(A,R)>

Difficult or impossible to ensure in an open, extensible framework

This is a matter of *semantics*, not of *syntax*, so ontologies don't help all that much

# Correspondence



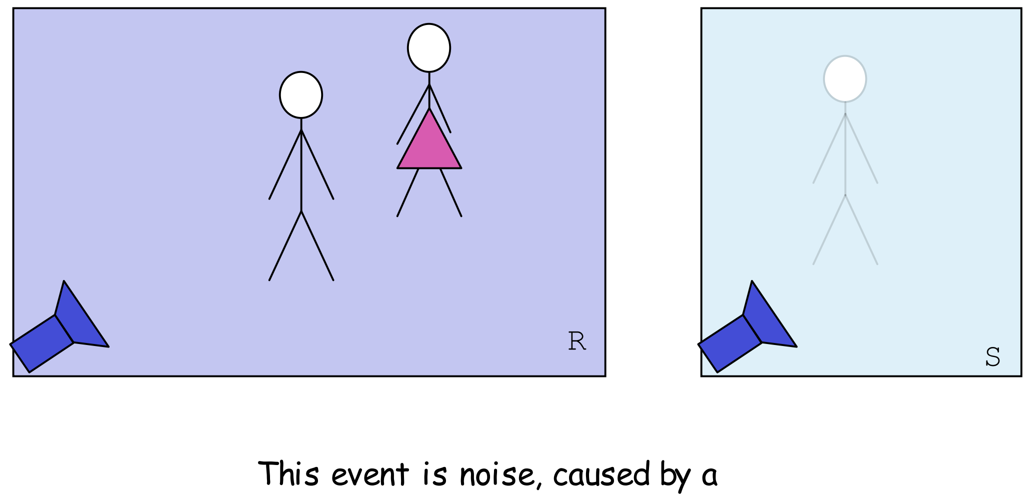

```
enter(A,R)
enter(A,R)
enter(B,R)
```

Does the enter(B,R) event bind to the first enter(A,R) event, or the second? Does it matter?

What happens if the sensors are noisy, or repetitive, or both?

- May leave a “dangling” event that screws-up later processing
- Need a more complex event algebra at the least

# More noise




`enter(A,R)`
`enter(A,S)`
`enter(B,R)`

This event is noise, caused by a fault in the sensor in `S` – but how do we know that?

Most location systems suffer from this problem to some extent

- Or the related problem of missing observations, *i.e.* RFID in a crowded space

# Not too certain

The problem is that we're treating an event as a *fact* when it is actually only *evidence of a fact*

- In a process algebra or traditional distributed system, these two are one and the same
- …but in a pervasive system with real-world sensing, some of the events come from inferencing and/or noisy sensors
- …so the techniques we've used before don't work in quite so obvious a way

Hard to solve these issues *within* the event system itself

- Relies on information *outside* the event stream
- Some systems keep event and do more refined end-point processing, which places the burden on the application programmer

# A hybrid model

Many pervasive systems can be stated in terms of rules, in some sense

- "While this, behave like that"
- "When this happened, do that"

The former is more stable than the latter and so might be preferred

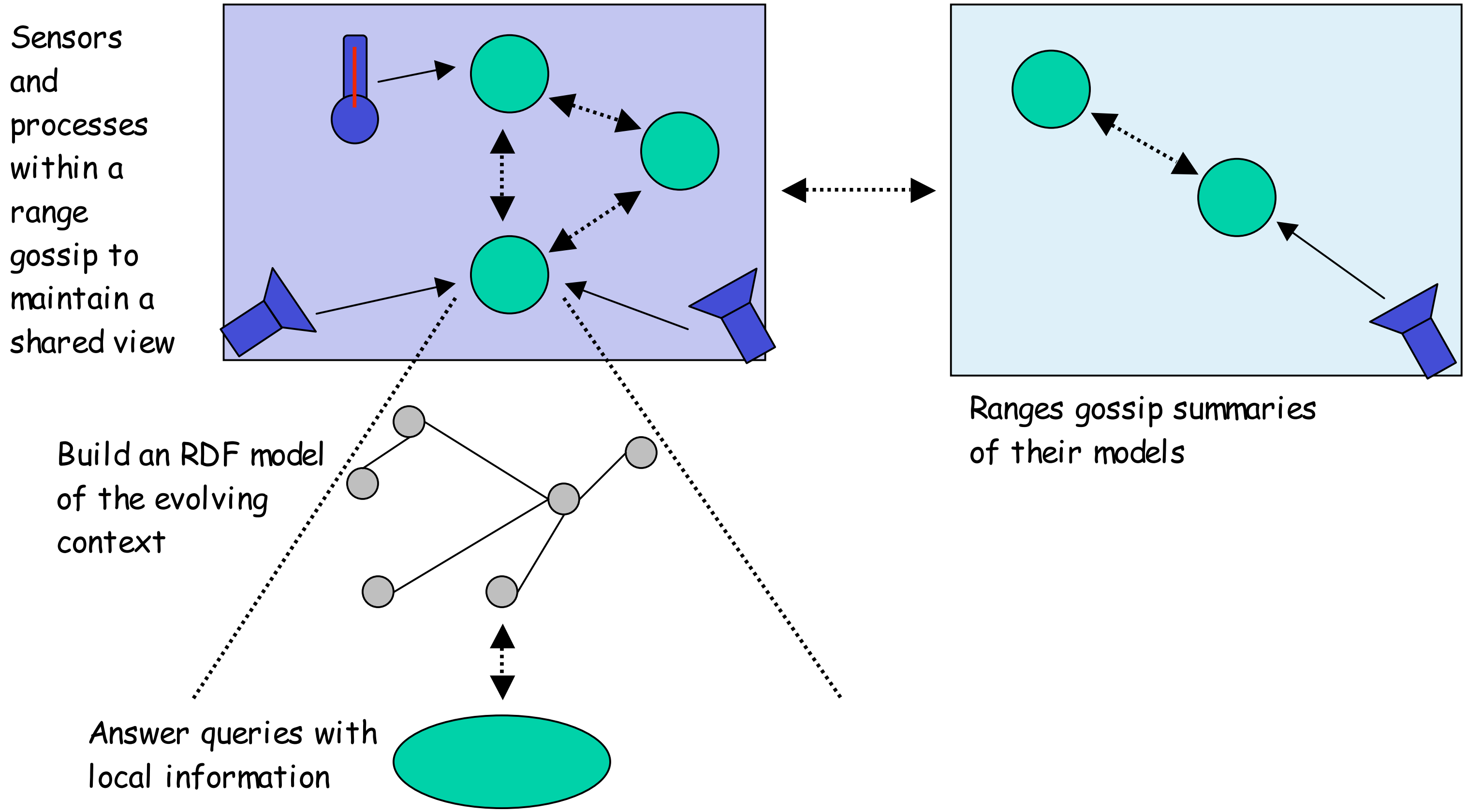
We'd like to *program* in terms of rules, but might want to *implement* in terms of events

- The programmaing advantages of rule-based systems
- The scalability of event-based systems

We've started working on a system that hybridises knowledge-based systems with events and contextors

- Construct – event infrastructure
- Concept – reasoning infrastructure

# General approach




More details on Construct can be found in Stevenson *et al*, Towards a reliable wide-area infrastructure for context-based self-management of communications, WAC 2005

# (Hypothesised) benefits

## Architectural drivers

- Use events where they're good – for scalable exchange of small packets of data
- Use RDF where it's good – for representing a complex web of information and knowledge

## Apply information fusion in the model

- Program in terms of predicates ("A is in R"), *not* in terms of the low-level, possibly faulty events that led to that conclusion
- Evidence theories of different kinds
- Side benefit is that the *same* knowledge arrived at by *different* inferential chains can be used transparently

Keep the confidence intervals in the model too

Not tying applications to individual sensors makes for better robustness – don't care (so much) if a sensor fails, if there's a back-up source of the knowledge we need

# Current work

Construct is working in lab conditions; Concept is under development

- Java, XML, RDF, SOAP, Jena, …

Hybridisation seems to offer certain benefits, although we can't properly quantify them yet

Previous work suggests a quite significant increase in noise tolerance

Active research questions

- How efficient is using contextual knowledge as a stabiliser?
- How much generic knowledge fusion can we do?
- How does gossiping behave on a network?
- How does the architecture interact with security concerns?
- What programming models work best?