Programming context-aware systems

1: Modelling and manipulating

Simon Dobson
simon.dobson@st-andrews.ac.uk
http://www.simondobson.org
Why are we here?

- Placing computers into the real world
  - Pervasive computing, mobile computing, sensor networks, …
  - Sensor-driven, responsive to environment
- How should we program services that make use of their context?

- My aim
  - Consider how to represent context of systems
  - How to use context to drive behaviour
The problem

- Changing use of computers
  - Not getting full attention
  - Disconnected from users/managers
- Services offered need to respond more directly to “what's going on” around them
  - Little/no/different user interface
  - Autonomous action
Use the environment

• Represent the environment in the machine
  • Use it as a knowledge base/user interface/…
• Sensor-driven systems
  • Sense its environment and uses
  • Respond directly to them
  • Less (or no) human-in-the-loop
These lectures

- Part 1: modelling and manipulating
  - Context-aware and sensor-driven systems
  - Characteristics
  - Design approaches for context models
- Part 2: reasoning and maintaining
  - Making decisions
  - Example from pervasive computing
  - Program architectures
The key to a pervasive system is building and maintaining a model of the context in which the system finds itself.

Context ties information from diverse sources together:
- An extensible view of the world
- Drive and adapt functions
- Subsumes many base techniques and technologies

Core requirements

- What’s going on
  - Watch the user’s every move
  - Change with time
  - Model of the environment
- How’s it done
  - Devices, noise, use
  - Constraints on what can be attempted
  - Related to information and functions
- What’s being tried
  - Extract default cases
  - Model of users’ tasks
What is context?

- **Information** about the **environment** a system operates in, **understood symbolically**
  
- That’s a very broad definition – and that’s exactly what we want
  
- Still a lot of structure and a lot of theory – but it’s a different and more open view of the way we build systems

- Not simply raw data, but processed information about what’s going on
- Might include physical, logical and digital "surroundings" of activity
- Made explicit and represented in a way we can reason over, not hidden within code
Example: location

• Surprisingly (or perhaps not) subtle domain

Co-ordinates and named spaces
  • “At 55deg3minN, 3deg45minW”
  • “In A1.15”
  • “In Willard’s office”
  • “In his car”

By negation
  • “Not …”

Unknown
  • “No idea”

Functional spaces
  • “In a conference room”
  • “In his office”
  • “In Willard’s office”
  • “In his car”

Non-located task
  • “Out/on holiday”

Relative
  • “With Willard”

Temporal
  • “At 1000 he will be…”
  • “At 0800 he was…”

Spatial
  • “Within 250m of…”
  • “Between … and …”
  • Either at … or … or …”

Proxy
  • “His badge was last seen at …”

Located task
  • “Meeting Willard”

Default
  • “At this time he is often/usually at …”

Another example

• Most new smartphones are full of sensors
  • Light (so they can control the backlight)
  • Ambient noise (to control their volume)
  • GPS (to know where they are)
  • Tilt (to know which way up they are)
• Basically sensor platforms that make phone calls as well
  • ...and great for games

For an example, try Teeter
http://www.youtube.com/watch?v=l2o_YKIS6Ns
Sensor and sense-ability

- My particular interest, and the most exciting new frontier
  - Active data collection
  - Computing and communications
  - Tiny, low-power
  - Network them together to get capabilities
What this gives us – reach

- Embed computing into the real world, close to the phenomena of interest
  - Detailed, long-term collection
  - Work in hostile or unpleasant environments for long periods
  - A viable alternative to graduate students...

- Data capture is *active*
  - Change observations over time
  - Look for events, rather than just data
Why this is important – 1

• Sensing and context lie at the heart of all the challenges facing the 21st century world
  • Climate change: what's actually happening?
  • Terrorism: what are people really doing?
  • Finance: how is money actually flowing

The solutions to all of these problems start with data, collected accurately and robustly from the real world.
Why this is important – 2

- Many systems can't be managed directly by humans
  - Too fast
  - Too complex
  - Too far away
  - Too intrusive
  - Too much fun
- In that case we need to have computers collecting data and responding to it
  - Clearly has major implications
Computers, don't you hate them?

- We think of computers as things like this:

- But actually a lot of them are like this:

- Embedded computers that can be placed close to the things we want to study
Where are the computers?

- Something like 95% of the world’s microprocessors aren’t in what we’d identify as “computers”
  - “White goods” – dish washers, …
  - “Brown goods” – TVs, DVD players, …
  - Cars – engine management, security, …
  - Sales – radio tags, …
- **Invisible computers**
  - No user interface
  - No explicitly user-visible function
What are they doing?

- Embedded microcontrollers

- Closely linked to some function
  - Sense some facet of their environment
  - Perform some action
  - Very specific – don’t typically re-program the microwave...
In the mid-80s I attended a conference on microprocessor design. One of the speakers stood up and predicted that the world demand for microprocessors could exceed several million units a year by 2000. He was immediately attacked from the floor: one person shouted “Millions?! – what are we going to have, a computer in every doorknob?”

Ten years later I went back to the same hotel I’d stayed in for that conference. Instead of a key, they gave me a plastic card. I swiped the card in the lock to get into the room.

There was a computer in every doorknob.

Danny Hillis

Hillis founded the supercomputing company Think Machines, and (with Richard Feynman) worked out why spaghetti always breaks into three pieces when snapped from the ends.
Sensor-driven systems

- Move beyond *collection* of data and to *responding* to observations or changes
  - A clinical thermometer just collects temperatures; a doctor decides what they mean
  - Means the computer must in some senses “understand” what the readings mean
- Make changes based on this “understanding”
  - Virtual: sample more rapidly
  - Physical: increase the drug dose

Get it wrong, you don't get the quality of data you wanted

Get it wrong, the patient dies
In theory, there is no difference between theory and practice. But, in practice, there is.

Jan L.A. van de Snepscheut
Characteristics

- Not all sensors are created equal
  - Accuracy – how often is a value wrong?
  - Precision – how close to the “real” value?
  - Frequency – how often is a sample taken?
  - Reliability – how often does it fail completely?
- These differences can be important
  - An inaccurate sensor can be misleading
  - An imprecise sensor may not provide enough information
  - Infrequent measurement can lead to inconsistency
  - Failure can be annoying, or can be catastrophic
Characteristics – accuracy

- Consider a location sensor, such as a camera watching a particular area for movement
  - Camera says “yes” when there's none
  - Camera says “no” when there is
    
    Movement may be reported when there's no-one around, or be missed when there actually is

- False positives and false negatives
  - “Noise” that masks what's actually happening

We're using location just for illustration - the same holds for most sensors
Characteristics – precision

- The camera isn't reporting where someone is exactly, only within bounds
  - May be more or less precise

- Person may be anywhere within the sensor “footprint”

- Don't assume more precision than we have
  - How far apart are these two people?
Characteristics – frequency

• The real world is constantly changing
  • Taking discrete samples at discrete intervals

A “ghost” may remain when the real value changes

Where is the person now, since the last observation?
Effects

• In most applications you can assume that the information you’ve got is accurate and timely
  • Or least if it isn’t it’s someone else’s fault
• However, sensor information is inherently imprecise, inaccurate and untimely
  • Inaccurate – if you react to every event, you’ll react to the noise too
  • Imprecise – you may mis-interpret the information
  • Untimely – you may work on a old state when the true state is different
Principles

• Most sensors work by taking a physical effect and turning it into data

• The limits of sensing are therefore the limits of physics
  • All sensing is noisy and lossy, and uses energy
  • Some things just can’t be sensed directly

• A partial picture of reality
  • More and/or different sensors
  • Background information
  • The partial picture may be misleading

The truth often isn’t just the sum of the parts
Trade-offs

● Size
  ● Larger sensor often more accurate and precise
  ● ...and often too big to fit into what you’re building
  ● ...and a smaller one might not be good enough

● Power
  ● A precise and accurate sensor may be just the job
  ● ...but may drain batteries too fast
  ● ...and a low-power one might not be good enough
Virtual sensors

- Increasing volumes of data online
- We can treat this as a domain into which we deploy “sensors”
  - Query the web rather than sense something directly
  - Human information, weather data, …
  - Treat all information as being sensed
    - Model imprecision, inaccuracy, timeliness, …
New modalities

- On-line sources let us sense things we might not otherwise detect
- Good example is interaction networks
  - Who's emailing whom?
  - Visits to web sites
  - Sense the “strength of attraction” between individuals, sites, products, ideas, ...
- Combine with real-world observations
  - Terrorist networks, social groups, ...
Sensor fusion

Since all sensed data is poor (to some degree), how do we work with it?

Fuse data from different sources

- Diary says he should be here
- Camera sees him here
- Cell towers see his phone here
  - but he's got a really average face
  - but that's only got a precision of 100m
  - and he might have had his phone stolen

...but he doesn't keep it completely up to date
Inference

- Several approaches
  - Pick the thing the sensors mostly agree on
  - Combine evidence derived from the different sources
  - Use more sophisticated models to support the reasoning process

Model the process we expect to see, use sensor information to confirm how it progresses

Various mathematics to support this: we'll come back to it later
The context-aware systems loop

1. Observe the real-world system
2. Use the data to populate and maintain a model of that system in a computer
3. Decide on actions to take based on the model and any explicit interactions
4. Take those actions, thereby affecting the system being observed

SENSE

ANALYSE

ACT

DECIDE
Weigh the clothes.

We know that wet clothes are heavier than dry ones, and we know how heavy the clothes were when they were put in the machine.

If the clothes are getting dryer, we should slow the spin speed; otherwise, we should speed up or keep going as we are.

“These clothes are still x% wet”

SENSE

Weigh the clothes

ACT

Change the spin speed

DECIDE
From a systems perspective

We don't know how wet the washing is, but we can measure how heavy it is...

...which should make them dryer... I mean lighter...

...and we can use this to make some guess as to how fast we should spin the clothes...
Implications

- Notice what we're doing here: we're using one thing (that we can measure) as a proxy for another thing (what we can't)
  - We know (or assume) that we understand the relationship between the former and the latter
- There's also a lot of uncertainty about the values we're working on
  - Was the washing really dry when it went in?
  - How does wetness relate to weight? Does it depend on the fabric? What else should we care about?
Where's the user in this?

- May not be one
  - Purely sensor- and model-driven

- If we *have* a user (and a user interface)
  - Consider the instructions as part of context
  - May be very definite and certain: user set the dial to “wash and spin”
  - May be less so: user waved in the direction of a particular light and said “on”

It may be worth handling user actions differently, and change their exact effects according to sensed context.

Or it might be worth treating everything as context, and run behaviour based on a model driven from this broad context.

We'll come back to this later when we consider how to structure context-aware systems.
In any event we want a context model

- Capture and maintain a view of the external world or the system being controlled
- Abstract-away from the details
- Capture the essential elements we need to make the decisions we need
- Place in one place away from the functional logic

SENSE

DECIDE

“These clothes are still x% wet”

Why? What happens if we entwine the context and the main system logic?
Things we want in a model

- Easily manipulated
  - Cheap to add and remove facts
  - Low overhead in keeping the model up-to-date
- Easily extended
  - Cheap and simple to add new types of information
  - No ambiguity in what we mean as things grow
- Scalable
  - As a consequence of the other two
- Efficiently searched

Representing context

• Options
  • As objects
  • As clauses
  • As XML
  • As graphs
  • As some combination of these

Make use of accepted tools and practices for object-oriented analysis and design

Simple, quick, long experience back to the early AI and Lisp days

Access the tools of the semantic web, designed from-the-ground-up for interoperability

Lots of structure, very general

• Remember what we’re looking for: extensibility, ease of use, ease of querying

• Which would you choose?
Object models

Standard approach, familiar to programmers

- Identify the objects in the domain, their attributes and relationships
- Suitable for mapping into an object-oriented implementation

Can we change the system easily as we add sensors, add new functions, etc?
The earliest knowledge-based systems used clause form derived from the Lisp programming language:

- Facts stored as simple, regular strings
- Usually stored in a plain text file
- Lots of knowledge already available

See for example the MIT Media Lab’s “common sense” database at http://openmind.media.mit.edu
A hybrid between objects and clauses

- Define a document format for the data to be exchanged
- Can use namespaces and links to connect fragments

- Need to get agreement on the format of the document, and use it collectively across sensors and applications

Will this work?
We can use the nodes of a graph for facts and the arcs as (binary) relationships between them.

- Arcs are typically called *predicates* or *relationships*.
- The set of arcs intersecting a node tells us the information we know about that fact or entity.
Graphs as knowledge – 1

• How do we use graphs to represent knowledge?

A “key” from which to hang the different facts

Is this too complicated? How do we know we’ll have the information we need, and it’ll all be appropriately typed?
Things to note

- Scaling and heterogeneous knowledge
- Agreement – what predicates “mean”
- Structure – how predicates relate
- Plurality – duplication
- Confusion – the same predicates could appear for different things
- Directedness – can make for awkward encoding
The trade-offs

- **Objects**
  - ✔ Familiar, good structuring, easily implemented
  - ✗ Premature commitment

- **Clauses**
  - ✔ Historically shown to be a good idea
  - ✗ Text files a bit unwieldy, looks archaic

- **XML and the semantic web**
  - ✔ The wave of the future, allegedly
  - ✗ Schemata and angle-brackets everywhere

- **Graphs**
  - ✔ Lots of theory, well-understood
  - ✗ Quite tricky to represent cleanly

Address the nasty issues while retaining access to the good bits?
An approach: RDF and OWL

- Two W3C standards
  - Resource Description Framework – graphs of knowledge
  - Web Ontology Language – structure of predicates and knowledge graphs
- Take the benefits of graphs, address the limitations
  - Agreement
  - Structure
  - Plurality
  - Confusion

Publish and re-use ontologies, compose multiple ontologies in the same knowledge base
Ontologies describe sets of predicates and their constraints
Re-use facts and structure
URIs and XML namespaces make references globally unique

Anyone know why this is called "OWL" not "WOL"? Neither do I...

...and use a richer query language
Consistency problems galore
Get used to it

Scaling
Directedness
Aside: the Semantic Web

• A framework for making data available in machine-readable and processable form
  • Also linked data

• A collection of W3C standards
  • XML, RDF, OWL
  • Also provenance, sensor descriptions, …

• Loved and hated in equal measure
  • Good ideas badly supported?
  • Encourage integration across domains
A simple fact in RDF

Subject: http://www.cs.st-andrews.ac.uk/
Predicate: http://rdf.cs.st-andrews.ac.uk/xml/demo.html#about
Object: School of Computer Science
A cluster of facts

- Given a common subject we can build a cluster of facts using nested predicate elements

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://rdf.cs.st-andrews.ac.uk/xml/demo.html#">
  <rdf:Description about="http://www.cs.st-andrews.ac.uk/">
    <s:about>School of Computer Science</s:about>
    <s:author>S. Punter</s:author>
    <s:phone>+44 1334 461234</s:phone>
  </rdf:Description>
</rdf:RDF>
```

As long as we agree what the predicates mean, we can use whichever we want.

Each of these gives rise to a triple with the same subject (inherited from the containing Description element)
Suppose we want to talk about a member of CS and capture their information too

- Not information about CS *per se*

- Build-out the knowledge graph from the CS node

This cluster of facts is related kind of as a unit to the group object.
Rather than having an `about` attribute, the object of this triple is assumed to be the subject of the nested `Description` element.

These predicates are now "about" the subject that immediately contains them.
Analysis

- Is all this XML really necessary?
  - Very verbose, hard to process, too many strings
- Separate the form from the function
  - XML is a transfer format: no sane person represents data inside a program in XML
  - Graph or database internally
  - The graph structure underlying the RDF is very rich and has all the properties we wanted
- Parse using a standard library
- Define and re-use knowledge
Adding more structure

- Clearly RDF lets us describe pretty much any graph we choose
  - Subjects and predicates are URIs
  - Objects can be values, or URIs used as subjects for further knowledge
- Also clearly will get nasty
  - Guarantee the "right sort" of data is at each end of a predicate
- What applications can expect
Structure and typing

- If we *don’t* mandate explicit structure
  - Applications have to check
  - Expensive and error-prone
  - Requires all applications understand all the details of the context model

- Think of JavaScript *vs* Java
  - Safety requires checking, not assuming
  - Progressively more difficult as system grows
  - In terms of data, but more so in terms of knowledge

A typical context model will use a lot of different namespaces to represent different collections of knowledge
Structuring knowledge

- **OWL** is basically a way of constraining the structure of an RDF graph
  - Classify subjects
  - What predicates they can be the subject of
  - What sort of data predicates relate

- **OWL** is a description logic
  - Decidable fragment of first-order logic
  - Assume an open world
  - OWL Lite, OWL DL, OWL Full

OWL is to RDF what XML Schemata are to XML—a way to say what elements can contain. (And the two are closely related.)

Fully reflective meta-modelling,

Limited cardinalities

Disjointness of classes

Classification and structure – 1

- Primates
- Porcine
- Equine
- Mammals
- Swimmers
- Whales

This class is made up of the union of many others.

This class is the intersection of "mammals" and "things that swim".

This class isn't orthogonal to the others, and can overlap and contain classes in complex way.
Define classes by union or intersection of others

Define a sub-class by restriction on a property: “the animals that swim”

This example requires OWL DL to build the unions and intersections

Something is in this class if it has this property, all other things being equal
Some modelling decisions

- Is chimpanzee a class, or an individual?
- Probably an individual: this ontology consists of classes of animals and individual species
Analysis

- This looks like a mess – and to some extent, it is
  - Namespaces seem confused
  - A lot of syntax for some very simple ideas from set theory
- Again, separate form from function
  - Describe the form of graphs
  - Compose them together, secure in the knowledge that they'll not be subject to confusion or capture
  - Computers find this easy to work with – even if programmers don't

To define an OWL ontology you basically need RDF, RDF Schemata, XML Schemata, OWL itself – and then the namespace whose structure you're defining.

Stevenson and Dobson. Sapphire: Generating Java runtime artefacts from OWL ontologies. ODISE 2011.
The story so far

- Context: symbolic understanding of environment
- Context models: representing knowledge
- RDF: graph model encoded in XML
- OWL: structure of model in terms of logic

This lets us represent the context and manipulate it using standard tools
Semantics

- Context is still low-level
  - Sensor data, diary entries, …
  - Not the level we like to program systems
- What we still need is a semantic layer
  - What does a particular context “mean”?
Situations

- The semantic interpretation of a context is known as a *situation*
  - What the context means
- The process of going from context to situation is called *situation recognition*
  - Sometimes called *activity recognition* for simple cases such as recognising that someone is walking
  - Lots of different contexts may be interpreted as instances of the same kind of situation
  - Don't care how we worked out Simon was in his office: the fact is enough
• Some context-aware components
  • Separate concerns
  • Model, collect, classify
  • Drive or condition behaviour based on context and interpretation

Diagram:
- Context model
- Ontology
- Current situation
- Conditions
- Behaviour
- Situations

Arrows:
- Populates
- Parameterises
- Recognises
- Informs
What's missing?

• We've said nothing about several aspects
  • Uncertainty: we know sensors are imperfect
  • Recognition: how does that happen?
  • How do the elements relate to the rest of the system?

• This is what we'll deal with next
  • Uncertain context
  • Situation recognition
  • Overall system architecture