Programming context-aware systems

2: Reasoning and maintaining

Simon Dobson
simon.dobson@st-andrews.ac.uk
http://www.simondobson.org
Overview

- Context is hard to represent
  - Openness and extensibility
  - Uncertainty
- Given a representation, we have to reason with the model
  - Make decisions in the face of uncertainty
- Also keep it up to date and usable
  - Track changes, and spot them when they happen
Review: representing context

- Context
  - Environment
  - Sensing
- Situation
  - Semantics
  - Interpretation

- Situation recognition
  - Map context to situation
The main issues

- **Reasoning**
  - Sensor data is noisy, contradictory and incomplete
  - How do we represent this properly?
  - How do we make decisions?
  - The best decisions we can make

- **Maintenance**
  - Changes in the environment → changes in model
  - How can we pick these change up effectively?
  - Integrate into the rest of the system cleanly
Simple ways: predicates

- Specify a predicate that determines an action for a given constellation of facts

Context:
- Simon is in his office
- The time is 1001

Recognition:
- In office \land \text{Meeting scheduled} \land \text{Meeting time}

Situation:
- Simon is in a meeting with Graeme
- Characterised by a simple predicate based on the facts from the context model

Simon has a meeting with Graeme 1000-1100
Critique

- Only have \textit{evidence of fact}
  - Noise, reality
  - May be regions over which we infer, not points
- Therefore we only have \textit{qualified recognition}
  - Evidence comes with probabilities
  - Leading to a conclusion with a given probability
Getting the bounds

- Exploratory specification of predicates
- Visualise how system would respond
- Ranges of parameters for which inference gets the “right” result


http://www.situvis.com
What we need is a structured way of performing situation recognition in a probability-preserving way

- Don't generate false sense of security
- Maintain multiple possibilities simultaneously
- Change conclusions as evidence (context) changes

Literally dozens of ways to do this

- Pick out decision theory with Bayesian probability

Simple alternatives

- The kind of imprecision is when you have a set of alternative facts, possibly with "weights" of some kind.

An RDF "alternative" collection - each element is a possibility.

Each element is a structured value with a location and a weight - for example the number of times we've observed the person in that location in the past.
Critique

- *Very simple*
  - How are the weights arrived at?
  - How do they change?
- This is an ideal approach for simple cases
  - Low overhead, easy to manipulate
  - Easy to explain, at least until someone asks difficult questions…
- As things get more complicated it becomes less acceptable
More principled

- Use probability
  - Convert accuracy/imprecision/untimeliness into a measure of a fact being true
  - Apply uncertain reasoning
  - Decide what choices are justified
- Some necessary maths
  - Conditional probability
  - Bayesian probability
Conditional probability – 1

• Suppose we have two events – crashed and drunk – with the expected meanings…
  • The probability of having a crashed car might be 0.02; that of being drunk 0.1
  • Normal probability theory can answer the question of how likely it is to have a crashed car and be drunk:

\[
P(\text{crashed} \land \text{drunk}) = P(\text{crashed}) \times P(\text{drunk})
\]

\[
= 0.02 \times 0.1
\]

\[
= 0.002
\]

This is an expression of the probability that a person selected at random from the population will be both crashed and drunk.
Conditional probability – 2

• How likely is it to have a crashed car given that we already know that you’re drunk?

  *Conditional probability* – changing the likelihood of something given supporting evidence

• Reduce the “selection space”: the crashed among all drunk people, not all people

This area represents the people who are drunk and crashed

This area is the proportion of people who have crashed

The full box represents all the people we’re studying
Conditional probability – 3

• Definitions
  • $P(\text{crashed}) = \frac{|\text{crashed}|}{|\text{people}|}$
  • $P(\text{crashed} \mid \text{drunk})$
    \[= \frac{|\text{crashed} \land \text{drunk}|}{|\text{drunk}|}\]
  • $P(a \mid b)$ is read as “the probability of a given that we already know b”
  • Note that
    $P(\text{crashed} \mid \text{people})$
    \[= \frac{|\text{crashed} \land \text{people}|}{|\text{people}|}\n    = \frac{|\text{crashed}|}{|\text{people}|}\n    = P(\text{crashed})$

Sometimes $P(a)$ is called a “prior” probability and $P(a \mid b)$ a “posterior” probability.

The reason is that the intersection doesn't remove anybody - anyone who crashes is a person.
Not probability

• Normally we assume that we have complete knowledge: we know how likely something is and how unlikely it is, exactly
  \[ P(\text{not } d) = 1 - P(d) \]

• Conditional probability works the same way
  \[ P(\text{not } s|d) = 1 - P(s|d) \]

• However, it is not the case that
  \[ P(A \mid \text{not } B) = 1 - P(A \mid B) \]
Why not?

- Consider the case of a test for a disease
  - $A = \text{“Person A has the disease”}$
  - $B = \text{“The test says that person A has the disease”}$
- But $B$ may not (will not) be 100% accurate
  - In some cases $A$ will be true and $B$ will be true
  - ...but $A$ might be true \textit{despite} $B$ being false
  - ...and also $B$ being true and $A$ being false
- So $P(A \mid B)$ and $P(A \mid \text{not } B)$ are actually \textit{different} observations that support $A$ in different ways, depending on the details of the test accuracy
False negatives and positives

- Suppose the test is 99% accurate in detecting the disease but also yields false positives 5% of the time
- Also suppose the prior probability of having the disease is 0.5%

\[
\begin{align*}
P(A) &= 0.005 \\
P(\neg A) &= 1 - P(A) = 0.995 \\
P(B|A) &= 0.99 \\
P(\neg B|A) &= 1 - P(B|A) = 0.01 \\
P(B|\neg A) &= 0.05 \\
P(\neg B|\neg A) &= 1 - P(B|\neg A) = 0.95
\end{align*}
\]
So far so good…

• ...except that we almost never know the sizes of the sets with any real accuracy
  • How many drunks actually have crashes?
• However, we may know *some* of the conditional probabilities quite well
  • e.g. 70% of crashes involve drunks
• So we can turn things around
  • Use combinations of evidence using any known conditional probabilities
  • Build up evidence to support a hypothesis, and still stay reasonably scientific
Bayes’ theorem

The key to performing this turn-around is Bayes’ theorem

\[ P(d|s) = \frac{P(d) \times P(s|d)}{P(s)} \]

The probability of crashing given that you’re drunk

The probability of being drunk given that you’ve crashed

We still have some requirements for knowing these “global” numbers, which remains a problem


Bayes was an English non-conformist minister with an interest in “natural philosophy”
For example

• $P(\text{crashed}) = 0.02$
  $P(\text{drunk}) = 0.1$

• $P(\text{crashed} \land \text{drunk})$
  $\quad = P(\text{crashed}) \times P(\text{drunk})$
  $\quad = 0.002$

• $P(\text{drunk} \mid \text{crashed}) = 0.7$

• $P(\text{crashed} \mid \text{drunk})$
  $\quad = \frac{(P(\text{crashed}) \times P(\text{drunk} \mid \text{crashed}))}{P(\text{drunk})}$
  $\quad = \frac{(0.02 \times 0.7)}{0.1}$
  $\quad = 0.14$
Bayes’ theorem does two important things:

- It lets us “move around” what we know about risks and inferences
- It lets us combine evidence rather than relying on random sampling

These are vital for making decisions

- Reasoning with incomplete information
- “Building a case” over time
- Without this sort of mathematics it’s hard to see how one could make decisions with even of pretence of science
Representing Bayesian values

• How do we use Bayes’ theorem in practice?
• We’ll stick to the one-variable case for simplicity

• We need to hold four things
  • The two prior probabilities
  • The other posterior probability
  • The posterior probability we’re looking for

More complex cases lead to Bayesian networks, that would need a more thorough treatment than is justified here
Representing a Bayesian value

- We’ll go back to the probability of being a crashed drunk again

\[ P(\text{drunk}|\text{crashed}) = \frac{P(\text{drunk}) \times P(\text{crashed}|\text{drunk})}{P(\text{crashed})} \]

So we can represent this as

Right?
Comments?

• Is this a good representation? Why (not)?
  ✓ Clear and simple
  ✓ Direct route to the information of interest
  ✗ “Hard-wired” the predicate
  ✗ Don’t expose the other elements of the computation
  ✗ Can’t share them
  ✗ Not obvious how to do updates

• An adequate representation, but perhaps not good
Still got a direct link to the interesting value

Still hard-wiring the predicates – not “generic” for all Bayesian values

Values are identified by relationships, not held independently as facts
What we’ve done here – 1

Separated the *type* from the *instance* in traditional style – the *object* from the *class*

✓ Single shared set of predicates to re-use, easy to implement as a Java object
✓ Can share the same probability in different Bayesian values
✓ May make updates easier
✗ More complicated to build and maintain
What we've done here – 2

- Adopted a *principled* way to manipulate context
  - Take the evidence
  - Associate probabilities to it
  - Manipulate those probabilities using known mathematics to derive *conclusions we can support*
- Lots of other formulations with similar issues
  - Fuzzy logic, Dempster-Shaffer evidence theory, …
  - Important point is keeping track of what you know and with how much confidence

Another way to look at this is that data comes with *provenance*, which we track through the reasoning process.
Application to services

- This is all quite abstract: how does it apply to services?

- For a given sensor suite
  - Record traces of activities you typically see
  - Decide what situations you want to recognise
  - Construct decision-making infrastructure
  - How predictive are the various sensors of the various situations?

What does “typically” mean?
Example: UCD CASL

- Instrument a space with sensors of different modalities
  - Location
  - Identity
  - Keyboard activity
  - Diary
Step 1: Record

- Acquire *ground truth*
  - Record a set of traces
  - Get people to record what they do
  - Mark-up the dataset with what the sensors were actually observing

- Getting *ground truth* is fiercely expensive
  - Persuading people to record themselves
  - Video
  - Mark-up

There is an acute shortage of publicly-available, multi-sensor, marked-up datasets
Step 2: Situations

- What situations do we want to recognise?
  - What services do you want to provide?
- Smart building
  - Busy at desk: reduce distractions
  - Coffee/lunch breaks: peripheral awareness to and of colleagues
  - Reading at desk: very quiet time
  - Formal meeting: keep records, deliver documents
Step 3: Learn to recognise

- Take the training dataset
  - How predictive of each situation is each sensor reading?
  - Taken as a whole, fusing the sensor readings
  - Probability that, given the current readings, a particular situation is in progress
- Then we can maintain a running probability of what's happening by examining the sensor readings as they change
Step 4: Validate – 1

- Either:
  - Collect new datasets and run them against the model we've developed

- Or:
  - Taken $n$ datasets (or days of data)
  - For each $d$ in $n$, eliminate $d$, learn a model, see how well it predicts $d$'s ground truth
  - Can find outlier days

Known as n-fold stratified cross validation
Step 4: Validate – 2

- Situation recognition is about information retrieval
  - Precision: how many situations were correctly recognised?
  - Recall: how many situations that could have been identified were recognised correctly?
  - F-measure:
    \[ F = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \]

Essentially fuses the two measures of quality into one, when all you want is the "goodness" of the approach.
Profile of results

- Different approaches, different qualities

Keyboard activity is a really good predictor.

No sensor that can detect "sitting around reading".
Structuring situations

- Situations have structure
  - "Meeting" vs "meeting with Erica" vs "Group meeting" vs ...
  - Capture this using a lattice relating observations to the situations they are consistent with

\[ \text{Ye et alia. Using situation lattices in sensor analysis. Proc. Percom'09.} \]
• None of the techniques we tried then – and none since, actually – can generate anything like certainty
  • Not sure what's happening
  • Have a level of confidence from the F-measure

• For services
  • Not sure what behaviour to exhibit
  • Be ready to be wrong!
Dynamism

• The foregoing used a dataset of a *trace*
  • A time series of events
• Doing this “live” means *maintaining* the model you've built
  • Update with new readings
  • Add new features, possibly new sensors and situations
• Some of these changes will (should!) trigger actions in the services using the model
Model maintenance

• A context-aware system is highly dynamic, and needs to respond to real-world changes in pretty close to real time
  • Typically it’s changes that provoke actions

• Two issues
  • *Maintaining* the model as a true and accurate representation of what’s going on
  • *Performing actions* in response to situations

Although they're typically not “real-time systems” in the usual sense
Is this maintenance?

• “Ensuring things work properly” or “fixing what’s broken”
  • The model may have been correct…
  • …but it needs to stay correct
• Maintain a view of the world that’s accurate enough for services to work with
  • May not mean completely up-to-date, as we’ll see
  • …and will always include uncertainties
  • …but does mean sufficiently up-to-date
  • …and with the uncertainties staying where you put them

There’s a trade-off between maintaining a “better” model and doing more “useful” work
In the abstract

We're looking for a means of making the context model reflect the environment in a way that satisfies the needs of the applications. There are therefore at least two feedback cycles going on.
Issues to consider

- What changes?
  - Everything? Are some things maintained?
- When do things change?
  - Any time? At fixed times? Will fixed times do?
- How will the model find out about a change?
  - Observe? Get an event? Callbacks?
- Are all changes significant?
  - What changes are actually noise?
- Does a change have consequences?
  - Is it consistent with other information?
Assumptions of stability

- May be possible to reduce change somewhat
  - Make some elements “constant”
  - Don’t allow changes through the environment
- For example, a security application won’t usually discover new users through sensors
  - Configuration issue
  - …which may be handled like a change or may not, but can usually be assumed to be consistent, accurate and timely at least

All the usual constraints of software engineering to do with evolution and change management apply strongly to context-aware systems too – and may perhaps be more difficult to handle because of their openness and dynamism...
Sufficient timeliness

• It’s a myth that all data needs to be completely up-to-date all of the time: many applications can work with older data
  • “At 1000 he was there”
  • “There were three people there last time I looked (which was recently enough for what you want)”
  • “Now you ask, I’ll have a look…”

• Expend time and effort to maintain information that’s better than needed?
Complexity and costs

- Any maintenance activity costs
  - Minimum – add data as it comes in, regardless
  - Maximum – truth-maintenance and inference

- Often need to trade-off the complexity of maintenance against the expected value
  - Timely information may require expensive checks to be performed for every change

- Compromises
  - Let the information get a bit stale
  - Perform periodic housekeeping
  - Hope things stay consistent

In fact a lot of AI techniques apply really well to context models

Needless to say there are lots of complex trade-offs going on here
High-level view of approaches

- The context model
  - Passive – model maintains a consistent, accurate view that applications can query as required
  - Active – changes in the model can also give rise to actions

- Which one to choose depends on the application
  - Passive – when actions are explicitly triggered and just conditioned by their context
  - Active – where actions happen autonomously
Example

- Target tracking
  - Track sensor results for targets
  - Maintain a running estimate of position
- Active
  - Set “hot spots” that trigger action
  - When these are entered/leave/whatever, notify the service

Query when the position is needed, service keeps control

So the model is in some sense responsible for taking actions: some of the service’s logic migrates inside
Polling

- The application queries the model whenever it needs context information
  - Model maintains itself “silently”
  - May be very complex, but quiet
  - Application reacts to stimuli that occur “outside” the context model, for example explicit user actions
  - Model isn’t a complete representation: control occurs “outside” it
  - Application never reacts to contextual changes
External events

Changes give rise to events sent to the application on some channel

- Separation of model and application
- Distribute events through a standard event system
- May filter events sent to each application

- Can be awkward for programmers
  - Gets very complicated in the presence of failure
  - Lots of events to handle, which can be slow and/or confusing
Application registers code to be called automatically when a change occurs

- Essentially an in-process event
- Good for one application, hard/impossible otherwise
- Close to other systems, i.e. AWT/Swing
- Can lead to efficient application code, easier to build and understand
  - Code is attached directly to the point of change
  - Limited complexity of responses to changes
Internal triggers

- Application provides a query and blocks until the model is in a state that satisfies it
  - Queries of arbitrary complexity
  - May be expensive to process changes if implemented naively
  - Tricky to do in a distributed fashion
- Like logic programming
  - Queries block until there is at least one solution
- Easy to get a simple version working

Needs a good understanding of AI and programming languages to be done properly
Maintaining consistency

• If I have a model, and I make a change, what happens?
  • May be inconsistent with other “facts”
  • May alter other values that “depend” on the changed element
  • All above and beyond what happens at the application level

• Partial solutions
  • Store how derived values are computed in the model, re-evaluate when arguments change
  • Run a truth-maintenance algorithm

This is essentially dataflow programming

More AI techniques...
When should we re-compute?

- One good example is when to re-compute a “crisp” value from an uncertain one
  - Situation recognition with one situation

- Two essential options
  - When data comes in – the model reflects the current “best estimate”
  - When an application asks – build the “best estimate” at the time of the request

- The question of when to compute is orthogonal to the one about what to compute

Correspond to forward and backward chaining in inference systems – and with the same trade-offs
Summary – maintaining context

- The point of a context model is to change as the world changes so as to reflect it
  - How timely does the data need to be
  - How will applications use the data
  - How is consistency/truth maintained
  - When do things happen
Querying

- We still need to get data out of the model
  - Lots of RDF and OWL
- Traverse the knowledge graph
  - Translate a question we want answered into the data that answers it
  - The same process might be triggered externally (by a query) or internally (to trigger action)
- Two approaches
  - Querying
  - Reasoning
“SQL for the semantic web”

- Extract data against a query

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
  { ?x foaf:name ?name .
    ?x foaf:mbox ?mbox }
Critique – 1

- Clean and computationally fairly simple
  - Given a well-structured model, anyway....
- Combine multiple RDF namespaces
  - Ignore links you don't understand / care about
  - Familiar to database programmers
Critique – 2

- Treating context as database is a limitation
  - We don't just have a schema, we have an ontology
  - Logical relationships between elements
  - Sub-classes, same-as, different-to, ...

- Having captured this richness, we should be able to make use of it
  - Open-up the model
  - Extensions can be handled cleanly
• Expand a query
  • If we know Lilly is a context-awareness researcher, we can infer she is a student
  • Find more about the data from the ontology
Reasoners

- Reasoners apply ontological reasoning across a model written in OWL to a model represented in RDF
  - Pellet is a well-known example

Limitations

- Those inherent to the use of description logics: pretty much impossible to prove negatives
- Computational expensive

See http://clarkparsia.com/pellet/
Fitting in to the system

Context model

Current situation

Informs

Recognises

Parameterises

Behaviour

Conditions

Populates

Ontology

Query engine

Reasoner

Interprets

Informs
Drawing it all together

- A rich context model is really useful
  - Record all the relevant data
  - Extensible and query-able
  - Maintain a view of what's happening
- Access can take several modes
  - Precise queries
  - Imprecise (reasoning-based) queries
  - Different ways to trigger action
What remain challenging?

- Everything :-)
  - Getting ground truth for testing
  - Describing situations, learning the right probabilities
  - Drift and heterogeneous behaviour
  - Multiple uses of the same space
  - Building applications we can understand and maintain
- What are the right programming abstractions?
Three things to take away

● Context-aware behaviour can be of enormous value when added to services
  ● ...but needs careful analysis and design

● Built around uncertain reasoning
  ● ...which is unfamiliar and challenging

● The semantic web provides an appropriate conceptual framework and a toolset
  ● ...but the syntax and APIs are lousy