What is context-aware computation?
What is context-aware computation?

Everybody knows intuitively

How important is it? within the framework of UbiNet
Context-awareness: Models and Analysis

Matthew Hennessy, U of Sussex

What is *context-aware computation*?

Everybody knows *intuitively*

How important is it? *within the framework of UbiNet*

*context is key in . . . efforts to disperse and enmesh computation into our lives*

Moran and Dourish
Context-aware computation - definitions

- Context is *location, identities of nearby people and objects, and changes to these objects*
  Schilit and Theirmer, in Disseminating ... to Mobile Hosts

- Context refers to *physical and social situation in which computational devices are embedded*
  Moran and Dourish, in Introduction to special issue of HCI on Context-Aware Computing

- Context is *any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is relevant to the interaction between a user and an application, ...*
  Anind Dey et al, in Understanding and Using Context
Context-awareness impinges on . . .

- **hardware:**
  - eg sensor technology for context acquisition

- **networks:**
  - eg distributed infrastructures for supporting automatic separation of application from context acquisition

- **language design:**
  - new language constructs for facilitating construction of context-aware applications

- **software:**
  - eg of management and filtering of context data

- **HCI:**
  - new design and evaluation principles for interface and interaction
Models of context awareness thin on the ground

Indirect: eg via distributed infrastructures designed to host context-aware applications

- The Context Toolkit www.berkeley.edu/ dey/context.html
  - *context widgets* - hide details of context sensing
- MobiComp www.cs.kent.ac.uk/projects/infra/mobicomp
  - *context elements* - objects for providing/managing contextual information

Direct: eg via languages or theories:

- ContextUnity, Roman et al, ETAPS 2004
  - Unity in which variables can be declared *internal* or *exposed*
- ULM, Boudol, ETAPS 2004
  - distributed asynchronous *reactive* programming
Outline of session

- Design abstractions for Context-Awareness, Lopes & Fiadeiro
  - a design language for distributed components

- Context-based simulator for sensor networks in . . ., Huebscher
  - Software simulation framework of sensor/application infrastructures

- Classifying Context Classifications: . . ., Kaenampornpan et al
  - improved system design via better understanding of contexts
Outline of session

- *Design abstractions for Context-Awareness*, Lopes & Fiadeiro
  - a design language for distributed components

- *Context-based simulator for sensor networks in*, Huebscher
  - Software simulation framework of sensor/application infrastructures

- *Classifying Context Classifications*: . . . , Kaenampornpan et al
  - improved system design via better understanding of contexts

- Context-aware *behaviour and reasoning*
  - outline of recent work with colleagues
  - ideas for future directions
Background

Agents act within an ever changing context

- *Behaviour* of agents depend on the agent’s knowledge of the potential of their context

- *Properties* of agents depend on the context’s knowledge of the potential of these agents
Background

Agents act within an ever changing context

- **Behaviour** of agents depend on the agent’s knowledge of the potential of their context
  
  *if A does not know of existence of SERVER then it can not use it*

- **Properties** of agents depend on the context’s knowledge of the potential of these agents
  
  *agents may want to keep certain capabilities private*
A language for expressing processes - picalculus

- processes compute by sending/receiving values along channels
- channels are read once variables

The language:

Read from channel \( u \):
\[ u ? (x) R \]

Write to channel \( u \):
\[ u ! \langle v \rangle \] - values \( v \) are data or channels

Channel creation:
\[ \text{new } n : E \) R \] - with capabilities \( E \)

Parallelism:
\[ R_1 | R_2 \]

Iteration:
\[ * R \]
Example: a server

A primechecking server $\text{PRIMES}$:
- receives a request at port $\text{req}$
  - an integer,
  - a return $\text{address}$
- analyzes the integer
- returns true or false on address

$$\text{PRIMES} \leftarrow *\text{req}?({x}, {y}) \ \text{let} \ a = \text{isprime}({x}) \ \text{in} \ {y}!\langle a \rangle$$
Example: a client

generates a return address reply
sends value 27 and reply to server
awaits answer on return address
prints answer when it comes

\[
\text{CLIENT} \leftarrow (\text{new reply}) \ req!\langle 27, \text{reply}\rangle \\
| \text{reply}?\langle x \rangle \ \text{print!}\langle x \rangle
\]
Does the server work?

Will the client print true or false?
Does the server work?

- Will the client print `true` or `false`?
- Is `SYS` behaviourally equivalent to `SPEC` where

\[
\begin{align*}
SYS & \Leftrightarrow \text{PRIME}S \mid \text{CLIENT} \\
\text{SPEC} & \Leftrightarrow \text{PRIME}S \mid \text{print!}\langle\text{false}\rangle
\end{align*}
\]
Does the server work?

- Will the client print \textit{true} or \textit{false}?
- Is $\textsc{sys}$ behaviourally equivalent to $\textsc{spec}$ where
  
  $\textsc{sys} \leftrightarrow \textsc{primes} \parallel \textsc{client}$
  
  $\textsc{spec} \leftrightarrow \textsc{primes} \parallel \text{print!⟨false⟩}$

- $S1$ behaviourally equivalent to $S2$ means no observer/user can tell them apart, in any context
  - formalise using
    - \textit{bisimulation equivalence}
    - \textit{contextual equivalence}
A bad server

A rogue server BADPRS:

- receives a request at port req
  - an integer,
  - a return address

- always returns true on return address

BADPRS $\leftarrow *req?(x, y) y!\langle\text{true}\rangle$
Does the server work?
Does the server work?

SYS not behaviourally equivalent to SPEC

because

SYS | BADPRS not behaviourally equivalent to SPEC | BADPRS
Does the server work?

SYS not behaviourally equivalent to SPEC

because

SYS | BADPRS not behaviourally equivalent to SPEC | BADPRS

recall:

SYS &lt;&gt; PRIMES | CLIENT

SPEC &lt;&gt; PRIMES | print!⟨false⟩

BADPRS &lt;&gt; *req?(x, y) y!⟨true⟩
The server works
The server works

under certain assumptions about the *working context*
The server works

- under certain assumptions about the *working context*
- assuming there is no other *server* in the *working context*
The server works

- under certain assumptions about the *working context*
- assuming there is no other *server* in the *working context*
- formalised using *visible* v. *invisible* capabilities
  cf *exposed* versus *internal* variables in *Context UNITY*
Roman etaps 2004 invited talk
The server works

- under certain assumptions about the *working context*
- assuming there is no other *server* in the *working context*
- formalised using *visible* v. *invisible* capabilities
  cf *exposed* versus *internal* variables in *Context UNITY*

Roman etaps 2004 invited talk

What is a server?
The server works

under certain assumptions about the \textit{working context}

assuming there is no other \textit{server} in the \textit{working context}

- formalised using \textit{visible} v. \textit{invisible} capabilities
  
  \textit{cf exposed} versus \textit{internal} variables in \textit{Context UNITY}

Roman etaps 2004 invited talk

What is a server?

An agent which has the \textit{capability} to read from request channel \texttt{req}
Contextual behaviour

\[ \mathcal{I} \models P \approx Q \]

$P$ and $Q$ are behaviourally equivalent in all contexts containing knowledge $\mathcal{I}$ of the capabilities/potential of $P$ and $Q$. 
Contextual behaviour

\[ \mathcal{I} \models P \approx Q \]

\(P\) and \(Q\) are behaviourally equivalent in all contexts containing knowledge \(\mathcal{I}\) of the capabilities/potential of \(P\) and \(Q\).

\[ \mathcal{I} \triangleright \text{req}\,(x)T \approx \text{stop} \]
if \(\mathcal{I}\) does not contain the capability to write to req

\[ \mathcal{I} \triangleright (\text{new } s) \text{dist}!\langle s\rangle \mid s?(x)T \approx (\text{new } s) \text{dist}!\langle s\rangle \]
if in \(\mathcal{I}\) dist can not read write capabilities
The server works

\[ \mathcal{I} \models \text{PRIME} \mid \text{CLIENT} \approx \text{PRIME} \mid \text{print!} \langle \text{false} \rangle \]

provided

\( \mathcal{I} \) does not allow reading on \( \text{req} \)
The server works

\[ I \models \text{PRIME} \ | \ \text{CLIENT} \approx \text{PRIME} \ | \ \text{print!}\langle \text{false} \rangle \]

provided

\[ I \text{ does not allow reading on req} \]

recall:

\[
\text{PRIME} \leftrightarrow \ast \text{req?}(x, y) \ \text{let} \ a = \text{isprime}(x) \ \text{in} \ y!\langle a \rangle \\
\text{CLIENT} \leftrightarrow (\text{new reply}) \ \text{req!}\langle 27, \text{reply} \rangle \\
| \ \text{reply?}(x) \ \text{print!}\langle x \rangle
\]
Formalising $\mathcal{I} \models P \approx Q$

Using bisimulations over *actions-in-context*

$$\mathcal{I} \triangleright P \xrightarrow{\mu} \mathcal{I}' \triangleright P'$$

where

- $\mathcal{I}$ is current knowledge of context
- $P$ is current system
- $\mu$ is description of possible interaction with context
- $\mathcal{I}'$ is new knowledge of context after the interaction
- $P'$ is resulting system after the interaction
Actions-in-context: examples

standard action:
\[ \text{req}?(x) \xrightarrow{T} (n)\text{req}?! \xrightarrow{T} \{n/x\} \]
Actions-in-context: examples

- standard action:
  \[
  \text{req}?(x) \xrightarrow{T} (n)\text{req}?n\xrightarrow{T}\{n/x\}
  \]

- action-in-context:
  \[
  \mathcal{I} \triangleright \text{req}?(x) \xrightarrow{T} (n:E)\text{req}?n\xrightarrow{T}\mathcal{I}, \quad n : E \triangleright T\{n/x\}
  \]
  provided allows writing \(n\) on \text{req}
Actions-in-context: examples

- standard action:
  \[ \text{req}\, (x) \xrightarrow{T} \text{req}\, n \xrightarrow{T} T\{n/x\} \]

- action-in-context:
  \[ \text{I} \xrightarrow{\text{req}\, (x)} \text{req}\, (n:E) \xrightarrow{T} \text{req}\, n \xrightarrow{T} \text{I}, \ n : E \xrightarrow{T} T\{n/x\} \]
  provided  allows writing  \( n \)  on  \text{req}

- standard action:
  \[ (\text{new } n : E) \xrightarrow{\text{req}!n} T\{n\} \xrightarrow{T} 0 \]
Actions-in-context: examples

- **standard action:**
  \[ \text{req}\? (x) \ T \ (n)\text{req}\? n \ T \{n/x\} \]

- **action-in-context:**
  \[ \mathcal{I} \triangleright \text{req}\? (x) \ T \ (n:E)\text{req}\? n \ \mathcal{I}, \ n : E \triangleright T \{n/x\} \]
  provided allows writing \( n \) on \( \text{req} \)

- **standard action:**
  \[ (\text{new } n : E) \ \text{req}! \langle n \rangle \ (n)\text{req}! n \ 0 \]

- **action-in-context:**
  \[ \mathcal{I} \triangleright (\text{new } n : E) \ \text{req}! \langle n \rangle \ (n)\text{req}! n \ \mathcal{I}, \ n : \mathcal{R}_{\text{req}} \triangleright 0 \]
  provided \( \mathcal{I} \) allows reading on \( \text{req} \)
Contextual reasoning

Contextual reasoning principle:

1. \( \mathcal{I} \models P \approx Q \)
2. \( R \) conforms to \( \mathcal{I} \)

implies

\[ \mathcal{I} \models P | R \approx Q | R \]
Contextual reasoning

Contextual reasoning principle:

- $\mathcal{I} |= P \approx Q$
- $R$ conforms to $\mathcal{I}$

implies

$$\mathcal{I} |= P \mid R \approx Q \mid R$$

eg if $\mathcal{I}$ is a type environment then $R$ conforms to $\mathcal{I}$ means $R$ can be typed by $\mathcal{I}$
Contextual reasoning - an example

\[ \mathcal{I} \models \text{PRIME} \mid \text{CLIENT}_1 \mid \text{CLIENT}_2 \]

\[ \cong \]

\[ \text{PRIME} \mid \text{print}_1!\langle \text{false} \rangle \mid \text{print}_2!\langle \text{true} \rangle \]

provided \( \mathcal{I} \) does not allow reading on \( \text{req} \)
Contextual reasoning - an example

\[ I \models \text{PRIMES} \mid \text{CLIENT}_1 \mid \text{CLIENT}_2 \]

\[ \simeq \]

\[ \text{PRIMES} \mid \text{print}_1!\langle \text{false} \rangle \mid \text{print}_2!\langle \text{true} \rangle \]

provided \( I \) does not allow reading on \( \text{req} \)

\[ \text{PRIMES} \leftarrow *\text{req}?(x, y) \ \text{let} \ a = \text{isprime}(x) \ \text{in} \ y!\langle a \rangle \]

\[ \text{CLIENT}_1 \leftarrow (\text{new reply}) \text{req}!\langle 27, r \rangle \]

\[ \quad \mid \text{reply}?(x) \text{print}_1!\langle x \rangle \]

\[ \text{CLIENT}_2 \leftarrow (\text{new reply}) \text{req}!\langle 19, r \rangle \]

\[ \quad \mid \text{reply}?(x) \text{print}_2!\langle x \rangle \]
Contextual reasoning - an example

\[ \text{context} I \models \text{PRIMES} \mid \text{CLIENT}_1 \approx \text{PRIMES} \mid \text{print}_1 \langle \text{false} \rangle \]

direct proof
Contextual reasoning - an example

\[ \mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \approx \text{PRIMES} \mid \text{print}_1!\langle\text{false}\rangle \] - a direct proof

\[ \text{CLIENT}_2 \text{ conforms to } \mathcal{I} \]

So \[ \mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \mid \text{CLIENT}_2 \approx \text{PRIMES} \mid \text{print}_1!\langle\text{false}\rangle \mid \text{CLIENT}_2 \]
Contextual reasoning - an example

\[ I \models \text{PRIMES} | \text{CLIENT}_1 \approx \text{PRIMES} | \text{print}_1!\langle \text{false} \rangle \] - a direct proof

\[ \text{CLIENT}_2 \text{ conforms to } I \]
So \[ I \models \text{PRIMES} | \text{CLIENT}_1 | \text{CLIENT}_2 \approx \text{PRIMES} | \text{print}_1!\langle \text{false} \rangle | \text{CLIENT}_2 \]

\[ I \models \text{PRIMES} | \text{CLIENT}_2 \approx \text{PRIMES} | \text{print}_2!\langle \text{true} \rangle \] - a similar direct proof

\[ \text{print}_1!\langle \text{false} \rangle \text{ conforms to } I \]
So \[ I \models \text{PRIMES} | \text{CLIENT}_2 | \text{print}_1!\langle \text{true} \rangle \approx \text{PRIMES} | \text{print}_2!\langle \text{false} \rangle | \text{print}_1!\langle \text{true} \rangle \]
Contextual reasoning - an example

\[ \mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \sim \text{PRIMES} \mid \text{print}_1!\langle\text{false}\rangle \] - a direct proof

\text{CLIENT}_2 \text{ conforms to } \mathcal{I}

So \[ \mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \mid \text{CLIENT}_2 \sim \text{PRIMES} \mid \text{print}_1!\langle\text{false}\rangle \mid \text{CLIENT}_2 \]

\[ \mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_2 \sim \text{PRIMES} \mid \text{print}_2!\langle\text{true}\rangle \] - a similar direct proof

\text{print}_1!\langle\text{false}\rangle \text{ conforms to } \mathcal{I}

So \[ \mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_2 \mid \text{print}_1!\langle\text{true}\rangle \sim \text{PRIMES} \mid \text{print}_2!\langle\text{false}\rangle \mid \text{print}_1!\langle\text{true}\rangle \]

Therefore

\[ \mathcal{I} \models \text{PRIMES} \mid \text{CLIENT}_1 \mid \text{CLIENT}_2 \sim \text{PRIMES} \mid \text{print}_1!\langle\text{false}\rangle \mid \text{print}_2!\langle\text{true}\rangle \]
How general is all this?

\[ \mathcal{I} \models M \approx N \]

- \( M, N \) agents: \( \mathcal{I} \) contains
  - known agents
  - agents rights
  - ...

- \( M, N \) distributed agents: \( \mathcal{I} \) contains
  - known sites
  - state of network - *failed sites, broken links*
  - migration rights
  - ...

UbiNet-Cambridge May 2004 – p.21/2