VPS @ Liverpool

[ongoing work]

Michael Fisher

Department of Computer Science, University of Liverpool, UK

Example

As we walk into a shopping area, our intelligent clothing interacts wirelessly with shops in the area and then with our mobile phone to let us know that our shoes are wearing out and that the best deals nearby are at shops X, Y and Z.

Our PDA, which holds our shopping list, also interacts with our phone to suggest the optimum route to include shoes in our shopping, and with the shops to assess their stock.

At the same time, our PDA interacts with the shopping area's network and finds that one of our friends is also shopping — a text message is sent to the friend's mobile/PDA to coodinate shopping plans and schedule a meeting for coffee at a close location in 15 minutes.

Example (2)

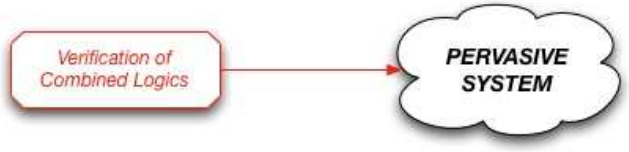
Even in this simple example the components of this pervasive system at least need capabilities to carry out:

- plan synchronisation;
- spatial reasoning and context-awareness;
- planning and scheduling;
- mobility and communication, etc....

And there are many dimensions to formalize:

- security and reliability;
- safety and liveness;
- real-time response, probabilistic behaviour, etc.

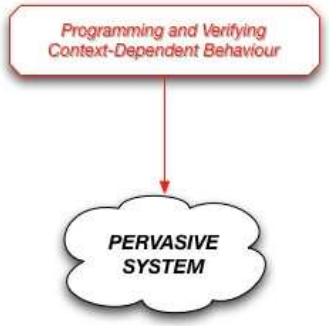
Problem: Multiple Dimensions



Pervasive Systems comprise many different *facets* and we need to describe/specify not just their basic dynamic behaviour, but also

- *real-time* aspects
- *uncertainty* and *environmental models*
- *collaboration* and *cooperation*
- *mobility*, *distribution* and *concurrency*
- *autonomous decision-making*, *etc*...

Problem: Context Dependency

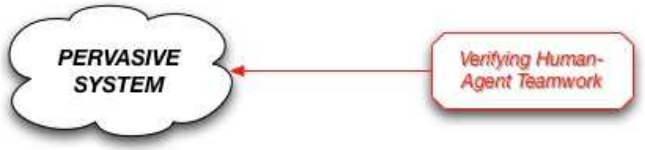


Pervasive Systems are *context-dependent* — behaviour can change *because* of movement between contexts.

Contexts do *not* just concern location. Contexts can be

- locations
- roles or societal norms
- teams or organisational structures
- styles or preferences, *etc*....

Problem: Humans “in the loop”



Pervasive Systems involve

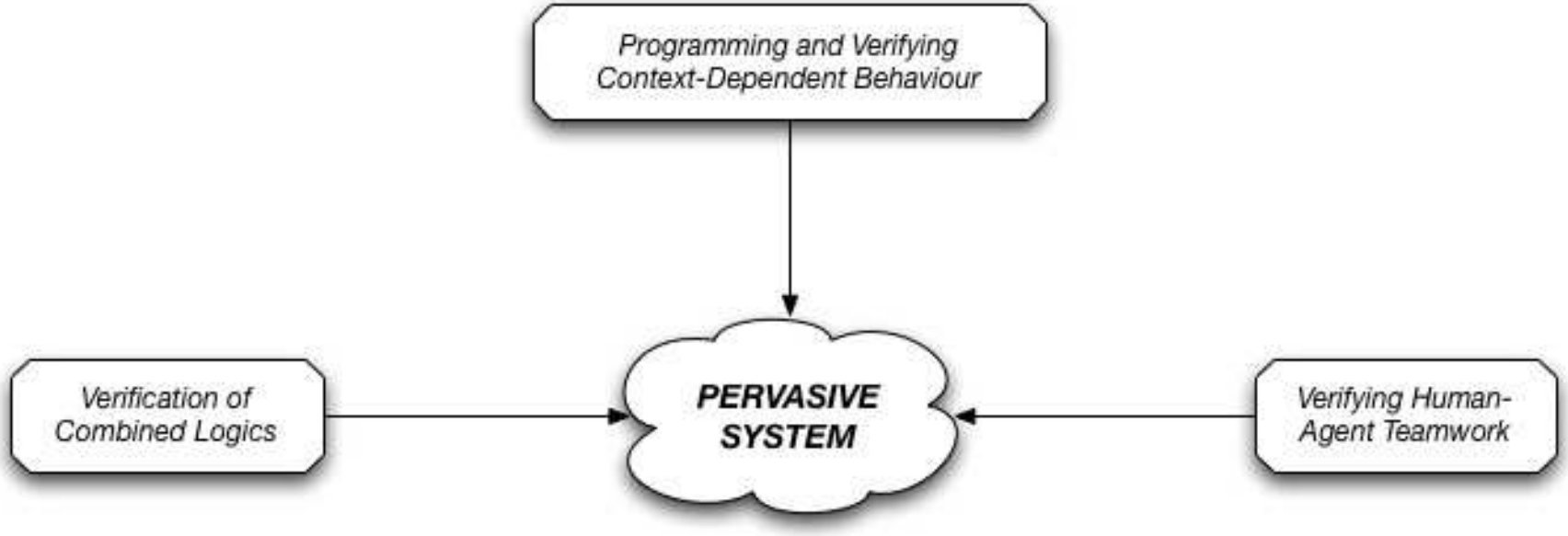
- *humans*, or at least external autonomous entities, and
- a variety of *tools* or *artifacts*

within the system.

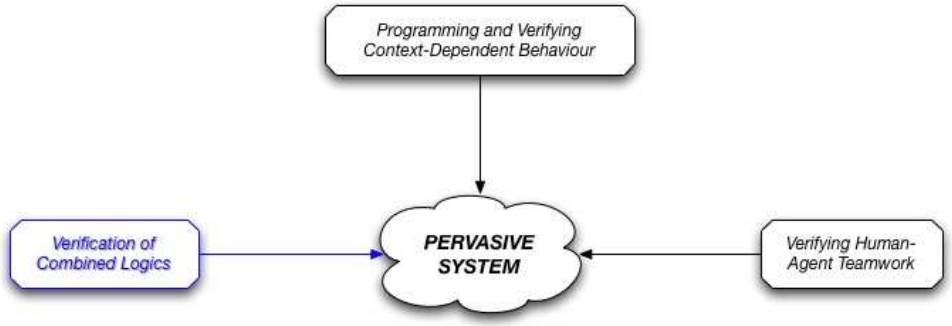
Typically, humans are embedded within pervasive systems.

But how shall we model humans (and other autonomous entities) within our formalisation?

Verifying Pervasive Systems [Our View]



A Plethora of Formal Logics



- dynamic communicating systems $\longrightarrow$ *temporal logics*
- systems managing information $\longrightarrow$ *logics of knowledge*
- autonomous systems $\longrightarrow$ *logics of goals, intentions*
- situated systems $\longrightarrow$ *logics of belief, contextual logics*
- timed systems $\longrightarrow$ *real-time temporal logics*
- uncertain systems $\longrightarrow$ *probabilistic logics*
- cooperative systems $\longrightarrow$ *cooperation/coalition logics*

Example: Pre-emptive Shopping

$$B_{me}^{>0.75} \Diamond G_{assistant} sell\_shoes(me) \Rightarrow I_{me} \Diamond^{<5s} leave\_shop(me)$$

"If I believe, with over 75% probability that at some point in the future the shop assistant's goal will be to sell me some shoes, then I intend that within 5 seconds I will leave the shop."

Combining Logics

Since we cannot describe all aspects of a pervasive system in one framework, we will often need to *combine* formalisms.

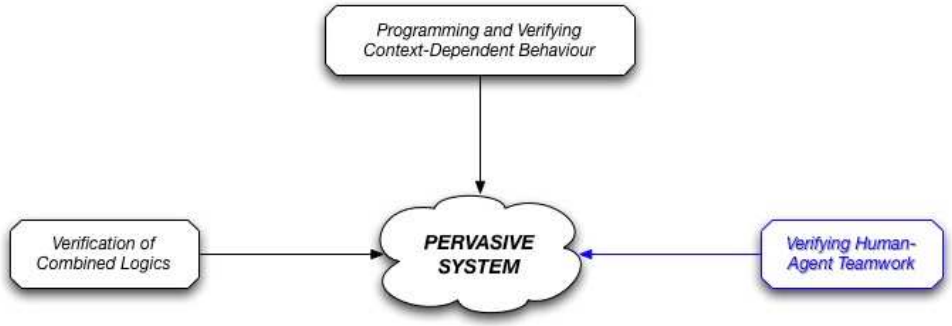
As we do *not* want to develop new verification techniques, we wish to re-use tools from the constituent logics.

So: we (*Savas*, *Sven*, *Michael*) are working on ways to practically verify complex combinations of logics by putting together the constituent model-checkers [A].

A. Model-Checking Combined Temporal Logics
— Konur, Fisher, Schewe. *In preparation.*

[*algorithms for, and complexity of, model-checking combinations of real-time temporal, probabilistic temporal, and modal logics*]

Modelling Teamwork



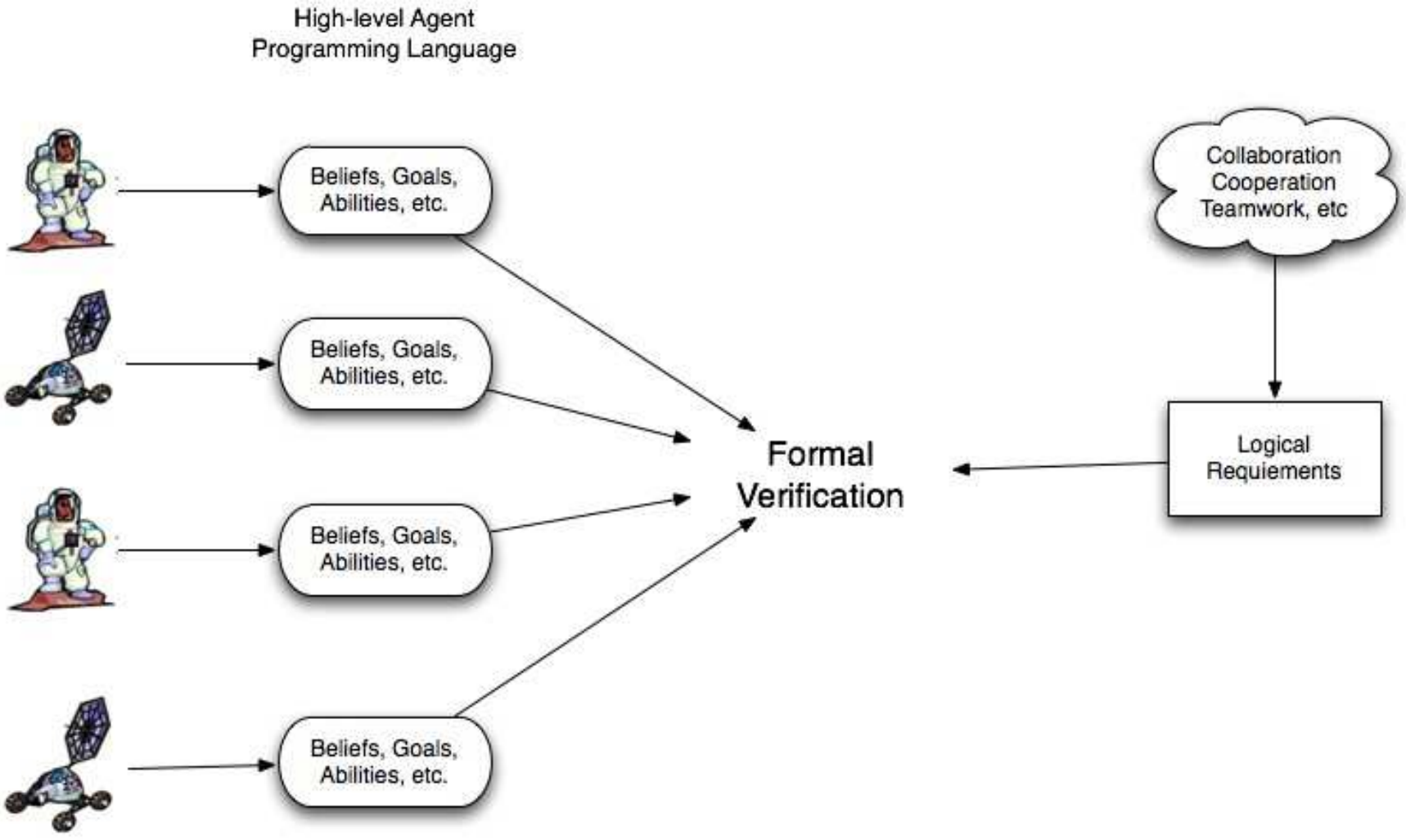
In a pervasive system, we can see humans and autonomous components as working together in a *team*.

But how might we try to verify behaviours within such human-computer teams?

We choose to model the *high-level* behaviours of the team participants, abstracting away from low-level details.

So, just describe the relevant behaviours of our humans and treat these as agents within our model-checking system.

Example — Astronaut-Robot Teams



Verifying Human-Agent Teams

Clearly: *humans* are impossible to describe precisely!

So, we model human behaviour at a very coarse level...
...then refine the model, introducing more detail if needed.

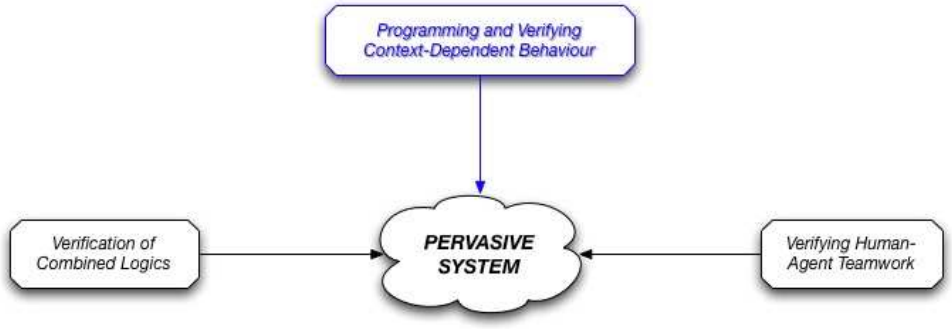
Richard will be looking at using *Louise*'s verification system [B] in order to verify high-level human-agent teamwork [C], for example utilising *Brahms* descriptions [D].

B. Automated Verification of Multi-Agent Programs
— Bordini, Dennis, Farwer, Fisher. *Proc. ASE'08.*

C. Formal Verification of Human-Robot Teamwork
— Bordini, Fisher, Sierhuis. *Proc. HRI'09.*

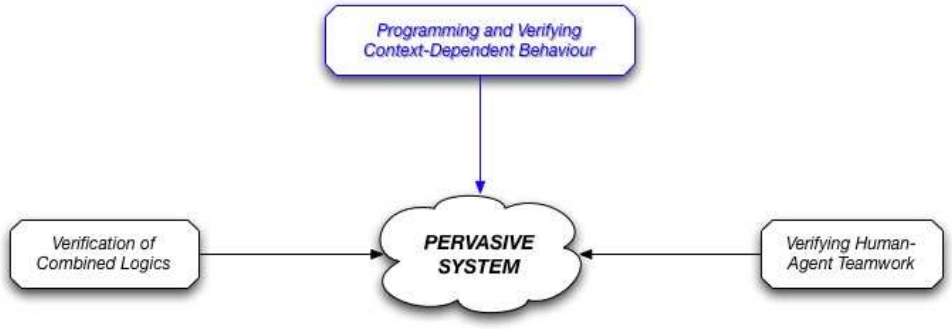
D. BRAHMS: A Multiagent Modeling and Simulation Language for Work System Analysis and Design — Sierhuis.

Context



Can we (formally) describe component/agent behaviour?

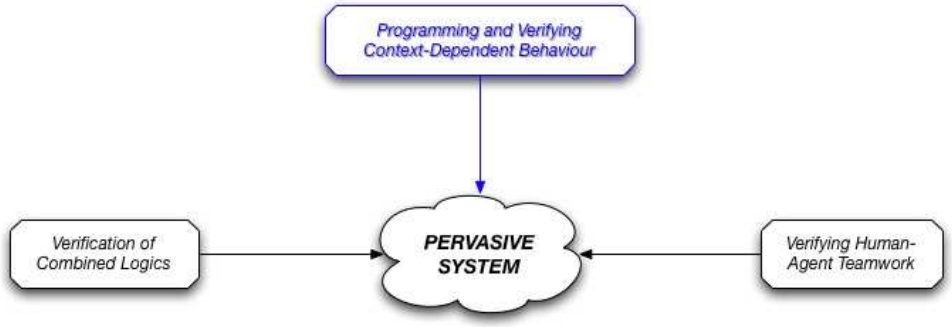
Context



Can we (formally) describe component/agent behaviour?

...especially if it occurs in multiple contexts?

Context

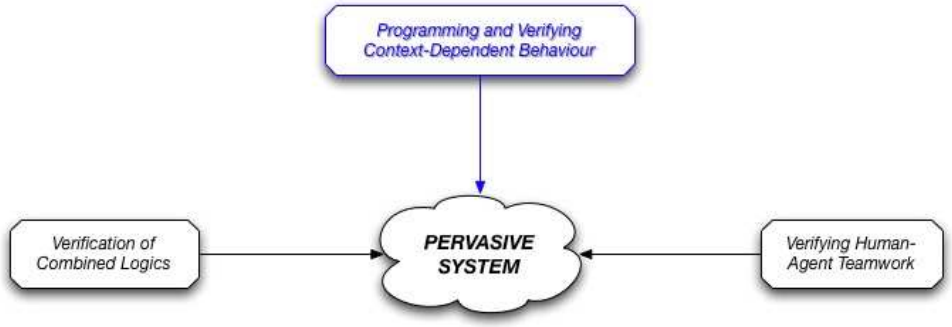


Can we (formally) describe component/agent behaviour?

...especially if it occurs in multiple contexts?

...and if it can move in and out of contexts dynamically?

Context



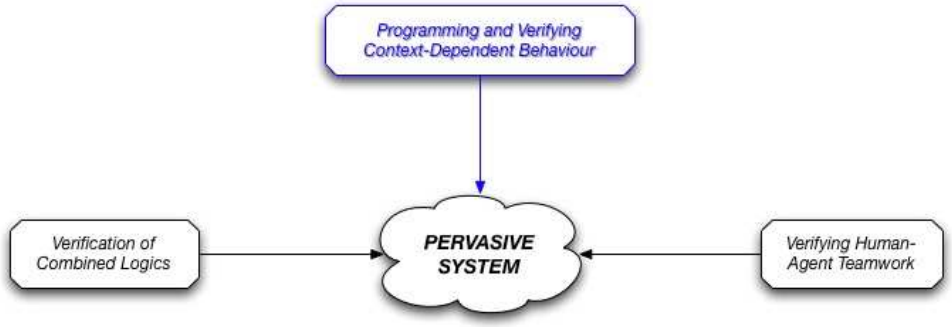
Can we (formally) describe component/agent behaviour?

...especially if it occurs in multiple contexts?

...and if it can move in and out of contexts dynamically?

...and if the notion of ‘context’ covers more than location?

Context



Can we (formally) describe component/agent behaviour?

...especially if it occurs in multiple contexts?

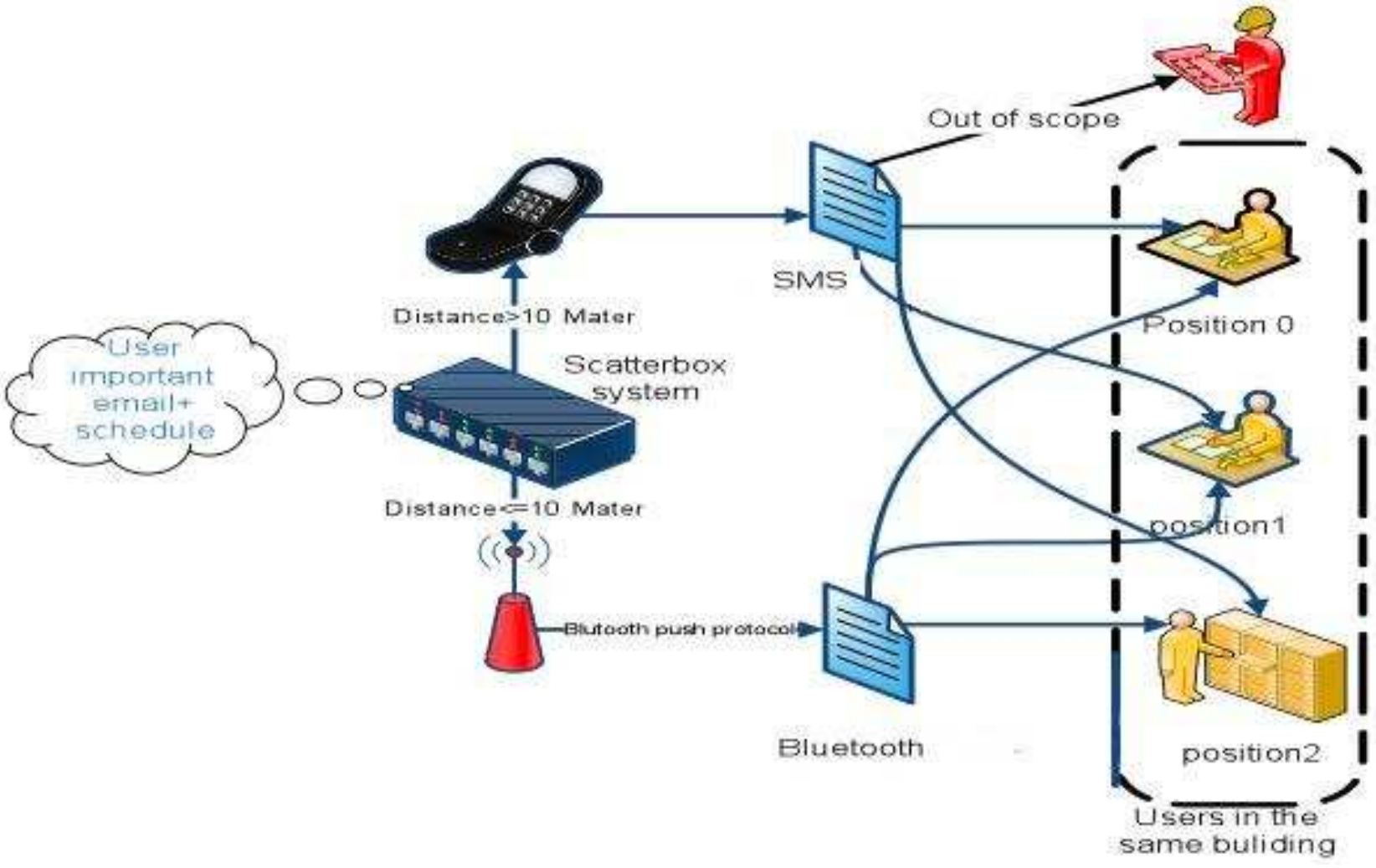
...and if it can move in and out of contexts dynamically?

...and if the notion of ‘context’ covers more than location?

We are looking at two aspects:

1. verification of a ‘real’ context-based system;
2. logical description of “context-oriented computing”.

Scatterbox


Out of scope
SMS
Distance>10 Mater
User important email+ schedule
Scatterbox system
Distance<=10 Mater
Blutooth push protocol
Bluetooth
Position 0
position1
position2
Users in the same buliding

Scatterbox Verification

Simple probabilistic models of:

- user movement;
- Scatterbox's belief about user position;
- Scatterbox's belief about user context;
- message-forwarding behaviour.

Then, in [E] *Savas* (and *Ahmed*, an MSc student) carried out simple PRISM verification

→ our aim is to refine this further in collaboration with Scatterbox developers.

E. Verification of a Message Forwarding System using PRISM. — Konur, Al Zahrani, Fisher *Proc. AVoCS'09.*

Contexts

In most pervasive systems, contexts just describe *location*.

However, contexts can represent *much* more:

- roles or societal norms
- teams or organisational structures
- styles or preferences, *etc*....

We are developing a logic-based programming language for such systems [F,G], but have yet to incorporate verification.

F. Language Constructs for Multi-Agent Programming
— Dennis, Hepple, Fisher. *Proc. CLIMA'08.*

G. Executing Logical Agent Specifications — Fisher, Hepple.
In *Multi-Agent Programming: Languages, Tools and Applications*.

Concluding Remarks

There's a lot to do!

Currently:

- theoretical work on combined model-checking;
- verification of 'realistic' systems using PRISM;
- high-level human-agent verification.

Big problem here is putting everything together.

Future:

security; context-oriented computing; abstraction; open/infinite systems; other application areas (robotics?); etc....

Spare Slides: Cooking Example

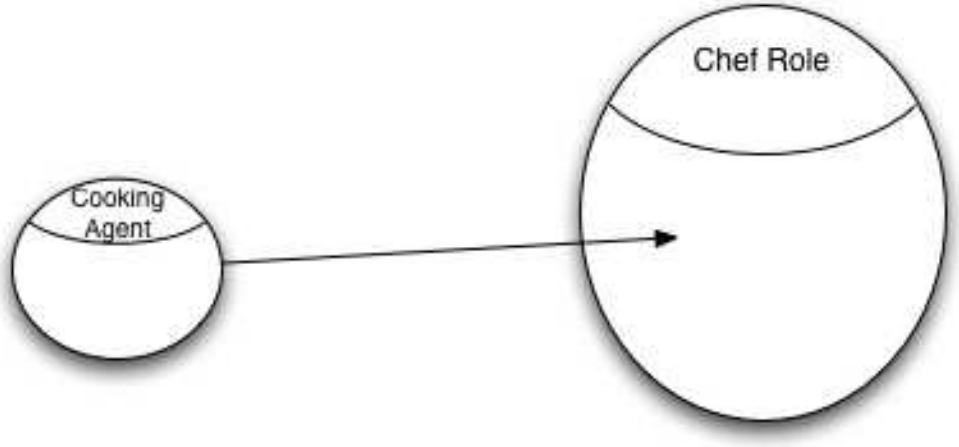
Context-Oriented Computing → Cooking

There is *much* more that we can do with contexts, as is shown by the simple ‘cookery’ example below.

Basically: *cooking agent* just wants to ‘cook’.

It doesn’t know *how* to cook and doesn’t know about any contexts — i.e. it has one goal and no plans!

So, the *cooking* agent moves into content of *chef role*.



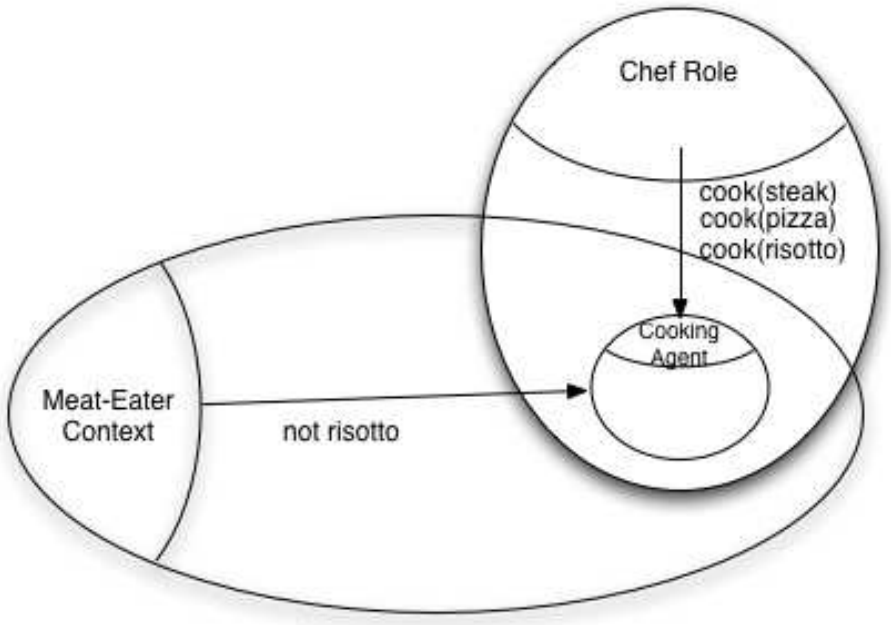
Learning to Cook

The *chef role* context provides capabilities of (i.e. plans for) cooking various meals: pizza; risotto; steak.

Since its one goal is to cook, it has 3 choices.

Meat eaters don’t like rice!

But: *cooking agent* now moves into a ‘meat-eater’ context.



Here, such meat-eaters dislike risotto and so send appropriate constraints.

Vegetarians don't like meat!!

Now: *cooking agent* takes on a vegetarian role, by moving into the appropriate context — *cooking agent* now can only cook pizza!