The Knowledge Engineering Review, Vol. 00:0, 1–19. © 2014, Cambridge University Press
DOI: 10.1017/S000000000000000 Printed in the United Kingdom

A Roadmap to Pervasive Systems Verification*

SAVAS KONUR[1] and MICHAEL FISHER[2]

[1] *Department of Computer Science, University of Sheffield, United Kingdom* *E-mail: S.Konur@sheffield.ac.uk*
[2] *Department of Computer Science, University of Liverpool, United Kingdom* *E-mail: MFisher@liverpool.ac.uk*

Abstract

The complexity of *pervasive systems* arises from the many different aspects that such systems possess. A typical pervasive system may be autonomous, distributed, concurrent and context-based, and may involve humans and robotic devices working together. If we wish to formally verify the behaviour of such systems, the formal methods for pervasive systems will surely also be complex. In this paper, we move towards being able to formally verify pervasive systems and outline our approach wherein we distinguish four distinct dimensions within pervasive system behaviour and utilise different, but appropriate, formal techniques for verifying each one.

1 Introduction

We are here concerned with the formal modelling and verification of *pervasive systems*, covering a very broad class of systems that are typically mobile, dynamic, autonomous, distributed, concurrent and context-based, and may involve humans and robotic devices working together (Dobson et al., 2010).

Let us consider a very typical pervasive system example that we have used in several earlier papers, e.g. (Arapinis et al., 2009):

> *As we walk into a shopping centre, 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 computer, which holds a 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 computer interacts with the shopping centre's network and finds that one of our friends is also shopping, so a message is sent to the friend's mobile/computer to coordinate shopping plans and schedule a meeting for coffee at a close location in 15 minutes.*

This might not yet exist, but such scenarios are neither very complex nor very far from being reality! There are currently several deployed pervasive systems partially employing similar scenarios, e.g. Scatterbox (a message forwarding system) (Knox et al., 2008), MATCH (a health care system for elderly and disabled people at home) (Turner, 2012), CityEvents (a mobile multimedia system for displaying geolocated cultural events) (Rosa et al., 2012), etc.

*The work described in this paper is partially supported in the UK by the following EPSRC projects: "Verifying Interoperability Requirements in Pervasive Systems" (EP/F033567) and "ROADBLOCK" (EP/I031812).

There are, of course, many definitions of a pervasive system (Want et al., 2002; Greenfield, 2006); even more if we include *ubiquitous systems* (Weiser, 1993) or *adaptive/autonomic systems* (Dobson et al., 2006). In this paper, rather than choosing one of these (often ambiguous) definitions, we will instead describe the systems we are concerned with in terms of the general classes of behaviour we *expect to see* within them. Thus, the "pervasive systems" we address here typically contain the following aspects:

1. multiple interacting entities, with multi-dimensional properties;
2. truly autonomous decision making;
3. the implicit involvement of humans *within* the system; and
4. context-dependent behaviour.

While we will explain these aspects in more detail in Section 2, it is important to note that many systems might have very simple/limited behaviours in one or more of these areas. For example, a pervasive system that has no interaction with humans clearly has a trivial behaviour in (3) above. It is also important to note that the four elements above are very general and while, for our analysis purposes, we consider them to be distinct, in practice there will be unavoidable overlap.

So, in describing and analyzing such pervasive systems, we choose to treat the above four aspects orthogonally as in Figure 1. As we see later, we verify properties of pervasive systems by verifying each aspect independently. However, in the next section, we will attempt to justify the above categorization, appealing to typical examples of pervasive systems.

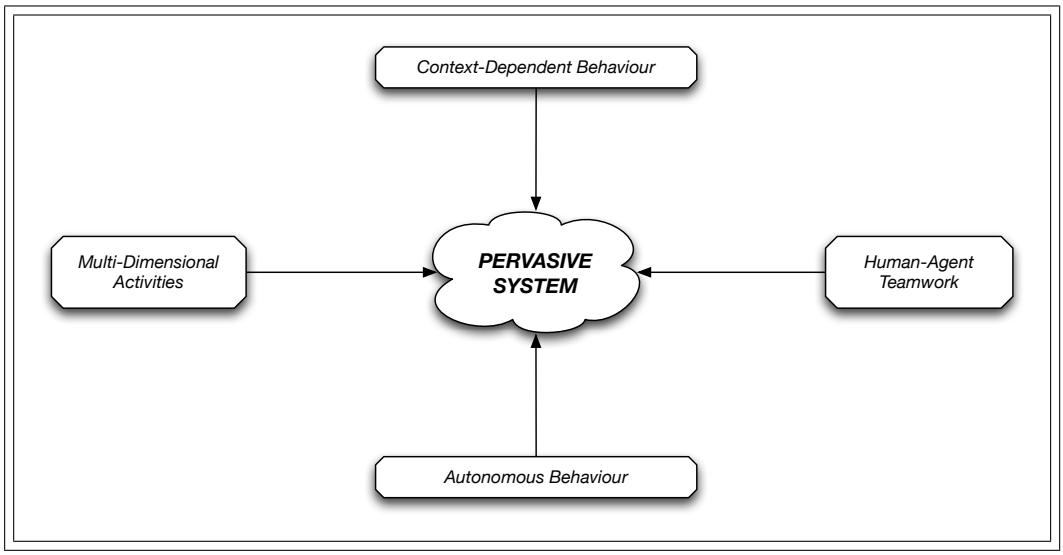

Figure 1: Four orthogonal aspects of pervasive systems.

Example. At this point, an obvious question is whether any real pervasive system can appropriately be described in terms of our four dimensions. At present we can only answer that all the example systems we have encountered can be viewed in this way. Here, we highlight a large pervasive system that can be potentially be analyzed as we suggest.

The K4Care platform (Isern et al., 2011) is aimed at organizing and providing healthcare services in "home care" domains. Specifically, it helps medical practitioners (e.g. doctors, physicians, specialists and nurses) provide administrative and medical services, simplifying the coordination between participants. The K4Care platform also features tools and methods helping the practitioners prepare personalised treatments for patients based on healthcare standards.

We briefly identify our four views on the K4Care example.

1. The K4Care model is implemented on the K4Care platform which is itself distributed and multi-layer. For example, the platform involves a *knowledge layer*, a *data abstraction layer*, and an *agent-based*

layer. That all these aspects must interact and that real-time response is clearly of importance, together points towards quite complex multi-dimensional descriptions.
2. The K4Care system is agent-based and involves a quite sophisticated range of reasoning techniques, e.g about treatments, effects, diseases, etc. There appears to be a certain amount of autonomous decision making in the system, again conforming to one of our key dimensions.
3. The provision of such health services involves not only interaction with a range of medical practitioners (e.g. doctors, nurses) but also with "Home Care patients". Such patients are typically elderly, with illness and/or physical impairment. Modelling the activities of the K4Care system clearly involves human-system interactions.
4. Finally, the system utilizes *context*, primarily in the form of ontologies, both at the level of reasoning about treatment/procedures, and at the level of reasoning about activities in different situations.

Clearly, semi-automated home care support is an area where safety, security, reliability and efficiency are all vital. And so formal methods, particularly formal verification, can surely be used.

2 Characteristics of Pervasive Systems

We now consider the four aspects from Figure 1 in more detail.

2.1 Multi-Dimensional Activity

Pervasive systems comprise many different facets and so we will need to describe/specify not just their basic dynamic behaviour, but also how all these facets/dimensions evolve and interact. Even in the relatively trivial "shopping" example above, we can see that the system involved has many different facets.

- It clearly has *dynamic* behaviour, together with (changing) knowledge about people and their plans.
- There are *real-time* aspects, not only in scheduling an appropriate meeting time but also in responding to shopper movements in a timely manner.
- Since at least some part of the pervasive system has to try to model the "real world", or at least an abstraction of the environment, and since the sensors that are typically used for this can never be perfect, then *uncertainty* must, necessarily, be involved.
- There is *collaboration* and *cooperation*, both at the level of agents 'representing' shoppers and at the level of shops and agents cooperating to provide appropriate information and capabilities.
- Many elements are mobile, and there are many programs working at once but in different areas. Thus, *mobility*, *distribution* and *concurrency* are essential.
- Clearly the software representing a shopper must also carry out *autonomous decision-making* (see Section 2.2), for example to instigate a meeting for coffee.

And so on. As we can see, even from this simple example, a pervasive system can have many different facets. Since we wish to specify and verify such systems, then we must ensure our specification (Section 3.1) and verification (Section 4.1) techniques take these dimensions into account.

We will later see how a typical pervasive message forwarding system, such as Scatterbox (Knox et al., 2008), requires a wide range of additional dimensions in order to explain/describe its behaviours:

Uncertain behaviour/accuracy of sensors ..... *probabilistic (behaviours)*
Location and tracking ..... *spatial*, *dynamic*
System *beliefs* about location and activity ..... *knowledge*
Importance of messages ..... *priority*
Varying levels of user interruptibility ..... *priority*
Communication connection failures ..... *probabilistic*
Requirements of system efficiency ..... *real-time*
Security of sensitive data ..... *knowledge*, *temporal*
Modelling intelligent choices ..... *motivations*
Roles and responsibilities ..... *multi-agent*

So, in general, to be able to clearly specify any non-trivial pervasive system we will require descriptions that combine, and explain interactions between, many such dimensions.

2.2 *Autonomous Decision-Making*

Pervasive systems contain components that can make autonomous choices, i.e. are often separated from direct human control. Thus, with this *autonomy* the computational components can decide, for themselves, what to do and when to do it. The specific form of autonomy we consider is that captured by the concept of a *rational agent*. An agent (Wooldridge, 2002) essentially captures the idea of an autonomous entity, being able to make its own choices and carry out its own actions. Beyond this, *rational agents* are typically employed to represent more complex autonomous decision-making where communication and explanation are vital; see Figure 2.

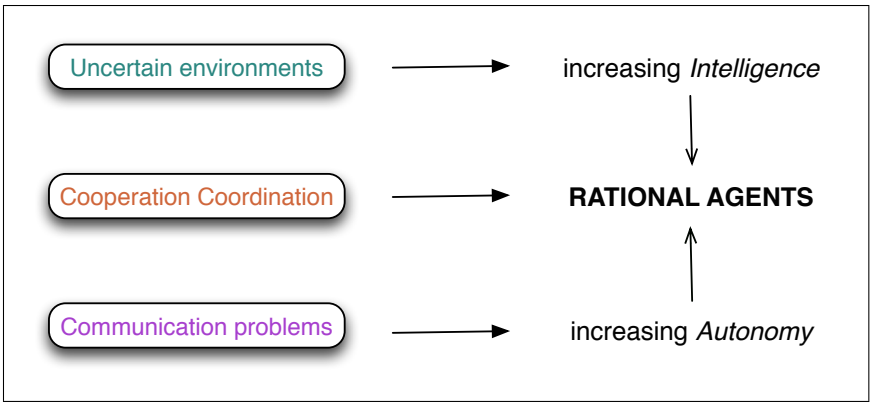

Figure 2: Motivation for use of rational agents.

These agents are *rational* in the sense that they take their decisions in a *rational* and *explainable* way (rather than, for example, purely randomly). The central aspects of the rational agent metaphor is that such agents are autonomous, but can react to changes in circumstance and can choose what to do based on their own agenda. In assessing such systems it may not be sufficient to consider *what* the agent will do, but we must often also consider *why* is chooses to do it. The predominant view of rational agents is that provided by the BDI (beliefs-desires-intentions) model (Rao and Georgeff, 1991, 1995) in which we describe the goals the agent has, the 'beliefs' it has, and the choices it makes. The rational agent's *intentions* are then determined by its *beliefs* and *goals*.

2.3 *Humans "in the loop"*

Pervasive systems typically involve *humans*, or at least external autonomous entities, and a variety of *tools* or *artifacts* within the system. Often, humans are actually *embedded* within pervasive systems, with computation and communication happening all around them. In many cases, the overall behaviour of the whole pervasive system depends on appropriate behaviour of the humans involved; in this case we can see the human/system relationship as a very weak form of cooperation.

In some areas, computational/robotic elements play an even more collaborative role in the human-system scenario. This is particularly the case if actual physical robots are involved. Although many practical activities suggest human-robot cooperation, space exploration is an area at the forefront of this where *cooperation* and *teamwork* is clearly required. As an example, NASA is interested in *Robot-Astronaut* teams to be deployed on Mars (Sierhuis et al., 2003); see Figure 3(a).

While not directly involving such space applications, pervasive systems not only can involve multiple robotic elements but also can require human-system interaction of a similar form. Consequently, the problems raised by human-robot interactions can often re-appear in pervasive scenarios. On the domestic front, the close interaction of multiple humans and robots in pervasive scenarios is becoming more

Figure 3: Human-robot teamwork (a) in planetary exploration (NASA, Astronaut-Robot-Team-Concept), and (b) in the home (RoboSafe, 2013).

prevalent, particularly in healthcare situations. Thus, the Robotics industry is actively developing "*Robots for the Home*", particularly as human assistants; see Figure 3(b). Clearly, this leads to safety concerns:

- Are the robots *safe*?
- How can we tell a robot what we *want*?
- *Will* a robot work out what we want it to do?
- And then, are we *sure* it will decide to do what we want?

This is a natural area where formal verification should, and can, make an impact (Fisher et al., 2013). Indeed, as we will see later, specification and verification techniques developed for human-robot interaction have an important role to play in assessing (high-level, and abstracted) human involvement in pervasive systems.

2.4 Context Dependency

Pervasive systems are also *context-dependent* — their behaviour can change *because* of movement between contexts. This can take (at least) two forms.

Simple context view

In the first form, "context modelling" is essentially concerned with recording someone's location or activity (Chen et al., 2003). For example, detecting a Bluetooth signal of one's cellular phone in a seminar room can be used to derive his/her context (in form of location or activity) in conjunction with other information available e.g. calendar.

This context modelling is a popular technique studied by software engineers who wish to differentiate subtle behavioural differences in systems (Henricksen and Indulska, 2004; Strang and Popien, 2004). These behaviours are often in response to, or in anticipation of, human activity and some notion of 'context' of this form is at the heart of very many pervasive systems.

Complex context view

Another, more general, view is that 'context' should cover *much* more than location or activity. For example, in the METATEM agent programming language (Fisher and Hepple, 2009), the context in which an agent is acting is a first-class construct. Indeed, contexts are also agents and an agent is considered to reside in (or be contained by) zero or more contexts. Context in METATEM is conceived as the more abstract and flexible notion of *anything* that has an influence over an agent in a system. Specifically, contexts do *not* just concern location, but can be

- roles or societal norms,
- teams or organisational structures,
- activities,

- styles or preferences,
- locations,
- *etc*....

This gives a very powerful mechanism for programming and describing quite complex context-dependent behaviours and so complex organisations and systems (Hepple et al., 2008). While there remains debate about exactly what an 'agent' is, this view of hierarchical and active agents perhaps needs further comment. While we can see an agent as capturing basic autonomous behaviour, there are many different levels of abstraction over which this can be taken. Consider, for example, a flock of birds[†]. Seen at a high level of abstraction, the 'flock' can be viewed as an autonomous entity with its emergent behaviour being opaque to an external viewer. So, we could model the flock as an agent. But then, of course, if we move into the flock we can discern local interactions between individual birds. Correspondingly, once we look inside the agent's content we can see (and interact with) representations of individual birds.

This duality, being able to treat the larger entity as an agent while at the same time being able to move inside to interact with further agents in its content, is very powerful. It clearly has ancestry on object-based systems, but extends this to agents with active, autonomous behaviour. The context/content view is, thus, very useful for modelling entities that are autonomous and active at many different levels of abstraction and often do not correspond to physical entities. For example roles and norms, teams and other organisational structures, styles and preferences, etc.

It is important, in our later sections, that we are able to verify both varieties of the notion of 'context' described above. The first is simpler, but is used in many deployed pervasive systems. The second approach is more powerful, and something like this is likely to become used in future pervasive system development. Below we give an example for each approach.

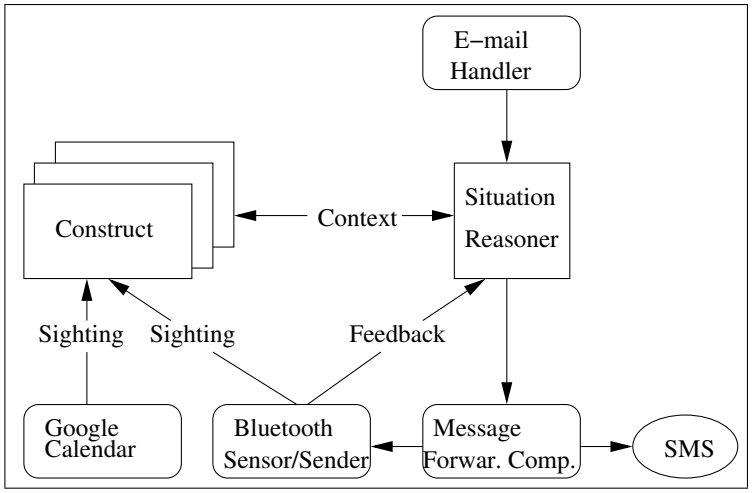

Figure 4: Scatterbox message forwarding system (Konur et al., 2014).

Example (Knox et al., 2008). The Scatterbox system, as well as exhibiting multi-dimensional behaviour as described in Section 2.1, also involves context-dependent behaviours. As described in (Knox et al., 2008), Scatterbox is a message forwarding system, primarily designed as a testbed for context-aware computing in pervasive scenarios. Each Scatterbox user's context is defined by tracking his/her location and by monitoring his/her daily schedule. This data is analysed, and 'situations' are identified that indicate the user's level of interruptibility. Once a message arrives, Scatterbox forwards it to the subscribed user provided the user's available context suggests they are in a situation where they may be interrupted. A schematic diagram of Scatterbox is given in Figure 4.

[†]Our thanks to an anonymous reviewer for highlighting this.

Example (Hepple, 2010). Consider an abstract 'cookery' example. Here, a *cooking agent* simply wants to 'cook'. But, to begin with, it does not know *how* to cook and does not know about any contexts. Specifically, it initially has one goal (i.e. to 'cook') but has no plans telling it how to go about this. So, this *cooking* agent now moves into a *chef role* context, which is itself another agent; see Figure 5(a). This context agent provides the cooking agent with appropriate information, plans and capabilities associated with the 'role' of being a 'chef'; see Figure 5(b). Once the agent has received and assimilated these aspects then it can legitimately take on the role of a 'chef' and can indeed satisfy its goal to be able to 'cook'. So, just by moving into a new context, the cookery agent has obtained new computational capabilities. Moving into further contexts can bestow additional capabilities, or even constraints, upon the agent. In this way, contexts are active, providing more than just a representation of 'location'.

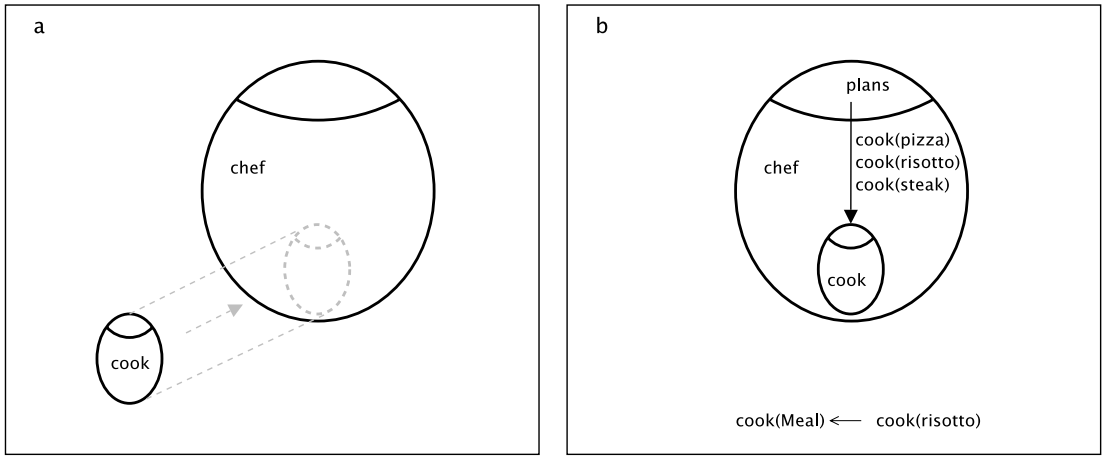

Figure 5: Cooking scenario (Dennis et al., 2008).

3 Specifying Pervasive Systems

Following our theme, if we are to program a pervasive system or specify system properties, then we must address each of our four dimensions in turn (see Figure 6). In this section, we will both discuss how the properties of multidimensionality are specified and how certain aspects of systems are programmed.

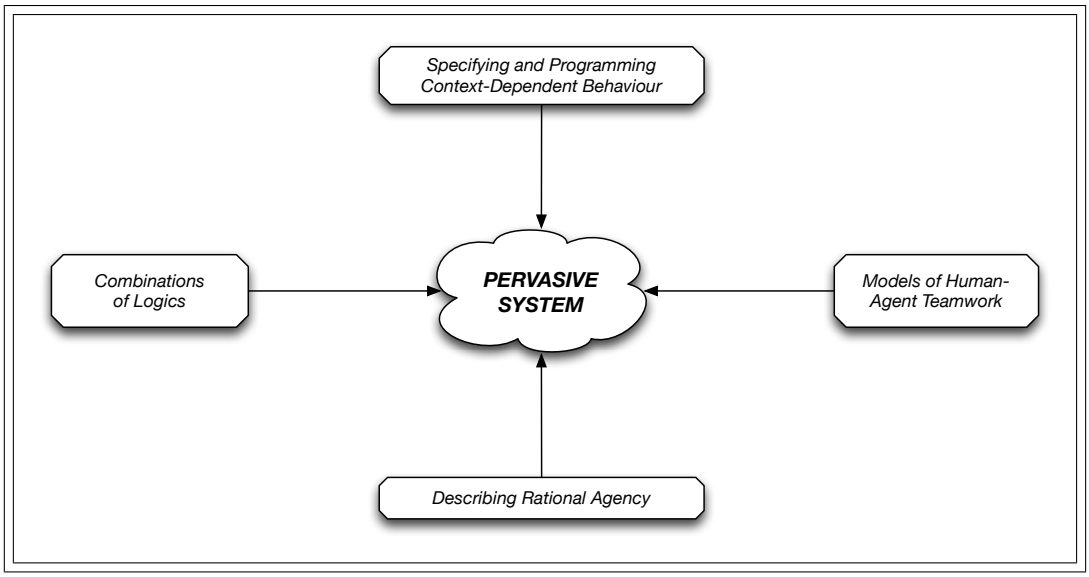

Figure 6: Four orthogonal aspects for *specifying* pervasive systems.

3.1 Multi-Dimensional Activity

In the *specification* of different aspects and dimensions, we use *formal logics* (Konur, 2013). Logics provide a well-understood and unambiguous formalism for describing the *behaviours*, *requirements*, and *properties* of systems. In addition, as we are concerned with the verification of pervasive systems, the well researched proof, complexity, and satisfiability checking results can provide us with many practical tools such as automated deduction or model checking systems.

An obvious issue with formally specifying pervasive systems is that formal logics have limitations. For example, there might be system properties which cannot be represented formally in a particular logical system. In order to maximise the expressivity of logics and hence to be able to verify more complex properties, we combine different aspects of different logics.

Beyond classical Boolean logic there are vary many logics devised for a huge array of different types of system. Choosing the appropriate logic provides a level of abstraction close to the key concepts of the software, for example

- dynamic communicating systems $\longrightarrow$ *temporal logics*
- systems managing information $\longrightarrow$ *logics of knowledge*
- 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*

Since we cannot easily describe all aspects of a pervasive system in one of these logics, we will often need to *combine* formalisms. There are several standard ways to combine logics, with the three standard ones being (Gabbay et al., 2003): *temporalization* (Finger and Gabbay, 1996); *fusion* (or *independent join*); and *product* (or *full join*). Now, imagine we have two logics to combine, logic A (typically, a temporal logic) and logic B.

We assume the logics A and B are graphically represented as in Figure 7.

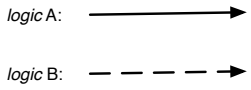

Figure 7: Notation (Konur et al., 2013).

The *temporalization* permits only a restricted way that two component languages can interact with each other. It is denoted by $\mathrm{A}(\mathrm{B})$, where a pure subformula of logic B can be treated as an atom within logic A. This combination is *not* symmetric — A is the main logic, but at each world/state described by logic A we might have a formula of B describing an embedded "B-world" (see Figure 8).

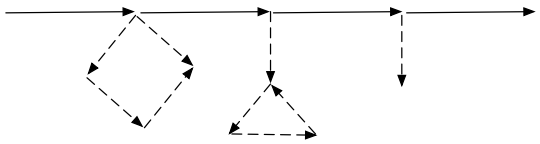

Figure 8: Temporalization of two logics (Konur et al., 2013).

As Figure 8 suggests, after we take some A-step(s) (straight arrow), we can take a B-step (dotted arrow). From this point on, we are in the B world, and we cannot take an A-step anymore.

The *fusion* $\mathrm{A} \oplus \mathrm{B}$ is more symmetric than temporalization in that, at any state/world, we can either take an "A-step" or a "B-step" (see Figure 9). It is important to note that the two logical dimensions are essentially independent (hence: "independent join").

The *product* combination, $\mathrm{A} \otimes \mathrm{B}$, is similar to the fusion, but with a much tighter integration of the logics (see Figure 10). Operators of the constituent logics tend to be *commutative*. This provides a stronger integration of the component logics but, correspondingly, often results in logics of higher complexity than fusions or temporalizations.

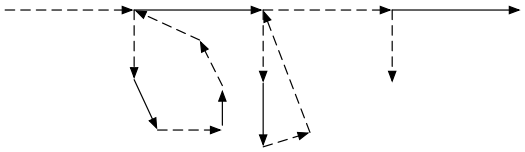

Figure 9: Fusion of two logics (Konur et al., 2013).

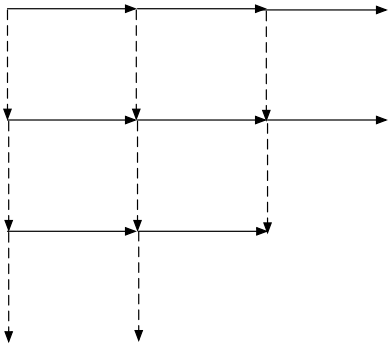

Figure 10: Product of two logics (Konur et al., 2013).

Example. Once we can combine logics, for example using fusions, then we can combine several logics together in order to provide a quite powerful descriptive framework. As an example of why combinations of logics may be appropriate for formally specifying multi-dimensional behaviour in a pervasive system, consider the following statement from an assistive healthcare scenario.

> *"If a patient needs medicine, then the patient believes that there is a probability of 65% that, within 10 minutes, a nurse robot will want to assist the patient."*

In formalizing this we might write

$$needs\_medicine(patient) \rightarrow B^{\geq 0.65}_{patient} \Diamond^{\leq 10} G_{nurse}\ assist(patient)$$

where

$B^{\geq 0.65}_{patient}$ means *patient believes with* 95% *probability* – probabilistic logics

$\Diamond^{\leq 10}$ means *within 10 minutes* – temporal logics

G_{nurse} means the nurse robot *has a goal*, and so *wants* to help the patient – BDI logics

3.2 Autonomous Decision-Making

Since our autonomous decision making components are to be characterised as rational agents, we *program* such autonomous decision making using high-level programming languages for rational agents. Typical programming languages for rational agents (Bordini, Dastani, Dix and El Fallah Seghrouchni, 2005; Bordini et al., 2009) provide representations of

- *beliefs*
- *desires* (often referred to as *goals*)
- *intentions* (or invocations of *rules/plans*)
- *actions*
- *deliberation mechanisms*

Based on its beliefs and desires an agent can select an intention, or plan of action, which can then invoke further basic procedures. This makes the programming and inspection of the decision making much more

intuitive since the programmer is encouraged to describe the situations in which a given course of action is applicable or desirable.

An example agent rule/plan in a typical programming languages for rational agents is given below (Fisher et al., 2013):

```
Goal(eat) : Belief(has_money),
            Belief(not has_food)
            <- Goal(go_to_shop),
               Action(buy_food),
               Goal(go_home),
               Action(eat),
               +Belief(eaten).
```

This means that if the agent intends to eat and if it believes it has money, but does not have food, then it will intend to go to the shop. Once the agent achieves its desire (i.e. goal), it will buy some food and then intends to go home. When it is at home, the agent will eat and then update its beliefs to record that it believes it has eaten.

Modern agent programming languages in this BDI (Beliefs, Desires and Intentions) paradigm were pioneered by the PRS in the early 90s (Georgeff and Lansky, 1987; Rao and Georgeff, 1992), which sought to abstract *decision-making* away from the more procedural aspects where control systems were used. This BDI architecture has been shown to generalise widely to any situation in which autonomous decision making is required to control complex software processes and a family of programming languages has been developed to support the paradigm, for example AGENTSPEAK (Rao, 1996; Bordini, Hübner and Vieira, 2005), 2APL (Dastani et al., 2005), GOAL (Hindriks et al., 2001) and Gwendolen (Dennis and Farwer, 2008).

3.3 Modelling Teamwork: Humans "in the loop"

In a pervasive system, we can often see humans and autonomous components as working together as a *team*. But how might we try to describe behaviours within such human-computer teams? To simplify, we choose to model the *high-level*, cooperation and interaction, behaviours of the team participants, abstracting away from low-level details such as movement. Specifically, we choose to carry out the modelling of these high-level behaviours using the *Brahms* framework (Sierhuis, 2001; van Hoof, 2000; Sierhuis and Clancey, 2002). This is a framework designed to model both human and robot activity again using *rational agents*. As it was developed to represent activities in real-world contexts, it also allows the representation of artifacts, data, and concepts in the form of classes and objects. Both agents and objects can be located in a model of the world (the geography model) giving agents the ability to detect objects and other agents in the world and have beliefs about the objects. Agents can move from one location in the world to another by executing *move* actions, so simulating physical movement. For a more detailed description, see (Sierhuis, 2001) and (Sierhuis, 2006).

The Brahms language was originally conceived of as a language for modeling contextual situated activity behavior of groups of people. This has now evolved into a language for modelling both people *and* robots/agents. As such it is ideal for describing human-agent/robot teamwork and indeed has been used for this both in mundane scenarios and in modelling the "Robot-Astronaut Teams on Mars" mentioned earlier. It has been particularly successful as it allows us to describe, concisely and precisely, the overall team behaviours and so assess how well such teams work together.

Example (Stocker et al., 2012). We now consider a domestic home care application as an example to illustrate the Brahms specification language. In this scenario, the helper robot reminds the assisted person Bob to take his medication. The scenario is explained as follows:

> *The Robot has a workframe to deliver medicine to Bob; activated at pre-allocated times. The Robot places the drugs on Bob's tray and then monitors them hourly to check if they have*

been taken. A detectable `takenMedicationC` *aborts the workframe if the drugs have been taken and then updates the Robot's beliefs. If the drugs have not been taken the workframe reminds Bob to take his medication. The Robot counts the number of times it reminds Bob, and after 2 reminders it notifies the House.*

A fragment of the Brahms specification of this scenario is given below (Stocker et al., 2012):

```
workframe wf_checkMedicationC
{
   repeat: true;
   priority: 3;
   detectables:
      detectable takenMedicationC
      {
        when(whenever)
        detect((Bob.hasMedicationC = false), dc:100)
        then abort;
      }
   when(knownval(current.perceivedtime > 14) and
        knownval(Bob.hasMedicationC = true) and
        knownval(current.checkMedicationC = true))
   do
   {
     checkMedication();
     remindMedicationC();
     conclude((current.checkMedicationC = false));
     conclude((current.missedMedicationC = current.missedMedicationC + 1));
   }
}
```

A `workframe` corresponds to a plan in standard rational agent programming languages. This particular one has a certain priority level, events that can be detected, a set of triggers (captured within the `when` clause), and a list of outcomes (after the `do`) such as calling further workframes or updating beliefs.

3.4 Context Dependency

Simple context view

In the shallow view of "context" we can simply describe/specify contextual information in terms of changes in location. For example, we might describe a simple model of user movement by a straightforward state-machine; see Figure 11. This then allows us to use a simple formal language, such as *temporal logic* to describe changes in context, such as "$\Diamond$SHOPPING", meaning "eventually the user will be in a SHOPPING context".

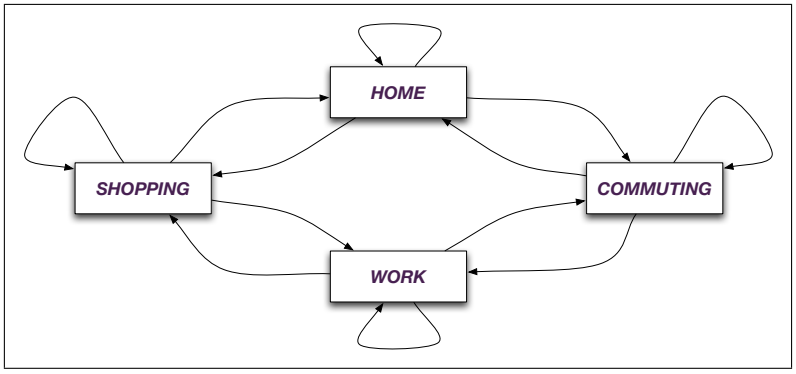

Figure 11: A simple model of user movement.

Complex context view

In the case where we have the much more complex view of context, then (because each context is itself an agent) we must embed this notion within an *agent programming language*. The formal semantics for such a language, a variety of AGENTSPEAK, is provided in (Dennis et al., 2008). We can see from the simple 'cookery' example earlier that it is straightforward to view contexts as agents and so develop contexts that share full agent-level behaviours amongst their contents. For example, when the cook enters the chef context (see Figure 5), the sending of plans is triggered, and the cook agent receives plans for preparing risotto, steak and pizza. This behaviour is written in the agent programming language as the following plan (Dennis et al., 2008):

```
+content(Agent)[source(self)]
  <- .print("Sending ", Agent, " plans...");
     .send(Agent, tellHow, "+!cook(risotto)
                              : check_constraint(cook,risotto)
                                <- make(risotto).");
     .send(Agent, tellHow, "+!cook(steak)
                              : check_constraint(cook,steak)
                                <- make(steak).");
     .send(Agent, tellHow, "+!cook(pizza)
                              : check_constraint(cook,pizza)
                                <- make(pizza).").
```

The plan is triggered, in the Context agent, by a new `Agent` entering it. Once this happens, the Context agent prints a statement then sends three messages to the `Agent` that just entered it. These messages contain the new plans that the `Agent` can then add to its own capabilities.

4 Verification

As described above, in order to carry out formal verification of pervasive systems, we will actually verify our four distinct dimensions separately; see Figure 12. So, let us examine all these four in more detail. As

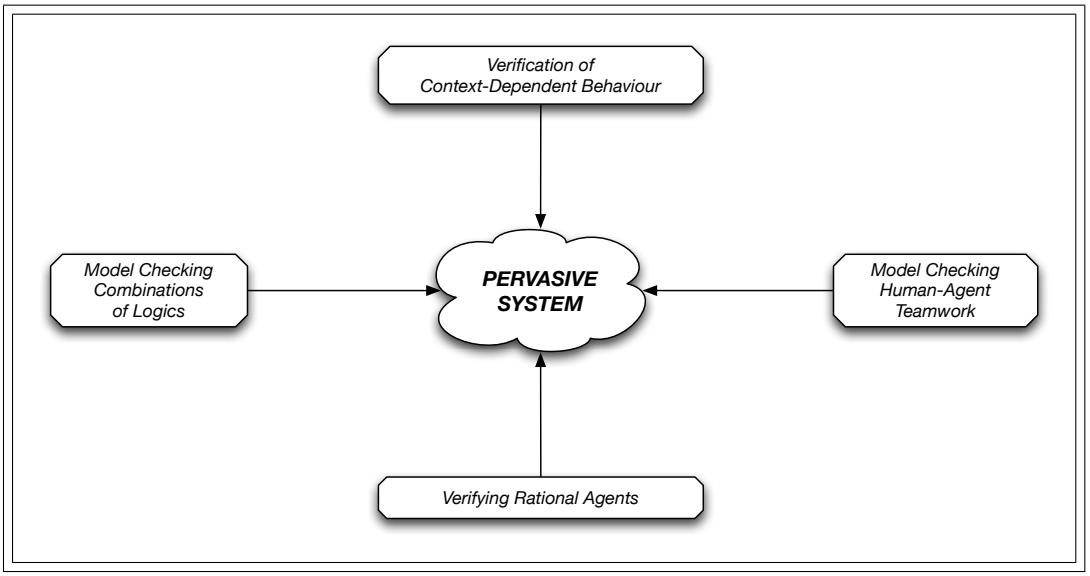

Figure 12: Four orthogonal aspects for *verifying* pervasive systems.

we will see below, all are based on model checking (Clarke et al., 1999), but in very different ways.

4.1 Verifying Combined Specifications

Formal description of pervasive and ubiquitous systems can involve many different logical dimensions. This makes the verification task extremely difficult, since the interactions between these dimensions increase complexity significantly. The techniques currently used for formalizing and verifying aspects of pervasive and ubiquitous systems are very limited, often trying to describe system behaviour by considering just one dimension rather than dealing with multiple formalisms at the same time.

One idea to tackle the problem would be to introduce complex logics comprising all the aspects needed and develop new model checking methods and tools. However, devising new model checking methods/tools for such complex logics is both difficult and time consuming.

In (Konur et al., 2013), we present a verification technique which allows the combination of model checking techniques for different formal frameworks. Namely, instead of introducing new logics for different combined aspects and developing model checkers for these logics, we combine existing logics and devise a *generic* model checking method for these combined frameworks. A combined model checking procedure can then be synthesised, in a modular way, from model checkers for constituent logics. This avoids the re-implementation of model checking procedures.

An overview of our approach is presented in Figure 13. As the figure illustrates, a combined model checker $\mathbf{MC}_{\mathrm{A}\oplus\mathrm{B}}$ for the combined logic $\mathrm{A}\oplus\mathrm{B}$ employs the model checkers $\mathbf{MC}_{\mathrm{A}}$ and $\mathbf{MC}_{\mathrm{B}}$ for the constituent logics A and B, respectively. Note that in the generic model checking procedure there are several calls to $\mathbf{MC}_{\mathrm{A}}$ and $\mathbf{MC}_{\mathrm{B}}$, but they are not shown in the figure to keep the presentation simple.

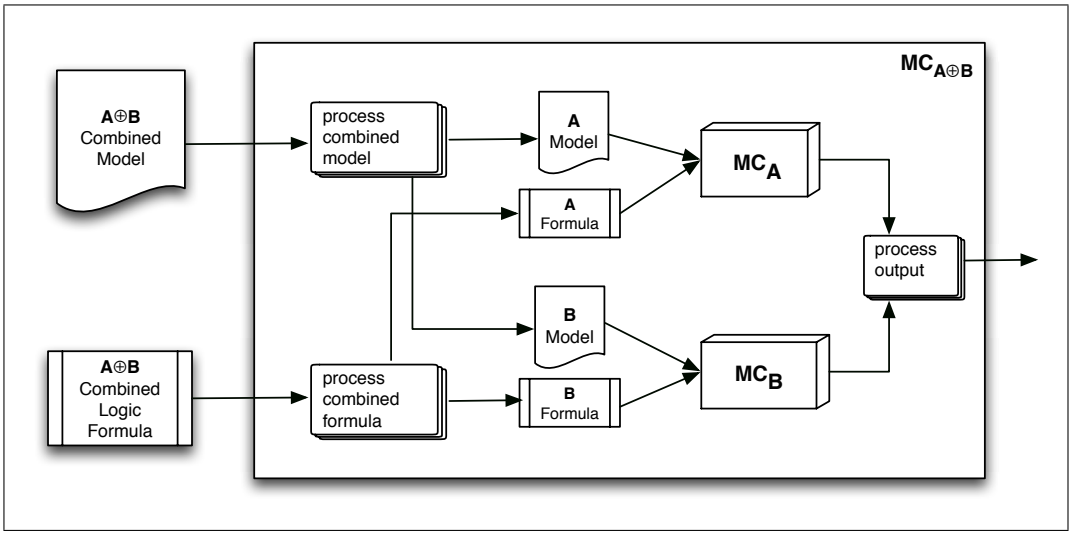

Figure 13: Overview of the combined model checking approach of (Konur et al., 2013).

This technique can be used to verify combined real-time, probabilistic, informational aspects, etc. which are at the core of pervasive and ubiquitous systems. Using this method, we can model check the combinations of the logics discussed in Section 3.1: *temporal logics*, *logics of knowledge*, *logics of beliefs*, *real-time temporal logics*, and *probabilistic logics*.

4.2 Verifying Autonomous Behaviour

An advantage of using the BDI-paradigm to program and model autonomous decision making is that several BDI programming languages now have associated formal verification techniques (Fisher et al., 2007; Bordini et al., 2006; Jongmans et al., 2010; Dennis et al., 2012). The property specification languages within such model checkers have to be extended in order to describe the key reasoning components (beliefs, desires, etc.) but since these environments generally do not implement full logics of knowledge or belief such modalities can be treated in a simple and efficient fashion, normally as propositions within a temporal logic.

The points where an autonomous decision making layer must interact with lower-level control layers or the real world are more of a challenge and involve the construction of abstract models of these parts of the system. There are two approaches to this problem: one is to use advanced mathematical techniques such as hybrid automata to model the real world and the other is to use simple, highly abstract models, and focus on capturing clearly the conditions under which the rational layer makes valid decisions.

The AIL/AJPF model checking toolkit (Dennis et al., 2012) that we have developed not only takes the second of these approaches but, through the generic *Agent Infrastructure Layer*, provides a Java framework to enable the implementation of interpreters for various agent programming languages and the direct use of the AJPF agent program model checker. Through this AIL/AJPF approach, see Figure 14, verification for several high-level agent programming languages has already been carried out.

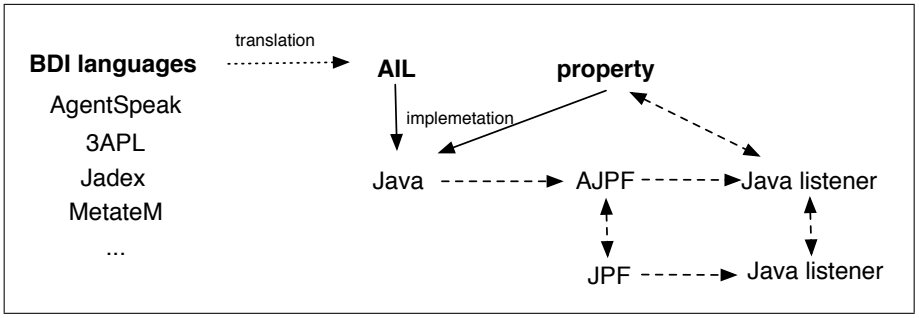

Figure 14: Verification of multiple agent programming languages (Bordini et al., 2008).

Here, operational semantics can be provided for a range of agent programming languages, including AgentSpeak and METATEM, and then these semantics can be used within the generic model-checking procedure provided by Java PathFinder (Visser et al., 2000).

4.3 Verifying Human-Agent Teams

As we have seen from Section 3.3, the *Brahms* simulation/modelling language is not only appropriate for describing human-agent teamwork, but has been used in practice for non-trivial human-robot interactions (Sierhuis, 2001; van Hoof, 2000; Sierhuis and Clancey, 2002). Consequently, Brahms provides us with a ready-made framework for describing human-agent teamwork, as well as providing a range of examples in specific areas (Clancey et al., 2003; Sierhuis et al., 2003; Sierhuis, 2006). It seems natural, therefore, to aim towards the formal verification of Brahms models. The formal operational semantics for the Brahms language produced in (Stocker et al., 2011) now forms the basis of our formal verification of Brahms; see Figure 15. In (Stocker et al., 2012) a Brahms model of a human-agent team scenario is translated to a Java representation of the corresponding semantics structure, which is then translated into the SPIN model checker's input language, `Promela` (Holzmann, 2003). Formal proerties are also represented in `Promela`, and the verification problem is passed on to the SPIN tool. This approach has been extended and refined, particularly improving efficiency by translating to alternaitve model-checkers (Hunter et al., 2013).

4.4 Verifying Context-Dependent Behaviour

Simple context view

In this view, the context information is obtained by getting the location or the activity. For example, in the Scatterbox system, the user context is obtained both by tracking the user's location (via location sensors) and by monitoring his/her daily schedule. The Scatterbox system analyzes this context data, and determines the user's context and corresponding level of *interruptibility*. However, the user's actual context might be different than the context derived by Scatterbox. Therefore, the exact interruptibility level, determined by the exact user movement, and Scatterbox's *perceived* user interruptibility might also be different. We therefore distinguish between the *exact* interruptibility and the Scatterbox's *belief* about

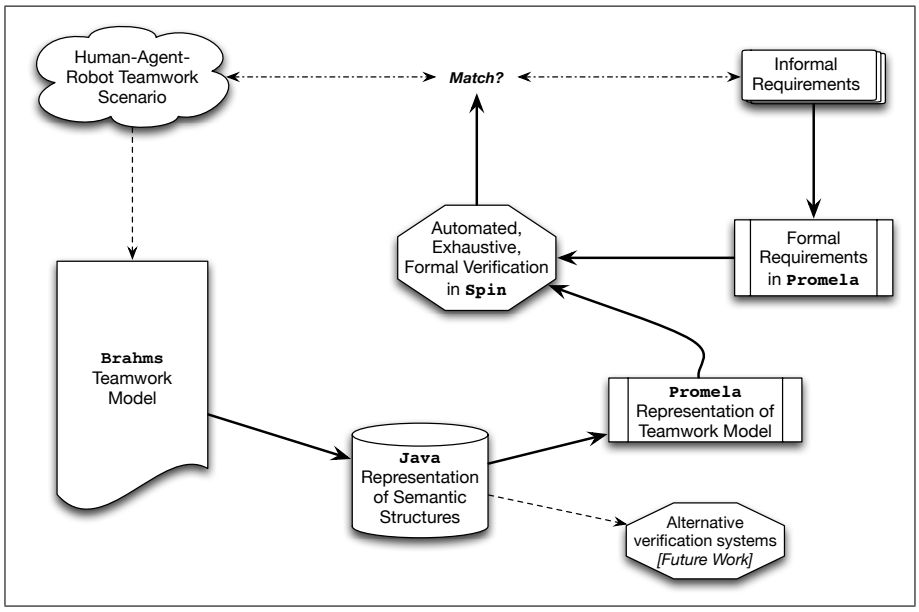

Figure 15: Formal verification process of Brahms (Stocker et al., 2012).

the interruptibility, as these two might have different values. The system's belief about the interruptibility depends on the accuracy of the context acquisition process.

The Scatterbox system also analyzes incoming e-mails to determine the *importance* level. Similar to the interruptibility case, the exact importance level and Scatterbox's perceived importance level might be different. We therefore distinguish between these two variables. For the same reason, we distinguish the exact calendar information and the perceived calendar information.

When we verify the overall operation of the Scatterbox system, we usually refer to the perceived information by Scatterbox. For example, if we want to verify that the Scatterbox accurately finds the interruptibility of the user, we need to check whether the formula $\Box(B\_\mathsf{intr} = \mathsf{intr})$ is true over the model describing the behaviour of the Scatterbox system (intr and B_intr denote the exact interruptibility and perceived interruptibility, respectively).

In (Konur et al., 2014) we formally verified the Scatterbox system using probabilistic model checker PRISM (PRISM, 2013). Since the user location is determined via location sensors, a quantitative understanding of the effects of uncertain sensor input and inference was an essential part of this analysis. We also analysed the dynamic temporal and stochastic behaviour of the system, allowing us to verify formal requirements even in the presence of uncertainty in sensors.

Complex context view

As discussed in Section 3.4, context can be modelled in BDI-style agent programming languages in a variety of ways — it can be subsumed into the belief system, used as part of a modular construction of agent systems or modelled separately as a first class component of the agent state (Dennis et al., 2008; Fisher and Hepple, 2009). All of these approaches can be incorporated easily into the operational semantics of the BDI languages and thus agent model-checkers extended in a straightforward fashion to reason about them.

As described above, the AIL/AJPF verification framework can, in principle, verify both AgentSpeak and METATEM programs. The latter already has built in capabilities to reason about languages involving Content and Context.

5 Concluding Remarks

There have been some approaches to the formalisation of pervasive systems, for example (Birkedal et al., 2006; Ranganathan and Campbell, 2008; Arapinis et al., 2009; Bakhouya et al., 2012). In addition,

significant aspects of the overall verification problem have already been tackled, for example *probabilistic* verification (Kwiatkowska et al., 2002), analysis of *context* (Boytsov and Zaslavsky, 2011), stochastic resource analysis (Gorrieri et al., 2002), and the verification of *autonomy* (Dennis et al., 2012). However, few of these have considered both specification *and* verification, and none have incorporated all the different aspects we describe here.

In this article, we have identified four orthogonal views and have tackled the verification of each of these independently, and each to a different extent. Below we give a summary of our work carried out so far:

1. *combinations of logics*: theoretical work on algorithms for model checking combined logics (Konur et al., 2013);
2. *autonomous decision-making*: characterising rational agents, and then developing agent model checking (Dennis et al., 2012) for analysing autonomous decision-making (Fisher et al., 2013);
3. *human-agent teamwork*: developing a verification method the Brahms workflow language (Stocker et al., 2011, 2012) and its application to human-robot teamwork; and
4. *context based computation*:
   - probabilistic verification of context-based systems (Konur et al., 2014), and
   - extending agent model checking to agent languages with context (Dennis et al., 2008; Fisher and Hepple, 2009).

At present the verification of all these aspects is carried out separately; in the future we will work on integrating all these efforts into a holistic verification framework for pervasive systems. This is a difficult problem: combining techniques for populations and individuals, discrete logic and feedback control, abstracted human behaviour and stochastic activity, etc, requires a sophisticated framework and it may well be that the complexity of this approach requires simplification and limitations on potential interactions.

References

Arapinis, M., Calder, M., Dennis, L. A., Fisher, M., Gray, P. D., Konur, S., Miller, A., Ritter, E., Ryan, M., Schewe, S., Unsworth, C. and Yasmin, R. (2009), 'Towards the Verification of Pervasive Systems', *ECEASST* **22**.

Bakhouya, M., Campbell, R., Coronato, A., de Pietro, G. and Ranganathan, A. (2012), 'Introduction to special section on formal methods in pervasive computing', *ACM Transactions on Autonomous and Adaptive Systems* **7**(1).

Birkedal, L., Bundgaard, M., Damgaard, T. C., Debois, S., Elsborg, E., Glenstrup, A. J., Hildebr, T., Milner, R. and Niss, H. (2006), Bigraphical Programming Languages for Pervasive Computing, *in* 'Proc. International Workshop on Combining Theory and Systems Building in Pervasive Computing'.

Bordini, R. H., Dastani, M., Dix, J. and El Fallah Seghrouchni, A., eds (2005), *Multi-Agent Programming: Languages, Platforms and Applications*, Springer.

Bordini, R. H., Dastani, M., Dix, J. and El Fallah-Seghrouchni, A., eds (2009), *Multi-Agent Programming: Languages, Tools and Applications*, Springer.

Bordini, R. H., Dennis, L. A., Farwer, B. and Fisher, M. (2008), Automated Verification of Multi-Agent Programs, *in* 'Proc. 23rd IEEE/ACM Int. Conf. Automated Software Engineering (ASE)', pp. 69–78.

Bordini, R. H., Fisher, M., Visser, W. and Wooldridge, M. (2006), 'Verifying Multi-Agent Programs by Model Checking', *Journal of Autonomous Agents and Multi-Agent Systems* **12**(2), 239–256.

Bordini, R. H., Hübner, J. F. and Vieira, R. (2005), Jason and the Golden Fleece of Agent-Oriented Programming, *in* Bordini, Dastani, Dix and El Fallah Seghrouchni (2005), chapter 1, pp. 3–37.

Boytsov, A. and Zaslavsky, A. (2011), Formal Verification of the Context Model — Enhanced Context Spaces Theory Approach, Technical report, Lulea University of Technology, Department of Computer Science, Space and Electrical Engineering, SE-971 87, Lulea, Sweden.

Chen, H., Finin, T. and Joshi, A. (2003), 'An Ontology for Context-aware Pervasive Computing Environments', *Knowledge Engineering Review* . (Special Issue on Ontologies for Distributed Systems).

Clancey, W., Sierhuis, M., Kaskiris, C. and van Hoof, R. (2003), Advantages of Brahms for Specifying and Implementing a Multiagent Human-Robotic Exploration System, *in* 'Proceedings of the Sixteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS)', AAAI Press, pp. 7–11.

Clarke, E. M., Grumberg, O. and Peled, D. A. (1999), *Model Checking*, MIT Press.

Dastani, M., van Riemsdijk, M. B. and Meyer, J.-J. C. (2005), Programming multi-agent systems in 3APL, *in* Bordini, Dastani, Dix and El Fallah Seghrouchni (2005), chapter 2, pp. 39–67.

Dennis, L. A. and Farwer, B. (2008), Gwendolen: A BDI Language for Verifiable Agents, *in* B. Löwe, ed., 'Proc. AISB'08 Workshop on Logic and the Simulation of Interaction and Reasoning', AISB, Aberdeen.

Dennis, L. A., Fisher, M. and Hepple, A. (2008), Language Constructs for Multi-Agent Programming, *in* 'Proc. 8th Workshop on Computational Logic in Multi-Agent Systems (CLIMA)', Vol. 5056 of *LNAI*, Springer, pp. 137–156.

Dennis, L. A., Fisher, M., Webster, M. and Bordini, R. H. (2012), 'Model Checking Agent Programming Languages', *Automated Software Engineering* **19**(1), 5–63.

Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N. and Zambonelli, F. (2006), 'A Survey of Autonomic Communications', *ACM Transactions on Autonomous and Adaptive Systems* **1**(2), 223–259.

Dobson, S., Sterritt, R., Nixon, P. and Hinchey, M. (2010), 'Fulfilling the Vision of Autonomic Computing', *IEEE Computer* **43**(1), 35–41.

Finger, M. and Gabbay, D. M. (1996), 'Combining Temporal Logic Systems', *Notre Dame Journal of Formal Logic* **37**(2), 204–232.

Fisher, M., Dennis, L. and Webster, M. (2013), 'Verifying Autonomous Systems', *Commun. ACM* **56**(9), 84–93.

Fisher, M. and Hepple, A. (2009), Executing Logical Agent Specifications, *in* R. H. Bordini, M. Dastani, J. Dix and A. El Fallah-Seghrouchni, eds, 'Multi-Agent Programming: Languages, Tools and Applications', Springer, pp. 1–27.

Fisher, M., Singh, M., Spears, D. and Wooldridge, M. (2007), 'Guest Editorial: Logic-Based Agent Verification', *Journal of Applied Logic* **5**(2), 193–195.

Gabbay, D., Kurucz, A., Wolter, F. and Zakharyaschev, M. (2003), *Many-Dimensional Modal Logics: Theory and Applications*, number 148 *in* 'Studies in Logic and the Foundations of Mathematics', Elsevier Science.

Georgeff, M. P. and Lansky, A. L. (1987), Reactive reasoning and planning, *in* A. Press, ed., 'Proceedings of the 6th National Conference on Artificial Intelligence (AAAI)', Menlo Park, CA, USA, pp. 677–682.

Gorrieri, R., Herzog, U. and Hillston, J. (2002), 'Unified Specification and Performance Evaluation using Stochastic Process Algebras', *Performance Evaluation* **50**(2/3), 79–82.

Greenfield, A. (2006), *Everyware*, New Riders Publishing.

Henricksen, K. and Indulska, J. (2004), A software engineering framework for context-aware pervasive computing, *in* 'Proceedings 2nd IEEE Conf. on Pervasive Computing and Communications', pp. 77–86.

Hepple, A. (2010), Agents, Context, and Logic, PhD thesis, Department of Computer Science, University of Liverpool, UK.

Hepple, A., Dennis, L. A. and Fisher, M. (2008), A Common Basis for Agent Organisations in BDI Languages, *in* 'Proc. International Workshop on LAnguages, methodologies and Development tools for multi-agent systemS (LADS)', Vol. 5118 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 171–88.

Hindriks, K., de Boer, F., van der Hoek, W. and Meyer, J.-J. (2001), Agent programming with declarative goals, *in* 'Intelligent Agents VII', Vol. 1986 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Heidelberg, Germany, pp. 228–243.

Holzmann, G. (2003), *Spin Model Checker, the: Primer and Reference Manual*, first edn, Addison-Wesley Professional.

Hunter, J., Raimondi, F., Rungta, N. and Stocker, R. (2013), A Synergistic and Extensible Framework for Multi-Agent System Verification, *in* 'Proc. International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)', IFAAMAS, pp. 869–876.

Isern, D., Moreno, A., Sanchez, D., Hajnal, A., Pedone, G. and Varga, L. (2011), 'Agent-Based Execution of Personalised Home Care Treatments', *Applied Intelligence* **34**, 155–180.

Jongmans, S.-S. T. Q., Hindriks, K. V. and van Riemsdijk, M. B. (2010), Model Checking Agent Programs by Using the Program Interpreter, *in* 'Proc. 11th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA)', Vol. 6245 of *LNCS*, Springer, pp. 219–237.

Knox, S., Shannon, R., Coyle, L., Clear, A., Dobson, S., Quigley, A. and Nixon, P. (2008), 'Scatterbox: Context-Aware Message Management', *Revue d'Intelligence Artificielle* **22**(5), 549–568.

Konur, S. (2013), 'A survey on temporal logics for specifying and verifying real-time systems', *Frontiers of Computer Science* **7**(3), 370–403.

Konur, S., Fisher, M., Dobson, S. and Knox, S. (2014), 'Formal Verification of a Pervasive Messaging System', *Formal Aspects of Computing* **26**(4), 677–694.

Konur, S., Fisher, M. and Schewe, S. (2013), 'Combined model checking for temporal, probabilistic, and real-time logics', *Theoretical Computer Science* **503**(0), 61 – 88.

Kwiatkowska, M., Norman, G. and Parker, D. (2002), 'PRISM: Probabilistic Symbolic Model Checker', *Lecture Notes in Computer Science* **2324**, 200–204.

NASA (Astronaut-Robot-Team-Concept), 'NASA Astronaut Robot Partner'. `http://history.nasa.gov/DPT/DPT.htm`.

PRISM (2013), 'Probabilistic Symbolic Model Checker'. `http://www.cs.bham.ac.uk/~dxp/prism`.

Ranganathan, A. and Campbell, R. H. (2008), 'Provably correct pervasive computing environments', *IEEE International Conference on Pervasive Computing and Communications* **0**, 160–169.

Rao, A. S. (1996), AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language, *in* 'Proc. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, (MAAMAW)', Vol. 1038 of *LNAI*, Springer-Verlag, pp. 748–752.

Rao, A. S. and Georgeff, M. P. (1991), Modeling Agents within a BDI-Architecture, *in* 'Proc. International Conference on Principles of Knowledge Representation and Reasoning (KR)', Morgan Kaufmann.

Rao, A. S. and Georgeff, M. P. (1992), An Abstract Architecture for Rational Agents, *in* 'Proc. International Conference on Knowledge Representation and Reasoning (KR)', pp. 439–449.

Rao, A. S. and Georgeff, M. P. (1995), BDI agents: From theory to practice, *in* 'Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS)', IEEE Press, Washington, DC, USA, pp. 312–319.

RoboSafe (2013), 'Trustworthy Robotic Assistants Project'. `http://www.robosafe.org`.

Rosa, P. M. P., Dias, J. A., Lopes, I. M. C., Rodrigues, J. J. P. C. and Lin, K. (2012), An ubiquitous mobile multimedia system for events agenda, *in* 'WCNC', pp. 2103–2107.

Sierhuis, M. (2001), Modeling and Simulating Work Practice. BRAHMS: a multiagent modeling and simulation language for work system analysis and design, PhD thesis, Social Science and Informatics (SWI), University of Amsterdam, SIKS Dissertation Series No. 2001-10, Amsterdam, The Netherlands.

Sierhuis, M. (2006), Multiagent Modeling and Simulation in Human-Robot Mission Operations. (See `http://ic.arc.nasa.gov/ic/publications`).

Sierhuis, M., Bradshaw, J. M., Acquisti, A., Hoof, R. V., Jeffers, R. and Uszok, A. (2003), Human-Agent Teamwork and Adjustable Autonomy in Practice, *in* 'Proceedings of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)', Nara, Japan.

Sierhuis, M. and Clancey, W. J. (2002), 'Modeling and Simulating Work Practice: A Human-Centered Method for Work Systems Design', *IEEE Intelligent Systems* **17**(5).

Stocker, R., Dennis, L., Dixon, C. and Fisher, M. (2012), Verifying Brahms Human-robot Teamwork Models, *in* 'Proceedings of the 13th European Conference on Logics in Artificial Intelligence', JELIA'12, Springer-Verlag, pp. 385–397.

Stocker, R., Sierhuis, M., Dennis, L., Dixon, C. and Fisher, M. (2011), A Formal Semantics for Brahms, *in* 'Proceedings of the 12th International Conference on Computational Logic in Multi-agent Systems', CLIMA'11, Springer-Verlag, pp. 259–274.

Strang, T. and Popien, C. L. (2004), A Context Modeling Survey, *in* 'Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing'.

Turner, K. J. (2012), *Advances in Home Care Technologies: Results of the MATCH Project*, IOS Press.

van Hoof, R. (2000), 'Brahms website: `http://www.agentisolutions.com`'.

Visser, W., Havelund, K., Brat, G. and Park, S. (2000), Model checking programs, *in* 'Proc. 15th Int. Conf. Automated Software Engineering (ASE)', IEEE Computer Society, pp. 3–12.

Want, R., Pering, T., Borriello, G. and Farkas, K. (2002), 'Disappearing Hardware', *Pervasive Computing* **1**(1).

Weiser, M. (1993), 'Some Computer Science Issues in Ubiquitous Computing', *Communications of the ACM* **36**(7), 74–84.

Wooldridge, M. (2002), *An Introduction to Multiagent Systems*, John Wiley & Sons.