Specification and Analysis of Context-Aware Systems

MPhil Thesis

Author
Bilel Remmache

Supervisors
Dr Corina Cîrstea
Prof. Vladimiro Sassone

Dependable Systems and Software Engineering
School of Electronics and Computer Science
University of Southampton

September 30, 2009

Abstract

In this report, an overview of research into context-awareness modelling is presented. We also give our approach to context-awareness modelling in the form of a process calculus called $C\pi$ (Context π). In this calculus, each process has an explicit representation of its context in the form of `(information, entity, value)` triples modelling the knowledge this process has about its environment. A process equiped with a context representation is called an *agent*. Agents make use of broadcast actions to exchange context information and make use of conditions on their context to perform *context-dependent* behaviour.

Acknowledgement

Many thanks to Dr Corina Cîrstea and Prof. Vladimiro Sassone for their ideas, help, and support.

Contents

Chapter 1

Introduction

The advancement of computing in the last decade has led to an explosion in the number of computing devices used. These devices are becoming smaller, mobile, more powerful, and more interconnected. This trend is likely to continue sparking a new era in computing where devices are embedded into the daily life activities of people. The concept of communicating computing devices located everywhere has been termed *Ubiquitous Computing* or *Pervasive Computing*. Applications with a Ubiquitous Computing flavour have already emerged. Examples include wearable devices to collect medical data, the smart home, the smart shirt [60], RFID tags, and the Autograph system [2].

Systems in industry are usually built in an adhoc manner, resolving problems only after they occur. Because of the large-scale networks of devices ubiquitous computing entails, the complexity of the required systems (software and hardware) is huge. Ubiquitous computers operate in highly distributed environments, share data, possess knowledge of their suroundings, and must work at low power in a timely manner. Therefore, designing and building such systems has to adhere to strict models to gurantee their safe and efficient operation.

The aim of the grand challenge of *Science for Global Ubiquitous Computing* [39] is to analyse and understand how these highly distributed, highly dynamic systems operate using *theoretical models of computation*. Examples of the problems tackled as part of this grand challenge are [48, 15]:

Space and mobility: a ubiquitous computing system can have components in different locations. Locations as well as movement of components between locations should be represented in any model of ubiquitous systems.

Hybrid systems: hybrid systems which are systems with both discrete and continuous components are common in real world applications. Examples of continuous components include sensor data and time.

Security and privacy: systems need to be designed with immunity against security attacks in mind. Safe languages are required, which allow verification of security using techniques such as model checking.

Stochastics: the incorporation of probabilistic modelling and numerical methods is important if many applications of ubiquitous computing are to be considered.

Boundaries, resources, and trust: we need to model allocation and deallocation of resources. Also, modelling of trust for the purposes of resource access and resource sharing is essential.

Context-awareness: devices operating in ubiquitous environments need to adapt to changing context such as location, collection of nearby people and accessible services. Context *representation*, *access*, and *effects* of context change on a system are examples of issues to be considered.

Our main focus will be on the problem of *context-awareness*. However, other issues such as space, mobility, resources, and privacy are also aspects of this problem [61]. So, our work will, in addition, include dealing with these aspects.

1.1 Motivation

Definition 1.1.1 (Context)**.** *Any information that can be used to characterise the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themeselves. Context is typically the location identity and state of people, groups and computational and physical objects* [23]

The issue of context-awareness in ubiquitous computing stems from the fact that ubiquitous entities are aware of the environment they operate in. This includes interactions with other entities, people, and any factors relevant to the activities of the system. The result of this awareness is that the system can reconfigure itself making its operation more efficient, reliable, and user friendly.

Some usage scenarios of context-aware systems include [10]:

Proactive triggering Applications record the context and trigger actions according to this context. Contexts recorded in the past might be matched to the current context triggering specific information to be displayed or actions to be performed. This can be used in assistive technologies. For example, the system can display food recipes depending the user's health, doctor's advice, available ingredients, and so on.

Streamlining interaction Humans implicitly use context when they talk to each other, which increases the conversation bandwidth. Humans can also benefit from the use of contextual information in their interaction with computers. The design of user interfaces can be simplified by taking advantage of the context the user is in.

Memory for the past The ability to save contextual information and query it at a later time can help users (especially those with weak memories) easily remember past events. An example of such use is the memory prosthesis work done at Xerox in 1994 [40].

Reminders for future contexts The user can attach a reminder to some possible future context. For example, 'when I meet person X, tell him...'.

1.2 Problem

In providing a model for context-awareness, a number of aspects should be taken into account. These aspects are mostly the result of the heterogeneity, unreliability, and sensitivity of context information. Examples of these aspects include:

- **Context representation:** It is important to have a representation of context that is general enough to cover most applications.
- **Context access:** How is contextual information accessed and modified?
- **Privacy:** The more an application knows about its context, the more privacy is invaded. The user should be able to control the information collected and restrict access to it.
- **Rapid change of context:** Some contextual information might change rapidly (eg. location). A realistic model of context should take the capture and use of such information into account.
- **Reliability:** It is almost always impossible to have exact information about the context of an application. Probabilistic treatment of contextual data is essential for the development of real world applications.

To better understand these aspects and be able to prove the correctness of context-aware systems, a formal model is required. Therefore, our aim is to design a formal language that allows us to model and study context-aware systems. Our research will initially focus on answering the following questions:

1. How can context be represented?
2. How is contextual information accessed?
3. How can we model the effects of context on the system's behaviour?

Our vehicle in answering these questions will be the various process theories such as the π-calculus, the ambient calculus, and bigraphical reactive systems. These theories have proved very useful in studying mobile and ubiquitous systems at a high level of abstraction. Specifically, the following steps will be followed to achieve our aim:

1. Develop a process calculus for modelling context-awareness
2. Investigate the use of this calculus in modelling different context-aware applications
3. Develop a modal logic to reason about properties of context-aware systems such as privacy and reliability

Thesis layout

Chapter 2 gives an overview of a number of process theories as well as modal logics. In chapter 3, research into informal and formal models of context-awareness is reviewed. Chapter 4 presents a process calculus $C\pi$ for context-awareness with the aim of being a step towards a powerful model of context-awareness. Chapter 5 provides details of a logic based on the μ-calculus to describe the properties of context-aware systems. Finally, chapter 6 goes through an example use case to demonstrate how the calculus and the logic are used in modelling. A conclusion is given in chapter 7 together with future work.

Chapter 2

Background

In this chapter some background on a number of theories for modelling mobile and dynamic systems will be covered. We discuss some process algebraic theories, which are used extensively in researching mobile systems. Section 2.1.1 talks about the π-calculus. The *ambient calculus* is introduced in section 2.1.2, and *Bigraphical Reactive Systems (BRSs)* in section 2.1.3. In addition, a brief discussion of modal logics, which are languages used to describe properties of transition systems, will be given in section 2.2.

2.1 Mobility

Systems in the ubiquitous computing world are usually mobile in the sense that their location, either physical or virtual, changes. Also, an important part of context-aware systems is their ability to reconfigure their services according to their current location and the location of important elements in the environment. Because mobility is central to ubiquitous computing, being able to express it and reason about it is essential.

A number of process calculi have been created to add the concept of mobility to concurrent process languages. Two of the most important contributions have been the π-*calculus* [49, 50] created by Milner, Parrow, and Walker, and the *ambient calculus* [13] created by Cardelli and Gordon. In the next two sections we introduce both calculi and give examples of their use.

2.1.1 The π calculus

The π-calculus is a language for the specification of concurrent processes where the communication links between these processes change as they interact [53]. The movement of links in the virtual space of linked processes is the way mobility is modelled in the π-calculus. Mobility is expressed by the movement of links because the location of a process is determined by the links it has to other processes in the virtual space of processes i.e. "your neighbours are those you

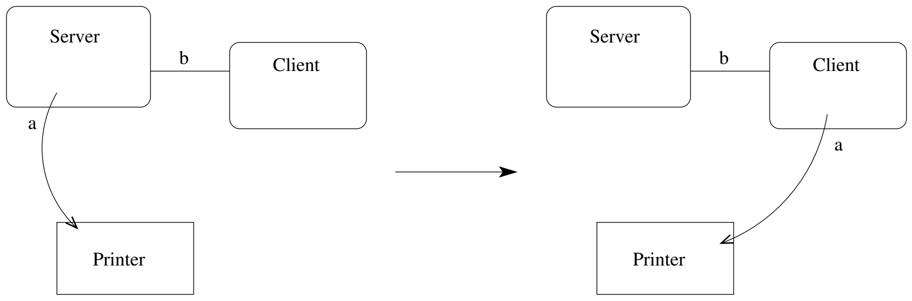


Figure 2.1: An example of link mobility

can talk to"[45]. The reason for adopting this meaning of mobility, as Milner argues in [45], is that it makes the calculus economical, flexible, and moderately simple.

To understand this treatment of mobility as movement of links between processes, consider the example of a server controlling a printer and passing access of the printer to a client process. Figure 2.1 shows links before and after the interaction. The server process sends the communication link a to the client process along channel b. If the only way to access the printer is through link a, then we can say that the printer *moves* to the client. This interaction is represented in the π-calculus using the transition

$$\bar{b}\langle a\rangle.S \mid b(c).\bar{c}\langle d\rangle.P \xrightarrow{\tau} S \mid \bar{a}\langle d\rangle.P$$

The syntax of a π-calculus process expression is given in table 2.1. Both the input action $a(x)$ and restriction $(\nu x)P$ bind the name x. Names are free in all other occurrences. In the π-calculus there is no difference between channel names and values. Channel names can be passed around in communication like any other values (as in the previous example). The restriction $(\nu x)P$ means that the name x is local to process P. In other words, it cannot be used for communication between P and its environment.

Structural congruence

Structural congruence equates processes that intuitively represent the same thing just from looking at their syntactic definitions. The structural congruence (relation $\equiv$) of the π-calculus is shown in table 2.2.

2.1.2 Mobile Ambients

An ambient is a bounded place where computation happens. The concept of an ambient is meant to be abstract in the sense that it can model not just a physical location but also things like an administrative domain, a web page, and a virtual address space [13].

Actions	$\alpha ::=$	$a(x)$	Input
		$\bar{a}\langle x\rangle$	Output
		τ	Silent
Processes	$P ::=$	$\mathbf{0}$	Nil
		$\alpha.P$	Prefix
		$P_1 + P_2$	Sum
		$P_1 \vert P_2$	Parallel
		$(\nu x)P$	Restriction
		$!P$	Replication

Table 2.1: Syntax of the π-calculus

$$P \equiv P\{a/b\} \quad \text{where } a \in \mathrm{bn}(P) \wedge b \notin \mathrm{fn}(P)$$
$$P + Q \equiv Q + P$$
$$P|0 \equiv P, P|Q \equiv Q|P, P|(Q|R) \equiv (P|Q)|R$$
$$(\nu x)(P|Q) \equiv P|(\nu x)Q \quad \text{if } x \notin \mathrm{fn}(P)$$
$$(\nu x)\mathbf{0} \equiv \mathbf{0}$$
$$(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$$
$$!P \equiv P|!P$$

Table 2.2: Structural congruence for the π-calculus

Capabilities	$M ::=$	*in n*	Can enter n
		out n	Can exit n
		open n	Can open n
Processes	$P ::=$	$(\nu n)P$	Restriction
		$\mathbf{0}$	Nil
		$P_1 \vert P_2$	Parallel
		$!P$	Replication
		$n[P]$	Ambient
		$M.P$	Capability
		$(x).P$	Input
		$\langle x \rangle.P$	Output

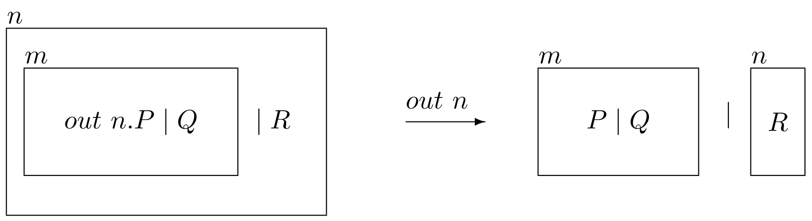
Table 2.3: The ambient calculus syntax

Ambients can be nested to form sub-ambients. Mobility is modelled in terms of ambients *exiting* parent ambients or *entering* sibling ambients. As ambients only represent a boudary, the movement is determined by processes running within the ambient. Opening a sibling ambient by disolving its boundary is another feature of the ambient calculus. The syntax of an ambient process is shown in table 2.3.

The ambient process *out* $n.P$ tells its enclosing ambient to exit its parent ambient n if this latter exists. This is modelled by the transition rule

$$n[m[\mathit{out}\ n.P \mid Q] \mid R] \rightarrow m[P \mid Q] \mid n[R]$$

which says that ambient m moves outside its parent n. Process R remains in n after the move, while the contents of m move with it. The action *out* n is consumed. Graphically, the transition can be represented as in the following diagram:



For a list of transition rules for all actions, refer to [13]. As an example, consider the process

$$m_1[n[\mathit{out}\ m_1.\mathit{in}\ m_2.\langle M \rangle] \mid P] \mid m_2[\mathit{open}\ n.(x).Q]$$

This can model a message M sent from one machine m_1 to another m_2 in the network. The message is enclosed in the ambient n, which is dissolved by

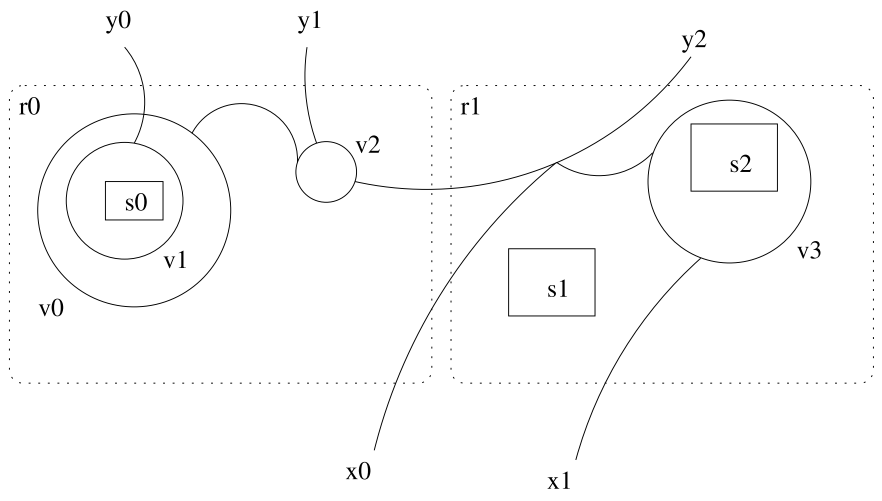


Figure 2.2: Example of a bigraph[47]

processes running inside the second machine. The transitions performed are as follows

$$\begin{aligned}
& m_1[n[\mathit{out}\ m_1.\mathit{in}\ m_2.\langle M\rangle] \mid P] \mid m_2[\mathit{open}\ n.(x).Q] \\
\xrightarrow{\mathit{out}\ m_1}\ & m_1[P] \mid n[\mathit{in}\ m_2.\langle M\rangle] \mid m_2[\mathit{open}\ n.(x).Q] \\
\xrightarrow{\mathit{in}\ m_2}\ & m_1[P] \mid m_2[n[\langle M\rangle] \mid \mathit{open}\ n.(x).Q] \\
\xrightarrow{\mathit{open}\ n}\ & m_1[P] \mid m_2[\langle M\rangle \mid (x).Q] \\
\xrightarrow{\tau}\ & m_1[P] \mid m_2[Q\{x \leftarrow M\}]
\end{aligned}$$

The last transition is a communication between parallel processes running inside m_2 resulting from synchronisation between the output action $\langle M\rangle$ and the input action (x). Process Q continues execution with the name x bound to message M (signified by $Q\{x \leftarrow M\}$).

Structural congruence

The structural congruence for the ambient calculus is defined in table 2.4.

2.1.3 Bigraphical Reactive Systems

Bigraphical reactive systems (BRSs) are a graphical model of mobile communicating systems. The aim of this model is to unify previous models such as the π-calculus and the ambient calculus [46].

A bigraph (figure 2.2) is a combination of two structures: a *place* graph that is a forest and a *link* graph that is a hypergraph (figure 2.3). Both graphs

$$
\begin{array}{l}
P \equiv P \\
P \equiv Q \Rightarrow Q \equiv P \\
P \equiv Q, Q \equiv R \Rightarrow P \equiv R \\
P \equiv Q \Rightarrow (\nu n)P \equiv (\nu)Q \\
P \equiv Q \Rightarrow P|R \equiv Q|R \\
P \equiv Q \Rightarrow !P \equiv !Q \\
P \equiv Q \Rightarrow n[P] \equiv n[Q] \\
P \equiv Q \Rightarrow M.P \equiv M.Q \\
P \equiv Q \Rightarrow (n).P \equiv (n).Q \\
(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P \\
(\nu n)\mathbf{0} \equiv \mathbf{0} \\
(\nu n)(P|Q) \equiv P|(\nu n)Q \quad if\ n \notin \mathrm{fn}(P) \\
(\nu n)(m[P]) \equiv m[(\nu n)P] \quad if\ n \neq m \\
P|\mathbf{0} \equiv P \\
P|Q \equiv Q|P \\
(P|Q)|R \equiv P|(Q|R) \\
!\mathbf{0} \equiv \mathbf{0} \\
!(P|Q) \equiv !P|!Q \\
!P \equiv P|!P \\
!P \equiv !!P
\end{array}
$$

Table 2.4: Structural congruence for the ambient calculus

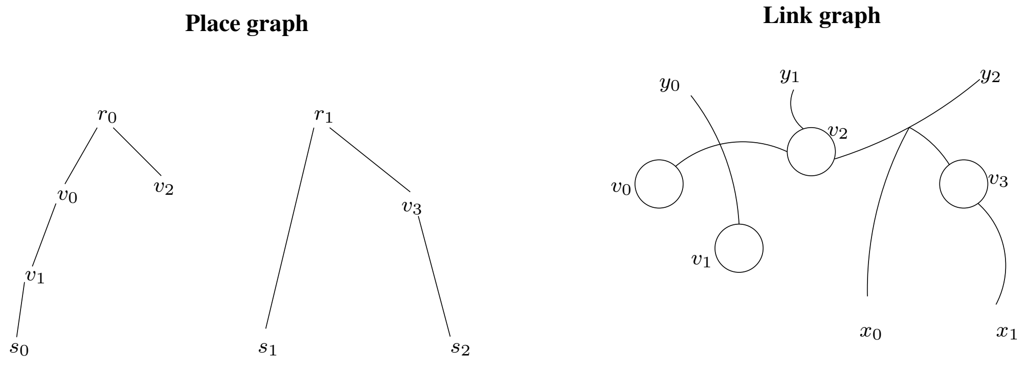


Figure 2.3: The place and link graphs of the bigraph in figure 2.2 [47]

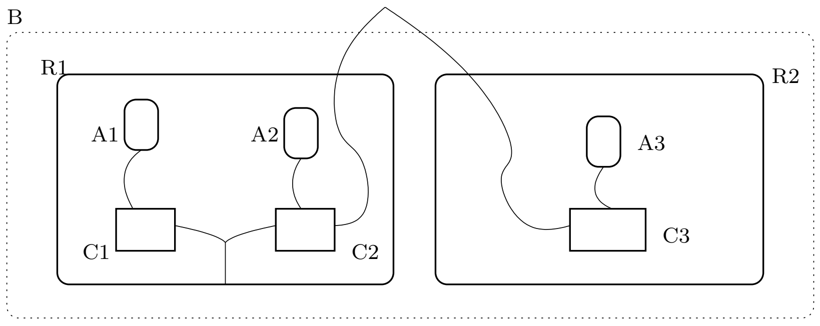


Figure 2.4: Modelling in bigraphs

have the same set of nodes ($\{v_0, ..., v_3\}$ in the example). The place graph is a representation of locality and is a result of the hierarchical structure of nodes in the bigraph. The link graph gives a topographical representation of the links between node ports in the bigraph. The sets of names x_n and y_n allow a bigraph to be linked to other bigraphs [47].

As an example of the use of bigraphs in modelling systems, consider the bigraph shown in figure 2.4. This bigraph models a building B containing two rooms R. Inside the rooms, there are computers C and agents A. Each agent has access to a computer. The computers *C1* and *C2* are connected through a network local to the room, while *C2* and *C3* are connected through a wider network.

2.2 Modal logic

A modal logic is a logic with operators called *modalities* or *modal operators* for handling concepts like *it is necessary that* and *it is possible that.* The name modal comes from the fact that the operators express the mode or way the truth of the rest of the proposition is to be interpreted[70, p. 5]. Other modal operators can be defined such as *it is known to be true that*, *it was once true that*, and *it will be true that.*

Modal logics are very useful in describing and reasoning about *relational structures.* A relational structure is a tuple consisting of a non-empty set of worlds (also called states or situations) W and a number of relations on this set. Such structures are common in Computer Science and modal logics provide a simple way of accessing the information contained in them as well as describing their properties [6].

Labeled transition systems (LTSs) are a type of relational structure. An LTS consists of a set of states W and relations $R_a \subseteq W \times W$ for each a in the set of labels A. If the relations of an LTS are restricted to being *partial functions*, then the LTS is deterministic, otherwise it is non-deterministic[6]. LTSs are representations of dynamic systems where W is the set of states of the system and labels are *actions* or functions acting on the states. Using a modal logic, we can describe properties of such systems. For example, it is possible to write formulae like $\Box\psi$ to mean "in all next states, proposition ψ is true" and $\Diamond\psi$ to mean "in some next state, proposition ψ is true".

A general definition of a modal language can be formulated as follows.

Definition 2.2.1 (Modal Language)**.** *A* modal language $ML(\tau, \Phi)$ *is built from a set of propositions* Φ *and a similarity type* $\tau = (O, \rho)$ *where* O *is a set of modal operators and* $\rho : O \to \mathbb{N}$ *is a function giving, for each modal operator, its* arity *(number of arguments it can be applied to). The formulas of the modal language are given by the following BNF*

$$\phi := p \mid \bot \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \triangle(\phi_1, \ldots, \phi_{\rho(\triangle)})$$

Where p *ranges over elements of* Φ*.*

The *basic modal language*[6] has *one* unary modal operator (arity 1), often refered to as diamond ($\Diamond$).

For each modal operator we can define a dual operator in the same sense that $\forall$ is the dual of $\exists$. The dual of the operator $\triangle(\phi_1, \ldots, \phi_n)$ is defined as

$$\triangledown(\phi_1, \ldots, \phi_n) = \neg\triangle(\neg\phi_1, \ldots, \neg\phi_n)$$

The dual of the opeartor diamond ($\Diamond$) is often refered to as box ($\Box$) and is defined as $\Box\phi = \neg\Diamond\neg\phi$.

Many modal logics exist where different modal operators are used with different meanings. In the following sections, three examples of modal logics are discussed.

The modal logic K

The logic K named after Saul Kripke is a modal logic that can be used as a foundation for a variety of logics. The symbols of K include $\neg$, $\vee$, $\bot$, and the modal operator $\Box$. The other propositional connectives such as $\rightarrow$ (implication) and $\wedge$ can be derived from the main ones. The dual of $\Box$ is $\Diamond = \neg\Box\neg$. K has the following axioms and rules which can be used to derive the valid formulas in the logic [38, ch. 3]

(taut) all propositional tautologies

(dist) $\Box(\varphi \rightarrow \psi) \rightarrow \Box\varphi \rightarrow \Box\psi$

(mp) from $\varphi, \varphi \rightarrow \psi$ derive ψ

(nec) from ψ derive $\Box\psi$

Kripke semantics

To give a semantics or meaning to modal logic K, Kripke invented a model based on a graph structure.

Definition 2.2.2 (Kripke Model)**.** *A* Kripke model $\mathcal{M}$ *of the basic modal language is a triple* (W, R, L) *where* W *is a set whose elements are called* worlds*,* $R \subseteq W \times W$ *is a relation called the* accessibility relation*, and* $L : W \rightarrow \mathcal{P}(\Phi)$ *is a labelling function giving for each world, the set of atomic propositions (a subset of* Φ*) satisfied in it* [70].

The semantics of modal logic K is then given by the satisfaction relation $\models$ defined as follows [38, ch. 3]

$(W, R, L, x) \models p$ iff $p \in L(x)$

$(W, R, L, x) \not\models \bot$

$(W, R, L, x) \models \varphi \rightarrow \psi$ iff $(W, R, L, x) \models \varphi \Rightarrow (W, R, L, x) \models \psi$

$(W, R, L, x) \models \Box\varphi$ iff $\forall y \in W, xRy \Rightarrow (W, R, L, y) \models \varphi$

In general, the semantics of a modal language with n modalities is given by a structure $(W, R_1, \ldots, R_n)$ with one relation for each modality. A relation R of a modality $\triangle$ with arity k is a subset of

$$\underbrace{W \times W \times \cdots \times W}_{k+1 \textit{ times}}$$

Hennessy Milner Logic (HML)

Hennessy Milner Logic is a modal logic for specifying the properties of processes modelled as Labelled Transition Systems (LTSs). A LTS is described by a set of states $\mathcal{P}$ and relations $R_a \subseteq \mathcal{P} \times \mathcal{P}$ for each a in the set of labels A. The following BNF defines the syntax of formulas in HML where K is a set of labels (a subset of A).

$$\Phi ::= T \mid F \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [K]\Phi \mid \langle K \rangle \Phi$$

K can take the form $\{a, b, c\}$ where a, b, and c are labels. We omit the curly brackets when K appears inside $[\,]$ and $\langle\,\rangle$ and write for example $[a, b, c]$ instead of $[\{a, b, c\}]$.

Of particular interest are the two modalities $[K]\Phi$ and $\langle K \rangle \Phi$. The first is satisfied by a process P if every process it evolves to after performing a transition in K has the property Φ. The second is the dual of the first and means that a process Q exists to which P evolves after performing a transition in K. We write $P \models \Phi$ to mean that process P has (satisfies) property Φ. It should be noted that the negation of Φ denoted by $\neg\Phi$ can be defined inductively in HML. This is because each connective and modality has its dual. For example,

$$\begin{aligned} &\neg T = F \quad \text{and} \quad \neg F = T \\ &\neg(\Phi_1 \wedge \Phi_2) = \neg\Phi_1 \vee \neg\Phi_2 \\ &\neg[K]\Phi = \langle K \rangle \neg\Phi \end{aligned}$$

Given a LTS $(\mathcal{P}, A, (R_a)_{a \in A})$, the satisfaction relation $\models$ between processes and properties is defined inductively on the structure of formulas. We write $P \xrightarrow{a} P'$ to denote $(P, P') \in R_a$.

$$\begin{array}{lll} P \models T & & \\ P \not\models F & & \\ P \models \Phi_1 \wedge \Phi_2 & \text{iff} & P \models \Phi_1 \text{ and } P \models \Phi_2 \\ P \models \Phi_1 \vee \Phi_2 & \text{iff} & P \models \Phi_1 \text{ or } P \models \Phi_2 \\ P \models [K]\Phi & \text{iff} & \forall Q \in \{P' : P \xrightarrow{a} P' \text{ and } a \in K\}.Q \models \Phi \\ P \models \langle K \rangle \Phi & \text{iff} & \exists Q \in \{P' : P \xrightarrow{a} P' \text{ and } a \in K\}.Q \models \Phi \end{array}$$

Also, we define the abbreviations $\langle - \rangle \Phi$ and $[-]\Phi$ to mean $\langle A \rangle \Phi$ and $[A]\Phi$ respectively where A is the set of all labels.

Modal μ-calculus

HML cannot express recurring traits of processes but only local capabilities. Properties such as "whenever a request is made, a reply will eventually be sent back" and many other temporal logic properties cannot be expressed using modalities $[K]$ and $\langle K \rangle$ unless the system has a finite number of transitions. The modal μ-calculus extends HML by adding least and greatest fixed point operators. This provides the logic with more expressiveness which encompasses that of many other logics including LTL (Linear-time Temporal Logic) , CTL (Computation Tree Logic), and CTL* which captures both of the latter two [8].

The following formulas are added to the grammar defining HML to define the syntax of μ-calculus.

$$\Phi ::= Z \mid \nu Z.\Phi \mid \mu Z.\Phi$$

The semantics of μ-calculus is given on a LTS $(\mathcal{P}, A, (R_a)_{a\in A})$ where $\mathcal{P}$ is the set of all processes (states) and A is the set of labels. the semantics is based on a valuation function V, which interprets a variable Z as a set of processes E. To interpret a formula containing free variables $X, Y, Z, \ldots$, we need to know the meaning of these variables. When the formula has no free variables (closed), the valuation function is implicit (i.e does not have to be explicitly provided for the interpretation of the formula). The satisfaction relation for the μ-calculus is given by the following rules in addition to the HML rules.

$$\begin{array}{lll} P \models_V Z & \text{iff} & P \in V(Z) \\ P \models_V \nu Z.\Phi & \text{iff} & P \in \bigcup\{E \subseteq \mathcal{P} : E \subseteq \|\Phi\|_{V[E/Z]}\} \\ P \models_V \mu Z.\Phi & \text{iff} & P \in \bigcap\{E \subseteq \mathcal{P} : \|\Phi\|_{V[E/Z]} \subseteq E\} \end{array}$$

Where $\|\Phi\|_V$ is the subset of $\mathcal{P}$ which satisfy Φ under valuation V. We omit V from this notation when describing a property with no variables and write $\|\Phi\|$ to mean the set of all processes satisfying Φ.

A formula $\Phi(Z)$ containing variable Z can be viewed as a function $f : 2^{\mathcal{P}} \to 2^{\mathcal{P}}$ where $2^{\mathcal{P}}$ is the powerset of $\mathcal{P}$ i.e. the set of all subsets of $\mathcal{P}$. The variable Z is interpreted by a subset of the set of all processes $\mathcal{P}$. Assuming the lattice structure on $2^{\mathcal{P}}$ given by set inclusion and the monotonicity of f, we know by Knaster-Tarski theorem that least and greatest fixed points exist for f. The semantics of modality $\nu Z.\Phi(Z)$ is given as the greatest pre-fixed point of f whereas the semantics of $\mu Z.\Phi(Z)$ is the least pre-fixed point[1]. The reason for using these operators is to be able to give semantics to recursive formulas which provide a way of expressing the usual temporal logic properties.

Informally, the least and greatest fixed points can be obtained by iteratively applying the function f to the $\emptyset$ and the set of all processes $\mathcal{P}$ respectively. This is equivalent to starting the equation $X = \Phi(X)$ with F (false) and T (true) to find the least and greatest fixed points respectively.

For example[2], the temporal property "Always Φ" or $\mathbf{AG}\Phi$ in CTL can be thought of as a property X such that if X is true then Φ is true and whenever we move to the next state, X remains true. In other words, X satisfies the modal equation:

$$X = \Phi \wedge [-]X$$

[1]least and greatest fixed points conincide with the least and greatest pre-fixed points respectively because of the monotonicity of f

[2]In the next two examples in this section, we assume Φ does not contain X or Z

To find the greatest fixed point, we start with $X = T$. This gives the following iterations.

$$\begin{aligned}
&\|\Phi \wedge [-]T\| \quad \text{i.e.} \quad \|\Phi\| \\
&\|\Phi \wedge [-]\Phi\| \\
&\|\Phi \wedge [-](\Phi \wedge [-]\Phi)\| \quad \text{i.e.} \quad \|\Phi \wedge [-]\Phi \wedge [-][-]\Phi\| \\
&\text{and so on} \ldots
\end{aligned}$$

We can easily see that with an infinite number of applications, this will give the set of all processes for which Φ holds in all future states. This is exactly what the property $\mathbf{AG}\Phi$ describes. Therefore, the property above is equivalent to the μ-calculus property $\nu Z.\Phi \wedge [-]Z$. The least fixed point solution excludes infinite processes.

Also, the temporal property "There exists a path on which Φ eventually holds" ($\mathbf{EF}\Phi$ in CTL) can be viewed as the recursive property "X holds if either Φ holds now or X holds in one of the next states". The recursive modal equation expressing this property is:

$$X = \Phi \vee \langle - \rangle X$$

With similar reasoning to the first equation. We can find that the least fixed point solution to this equation is:

$$\|\Phi \vee \langle - \rangle \Phi \vee \langle - \rangle \langle - \rangle \Phi \vee \ldots\|$$

which represents the solution we are looking for. Therefore, the CTL property above is equivalent to the μ-calculus property $\mu Z.\Phi \vee \langle - \rangle Z$. The greatest fixed point solution also includes processes that perform actions forever without ever satisfying Φ.

To know which type of fixed point to use, a rule of thumb is that if we have to apply a test a finite number of times then the least fixed point should be used. Otherwise, the greatest fixed point should be used. For the first example above, the greatest fixed point was the correct choice as we have to apply the check $S \models \Phi$ for all states S of a process P to prove that $P \models \mathbf{AG}\Phi$ which could be infinite. However, in the second example, we need only apply the test a finite number of times until we find a state $S \models \Phi$. As a result, the least fixed point was the appropriate choice in this case [8].

Summary

In this chapter a number of formal languages for modelling mobile and ubiquitous systems were discussed. Usually, these languages do not provide high level constructs which make modelling specific types of systems such as context-aware ones easier. However, these languages represent a platform on which higher level languages could be built. On the other hand, modal logics can be used to express the properties of ubiquitous systems including context-aware ones. Examples of modal logics for ubiquitous systems include [14] for the ambient calculus, [51] for the π-calculus, and [19] for Bigraphical Reactive Systems.

Chapter 3

Literature Review

In this chapter, we present an overview of research in context-awareness. A number of formal and informal attempts at solving some of the issues of context-awareness are discussed. The aim of this chapter is not to be a complete account of the work done in this area, but a quick discussion of the design strategies followed by the researchers. In particular, we focus on the problem of modelling the different aspects of context-awareness such as the model of context used and the way context information is accessed. Issues relating to implementation and efficiency are not discussed.

Research in the theme of context-awareness has been active for over a decade now. Some of the questions usually investigated in this field of research are:

1. How do we (efficiently) capture context from (low-powered) sensors with limited capabilities? [17, 28, 18]

2. What constitute good applications of context-awareness? What applications can we develop to prove the usefulness of this technology? [37, 44, 31, 24]

3. What is context? What environmental information can be considered part of context? [62, 22, 21]

4. What is a good way of designing context-aware systems? The aim here is to reduce the complexity that plagues the development of such systems. A number of APIs and middleware solutions have been developed to answer this. Some of these are discussed in section 3.2.

5. How do we formally specify and verify context-aware systems? What should a good model of context-awareness capture? Section 3.3 discusses some of the models suggested in research.

6. How do we model context information? Taking into account that such information will be exchanged between different parties and should scale to describing low as well as high level information. Some of the models suggested in literature are investigated in section 3.1

Question 5 forms the central objective of our research. In this review we focus on models of context (question 6), middleware and APIs developed (question 4), and the formal models of context-awareness (question 5).

3.1 Models of context

In context-aware systems, context is usually represented explicitly using some concrete model. This model needs to take into account the following characteristics of context information:

- **Heterogeneity:** context information is of many different types
- **Distribution:** it can be obtained from distributed sources, so it should be aggregated
- **Uncertainty:** there is some doubt as to the correctness of the information

Popular choices for modelling context information are: key-value pairs, XML, and logic-based models. In the next few sections we discuss each of these choices.

3.1.1 Key-value models

This is the simplest choice for modelling context-awareness. Many file formats encode context information as meta-data using key-value pairs such as JPEG (using the Exif format [25]). This model also forms part of many existing communication protocols where information about clients and servers is exchanged in message headers. An example is the HTTP protocol which transmits, as part of a request for a web document, a header containing details of the client sending the request. A HTTP header might look like this:

```
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
Host: www.yahoo.com
Referer: http://www.google.com/search?hl=en&q=yahoo
```

Another example of key-value models is the one used in the Active Badge project [61] where context-triggered Unix commands are started with information about the context (badge owner, owner's office, . . .) supplied as environment variables. Key-value pair models are not good at modelling the relationships between different elements of context information. For instance, modelling hierarchical location is not straightforward.

3.1.2 XML models

A number of context models are based on RDF (Resource Description Framework) [66], which can be encoded in XML or in other formats such as N-triples [65]. RDF models information as triples of the form *(subject, predicate, object).*

The subject denotes a resource or an entity and the predicate expresses a relationship between the subject and the object. As an example, the information "Southampton has postcode SO" can be represented as a triple *(Southampton, has postcode, SO)*. This can be encoded in XML as follows:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:terms="http://purl.org/dc/terms/">
        <rdf:Description rdf:about="urn:cities:UK:Southampton">
                <terms:alternative>SO</terms:alternative>
        </rdf:Description>
</rdf:RDF>
```

The subject and predicate in RDF must have unique identifiers because RDF data is meant to be exchanged between computers. The subject can be *anonymous* in that it has no name. The predicate can also be used as a subject representing a relationship about which statements can be made.

Examples of context models based on RDF include UAProf (User Agent Profile) [68] and CC/PP (Composite Capabilities/Preferences Profile) [67]. Another model is CSCP (Comprehensive Structured Context Profiles) [29], which extends CC/PP by removing some of its restrictions (such as the two-level hierarchy). The aim of these formats is to allow the content served to mobile devices to be adapted based on the capabilities of these devices (eg. screen size and platform). Other examples of XML-based context models include CCML (Centaurus Capability Markup Language) [34] and ConteXtML [58].

3.1.3 Logic-based models

Context in logic systems is usually represented as a set of facts and inference rules. Context in logic was considered mainly by researchers in AI such as McCarthy in [43] and Buvač in [11]. In general, facts are related to contexts using a relation such as $c' : ist(c, p)$, which is taken to mean that proposition p is true in context c and that this is asserted in an outer context c'. For example, the assertion that "Holmes is a detective in the context of Sherlock Holmes stories" is written as:

$$c_0 : ist(contextof(\text{“Sherlock Holmes stories”}), \text{“Holmes is a detective”})$$

Where c_0 is the outer context [1].

A context c_1 is said to be more general than another context c_2 if c_2 contains all the information of c_1 and possibly more [1]. This is written as $c_1 \leq c_2$. An assertion $ist(c_1, p)$ can be *lifted* to a context c_2 if $c_1 \leq c_2$. Some functions usually defined on contexts are:

- $value(c, t)$: returns the value of term t in context c. For example,

$$value(contextof(\text{“football match”}), \text{“number of players”}) = 11$$

- $assuming(p, c)$: returns a context like c with proposition p assumed

Summary

In this section, some examples of context models were briefly discussed. Other models used in literature include: object-oriented, entity-relationship diagrams, and ontology-based. The reader is referred to [64] for a discussion of these.

A model of context is concerned with the structure of context data, which must be acquired in the first place. Acquisition of context data as well as its preprocessing are usually performed by a software layer called a *middleware*. The middleware abstracts the low level tasks of data collection and communication of context information between software components. It also allows programming context-aware applications using high-level constructs. The following section discusses a number of middleware solutions that aim to simplify the creation of context-aware applications.

3.2 Middleware approaches

Definition 3.2.1 (Middleware)**.** *A middleware is a general-purpose service that sits between platforms and applications. By platform, we mean a set of low-level services and processing elements defined by a processor architecture and and Operating System's API. [4]*

A lot of research has been devoted to the creation of middleware architectures to make it easier to create applications that react to changes in their context and make use of information in their environment. The aim is to make the development of such applications as transparent as possible, delegating the job of capturing the right information, interpreting it, and executing the corresponding task to the underlying middleware.

Programming languages are enriched with new syntax for specifying the type of information required, conditions on such information, and the processes to execute in case these conditions are satisfied.

Figure 3.1 represents a general structure of a context-awareness middleware. The application registers interest in being notified of particular events or specifies which processes are executed in what conditions. The middleware gathers data from sensors (physical or logical), performs some processing (aggregation, reasoning,. . .) to extract high level information from low level sensor data, stores the data in some form of database, and notifies interested applications of changes to the context.

The *sensing* element usually consists of multiple distributed sensing components due to the distributed nature of context information [64]. A number of middleware architectures (eg. EgoSpaces [33] and Fulcrum [7]) also consider applications to be sensing elements. In other words, applications are providers as well as consumers of context information as shown in figure 3.2.

Another approach to middleware design is to allow applications to issue commands whose interpretation is left to the underlying middleware. The middleware then adapts to different contexts such as varying resources and services. Therefore, applications don't usually need to worry about context changes. For

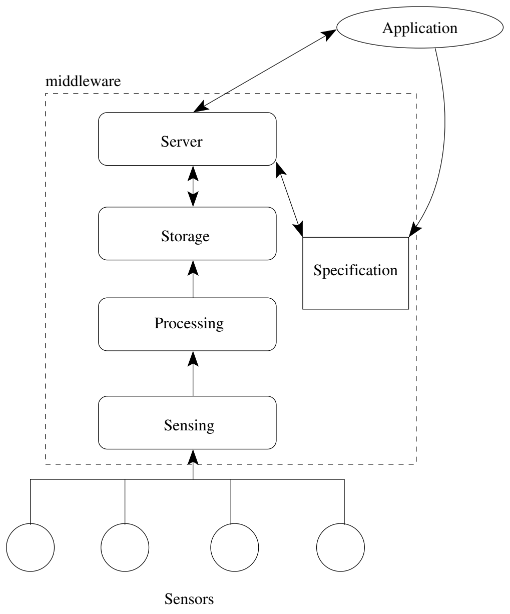


Figure 3.1: A general structure for a context-awareness middleware

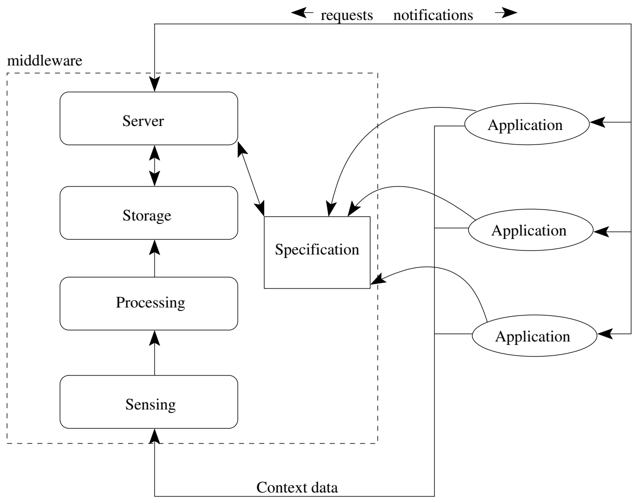


Figure 3.2: A middleware design where applications are context providers and users

instance, a `print` command can be interpreted as printing to the nearest printer by the middleware.

This approach does not require applications to be reimplemented to make use of context changes. However, it is more suitable in domain specific development where the middleware is merely a way of separating the context-aware part of an application and the context-unaware one. Examples of middleware designs that follow this approach include: CARMEN [3] and Satin [69].

The first approach makes the middleware simple but forces the applications to implement all the logic for adaptation. The second approach is not very flexible but the application can be completely ignorant of changes in context.

A third approach might be considered where the adaptation is possible at both the middleware and the application layers. Here, the middleware makes some decisions on behalf of the applications and leaves others to the application. MobiPADS [16] claims to follow such approach.

In the next sections, the following context-awareness middleware projects are discussed:

- Context Toolkit [59]
- EgoSpaces [33]
- Active Spaces and the GAIA middleware [57]
- CARMEN [3]
- CASS (Context-Awareness Sub-Structure) [26]
- Fulcrum [7]

3.2.1 The Context Toolkit

The authors build a library of components called *context widgets*, which can be used in monitoring the context. These components can then be used by applications to adapt to changing context. The widgets can also be composed to form other widgets in order to provide higher-level context information from low level data. Widgets can be thought of as distributed services that make use of a number of hardware or software sensors to monitor the context.

Applications can register their interest in being notified of changes to the context in order to adapt to those changes. Figure 3.3 shows examples of context widgets. The `IdentityPresence` widget monitors a specific location and notifies interested applications about the arrival and departure of people. The `Meeting` widget makes use of the `Activity` and `IdentityPresence` widgets to notify interested applications when a meeting in a specific location has started or finished based on the level of activity in that location and the number of people present. The Context Toolkit implements both synchronous access of context data (using attributes such as `location, identity`) and asynchronous (using notifications).

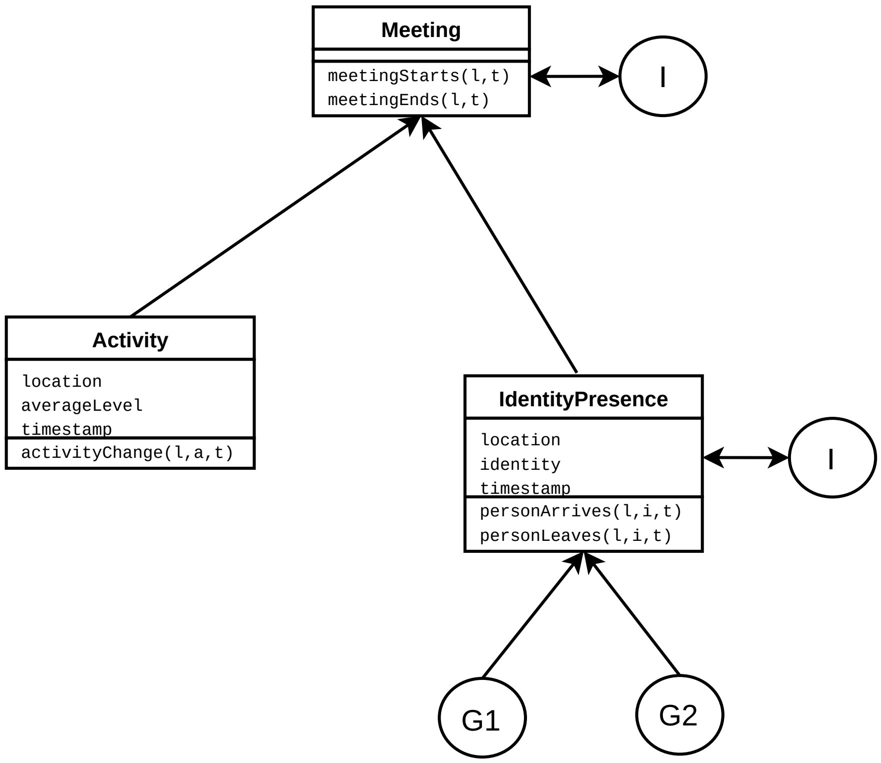


Figure 3.3: Example of widgets in the Context Toolkit [59]

A context widget usually has one or more context generators denoted by *G1* and *G2* in figure 3.3 as well as one or more interpreters (denoted by I). The generators are the source of raw context data, which can come from sensors in a low-level format not suitable for use by applications. Interpreters, on the other hand, provide tasks such as filtering uncertain data and abstracting to higher-level information. For example, the interpreter of the `Meeting` widget can decide that a meeting has started if the activity level is greater than 0.7 and two or more people are currently present in the room.

The Context Toolkit provides a good abstraction where context sensing is hidden in the context widgets. This should make the development of context-aware applications simpler. However, the toolkit requires a centralised registry which is queried for available widgets and interpreters. This is a downside because in mobile environments, a centralised service cannot always be guaranteed to exist. Other middleware architectures such as the Hydrogen project [30] and EgoSpaces [33] try to be more peer-to-peer.

3.2.2 EgoSpaces

The aim of the EgoSpaces middleware is to make programming context-aware systems simpler in mobile ad-hoc environments where the topology of the network changes dynamically and unpredictably. An example of such network is one forming on a motorway among vehicles in which a driver is warned if a vehicle is approaching.

The middleware models the network as a collection of *hosts*. Each host contains a collection of *agents*. Agents can migrate between hosts and each has its own local *tuple space*, which is a collection of tuples. A tuple in EgoSpaces is a set of triples of the form $(name, type, value)$. Each host has a *profile*, which is a private tuple used to hold the properties of the host. Properties of a host representing a vehicle might include the vehicle's type, speed, and direction. This is specified, for instance, using the tuple:

```
<(vehicle_type, enumeration, car),
 (direction, string, NORTH),
 (speed, integer, 65)>
```

The basic concept in the middleware is that of a *view*. A view is a set of tuples satisfying the view's constraints. The concept is derived from database theory where a view is a dynamic table that reflects the current result set of a query. An agent specifies a view in EgoSpaces by providing constraints on hosts, agents, and tuples. Constraints are specified using *patterns* where a pattern is a set of triples of the form $(name, type, constraint)$. For a tuple u to match a pattern p, for every triple (n, t, v) in u and triple (n, t, c) in p, the value v must satisfy the constraint c (i.e. $c(v) = true$). As an example, an agent might specify a view which includes only vehicles travelling in the same direction and at nearly the same speed as this agent's host. The following pattern can be used for this purpose:

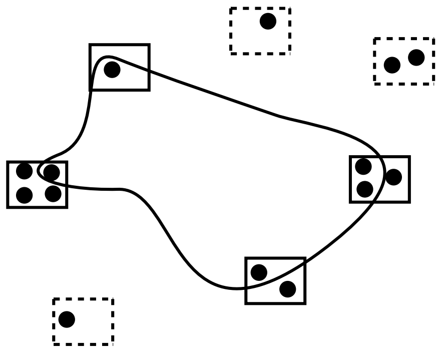


Figure 3.4: Example of a view in EgoSpaces[33]

```
<(direction, enumeration, =mydirection),
 (speed, integer, < myspeed + 2),
 (speed, integer, > myspeed - 2)>
```

An example of a constraint on agents might restrict the view to those agents of type $WeatherService$. A constraint on tuples can for example restrict the view to tuples with name *location*. Figure 3.4 shows a conceptual model of a view as an enclosure of different tuples from different hosts. Hosts not matching the view's host constraints are drawn with dotted borders.

For efficiency reasons, the middleware also requires a constraint on the network in the form of a distance D. A host is included in the search for matching tuples only if its distance to the reference host is less than D. A distance between two hosts is computed using a cost function provided by the application.

Applications using the middleware react to changes in context by first defining the views they are interested in and then performing view operations. Examples of view operations include:

- $rdp(v, p)$: return a copy of a tuple satisfying the view v and pattern p. If no such tuple exists return null.
- $inp(v, p)$: same as rdp but also deletes the matched tuple.
- $rdg(v, p)$: same as rdp but blocks until a matching tuple becomes available instead of returning null.
- $ing(v, p)$: same as rdg but also deletes the matched tuple.

3.2.3 GAIA

The basic concept in the GAIA middleware is that of an *active space*. An active space is a physical space equipped with a virtual computational space consisting of a GAIA process. This process manages movement of devices into and out of the active space while mapping resources to applications automatically.

The GAIA process allows applications to register their interest for context information and uses an event service to deliver high-level context (such as user behaviour and activities) to interested applications. Context information is supplied by *context providers*, a list of which is maintained using a registry accessible to applications.

The GAIA middleware is built for restricted networked environments that can be managed by centralised servers. This approach does not work well in settings where no central control exists.

3.2.4 CARMEN

The middleware for context-aware resource management CARMEN [3] aims at allowing services for wireless mobile devices to adapt the resources provided to clients to the changing *service provisioning context.* This context includes the client location, device capabilities, subscribed services, user preferences, and the level of trust. The middleware allows the specification of the service adaptation strategy through policies encoded in the Ponder policy specification language [20]. The *policies* make use of device, user, and site *profiles* to determine the context of the service and to specify which actions to perform in which contexts. The following is an example (taken from [3]) of a policy for resource binding. Resource binding is a mapping from names to resources. The binding should change in case the device moves to a new location in which case a resource can either be copied, moved, linked to, or a new binding to an equivalent resource is created.

```
inst oblig ResourceMovement {
    on DomainArrival(DeviceID, LocalityID);
    subject s = DeviceID.getServingProxy();
    target t = s.myContext;
    do t.setAgentBindingType("resource movement");
    when CARMEN.Monitoring.getFreeDiskSpace(DeviceID)
                > threshold;
}
```

The above profile specifies that on the movement of the device with identity `DeviceID` to a new locality `LocalityID`, the device's resource binding strategy should be one of moving resources on migration. This should be done only when there is enough free disk space on the device as reported by CARMEN's monitoring process.

Specifying adaptation to context changes using meta-data, although simple, might not give enough flexibility to the application because of the limited expressive power of the policy specification language. An approach where context adaptation is allowed in the application logic as well as using meta-data would allow more flexibility without compromising the simplicity of meta-data. Also, CARMEN is not a general purpose context-awareness middleware but is specific to wireless services.

3.2.5 CASS

The CASS (Context-Awareness Sub-Structure) middleware aims at allowing applications on mobile devices to react to changing context. The middleware provides an abstraction of context sensing and generation of high-level context. Applications specify their context dependent behaviour using a set of rules. Sensors, which are hosts capable of providing raw context data, send context data to a centralised server. The server abstracts the raw data and triggers the rules whose conditions are satisfied.

The reliance on a central server is a downside of CASS as in a lot of mobile applications, it is not guaranteed that such a server would exist in which case a truly distributed solution is required.

3.2.6 Fulcrum

The Fulcrum middleware [7] allows agents to access context information using event publishing and subscribing. An event consists of a set of elements of the form $(name, type, value)$. An agent wishing to receive notifications of events subscribes to such events using filters each of which is a set of constraints of the form $(name, type, operator, value)$, where $operator$ is a binary predicate.

The matching of filters and events is done by *brokers*, which are mediating services provided by the middleware. Fulcrum also allows a subscriber (for efficiency reasons) to subscribe for relationship events which are events linking two or more other events, allowing an agent to be notified when a value of some context information changes in a certain way with respect to some other information. For example, an agent can choose to be notified when person x is located within 10 meters of person y.

The approach of Fulcrum may seem similar to that of EgoSpaces as both middleware approaches use similar representations of context information (tuples). However, the exchange of this information is done differently. EgoSpaces uses a shared-data model where tuples live in a virtual persistent data store and are only removed explicitly by an agent. On the other hand, Fulcrum uses a publish-subscribe model where events have a limited life time (the event is discarded after being dispatched). The two approaches are discussed in more detail in section 4.1.

Summary

Many middleware projects have been udertaken in research to allow the implementation of context-aware applications. These middleware projects usually include similar components for sensing, processing, storage, and serving of context information. Some of the projects are aimed at specific application areas such as adhoc networks (eg. EgoSpaces) and mobile services (eg. CARMEN) while others aim to be more generic (eg. Context Toolkit).

A better understanding of these middleware approaches and their essential features requires studying the problem of context-awareness from a formal point of view. This also equips the developer with a method for proving the properties of context-aware systems such as security and privacy. This is important given that these systems manipulate information about people's daily life activities.

3.3 Formal models of context-awareness

A formal model consists of a syntax and formal semantics. One of the aims of formal modelling is to capture the interpretation (or meaning) of a set of systems using the formal syntax and semantics. In the case of context-awareness, we want to encode dynamic systems that react to changes in their environment using a notation together with an associated meaning given using one or more semantic theories such as operational, denotational, and categorical. It is important that the semantics capture the correct meaning of the system and that the syntax used to encode the system is straightforward. By this we mean the developer should be able to model a system with the minimum effort.

In this section, a number of attempts at formalising the concept of context-awareness are presented together with their main features and drawbacks. In particular, the following attempts will be discussed:

- Context Unity [55]
- Context Awareness Calculus (CAC) [71]
- The CONAWA calculus [36]
- Contextual Reactive Systems [9]
- Plato-graphical models [5]

3.3.1 Context Unity

Context Unity [55] is a formal model of context-awareness based on Mobile Unity [56]. The authors define what they termed the *context-aware paradigm* as a new design style that satisfies the following properties:

- **Expensiveness:** the design must take into account the fact that distant entities in the environment can affect a system's behaviour. In other words, it should not place a limit on the extent of the system's context.
- **Specificity:** the design must allow agents to specify what is part of their context and should allow such specifications to be modified.
- **Explicitness:** an agent should have an explicit context associated with it. In other words, what the agent views as context should be represented concretely using some form of data-structure.
- **Separability:** building of the context must be done separately from the rest of the system's behaviour.

- **Transparency:** maintenance of the agent's context should be taken care of by some underlying system (eg. a middleware). For this to be possible, the context must be defined at an adequate level of abstraction.

These properties make a distinction between a system that adapts to its environment and a system that follows the context-aware computing paradigm. The authors try to build a formal model which can be used to specify systems conforming to this design style. They also extend Mobile Unity's Hoare-style proof logic with a new rule to allow proving properties of these systems.

In Context Unity, a system consists of a community of interacting agents each of which can provide context information and make use of information provided by other agents in the system. The exchange of context information is done through two types of variables:

1. exposed variables which are shared between agents and are used by an agent to provide context information to other agents. Exposed variables are declared using the syntax $l!n : t$ where l is a local handle, n is a publicly accessible name, and t is the type of the variable.

2. context variables which contain data from the exposed variables of other agents and are used to acquire context information. Context variables are declared using the syntax $Q : t$ where Q is the name of the variable and t is its type. Context variables are local to the agent.

A separate context specification is used to define the context variables based on the desired properties of the exposed variables of other agents. Figure 3.5 shows an example of a Context Unity system with two agent types: `Agent1` and `Agent2` (defined using `Program` sections). These declarations are instantiated in the `Components` section where in this case two instances of `Agent1` and one instance of `Agent2` are run in parallel. `Agent1` declares two exposed variables: *id* of type `agent_id`, which stores a unique identifier for the agent and λ which stores the current location of the agent.

A context variable Q is declared as a set of agent ids. This context variable can be restricted to hold the set of identifiers of only those agents located within a predefined range. This is done in the `context` section, which specifies how context variables change with respect to the exposed variables of other agents. Context definitions ending with the word `reactive` are updated as soon as a change in exposed variables happens and before any other non-reactive statements.

The authors consider context-awareness as a design paradigm where the communication between a system and its context is abstracted. This abstraction allows the development of applications at a high-level without worrying about the details of the communication taking place to update the context model. In general, most middleware architectures and formal specifications seek a good abstraction to the different aspects of context-awareness (context update, sensing, context-dependency, inconsistency in context information, ...). However, as mentioned at the beginning of this chapter, there are other problems to be solved in this field of research.

```
System AcquaintanceManagement
   Program Agent1
      declare
         internal range : integer
         exposed id ! agent_id : agent_id
                 λ ! location : location
         context Q : set of agent_id
      initially
         range = 20
      assign
         definition of local behaviour
      context
         Q uses      l!location in a
            given    |l − λ| ≤ range
            where    Q becomes Q ∪ {a}
            reactive
         Q uses      l!location in a
            given    |l − λ| > range
            where    Q becomes Q − {a}
            reactive
   end
   Program Agent2
      declaration of Agent2 similarly to Agent1
   end
   Components
      Agent1[new_id], Agent1[new_id],
      Agent2[new_id]
end AcquaintanceManagement
```

Figure 3.5: Example of an application in Context Unity [55]

It is important to note here that a context-aware system should not be defined as one that conforms to the context-awareness design paradigm. In our opinion, what determines that a system is context-aware is whether its *behaviour* satisfies a given set of criteria. For example, a car agent is context-aware not because it was designed by the developer using a particular methodology, but because it behaves according to certain properties in certain situations. To call this agent context-aware, we might consider it sufficient that it warns the driver when another vehicle is approaching or we might require other properties in addition.

3.3.2 Context Awareness Calculus CAC

The Context Awareness Calculus [71] is an attempt to come up with a formal model of context-awareness based on mobile ambients [12] and the join calculus [27]. Ambients are extended with *ambient definitions*, which are rewrite rules mapping so called *atom* processes to other processes as in:

$$a(x\langle\bar{y}\rangle \triangleright P)[\ldots \mid x\langle\bar{y}\rangle \mid \ldots]$$

Whenever an atom $x\langle\bar{y}\rangle$ appears inside the ambient a (inside the square brackets [and]), it is translated or rewritten into process P. Ambient definitions are used to represent context-dependency, where an atom has different meanings in different ambients. As an example, consider the agent:

$$P \overset{def}{=} app()[print\langle letter\rangle]$$

and suppose this agent is executing in a setting where printing is done on a laser printer. We can represent this using:

$$a(print\langle x\rangle \triangleright send\langle x, laser\_printer\rangle)[P]$$

Which makes a transition to:

$$a(print\langle x\rangle \triangleright send\langle x, laser\_printer\rangle)[send\langle letter, laser\_printer\rangle]$$

Replacing the atom $print\langle letter\rangle$ in P with the *send* action in the rewrite rule of ambient a. A substitution of variable x with the name *letter* was also done. The same agent P will behave differently by printing to a inkjet printer if placed in ambient b as follows:

$$b(print\langle x\rangle \triangleright send\langle x, inkjet\_printer\rangle)[P]$$

The substitution of variables for names when applying the rewrite rules is the means by which agents communicate in CAC. This is achieved using substitutions in a similar way to the join calculus. For instance, in the following example:

$$c(x\langle z\rangle \parallel y\langle t\rangle \triangleright P \parallel Q)[a()[x\langle u\rangle] \mid b()[y\langle v\rangle]]$$

The variable z (respectively t) binds any free occurrences of z (respectively t) in both P and Q. So, the atoms $x\langle u\rangle$ and $y\langle v\rangle$ are rewritten into $P\{u/z, v/t\}$ and $Q\{u/z, v/t\}$ respectively. This way, name u is communicated to agent b (if z appears free in Q) and similarly name v is communicated to agent a (if t appears free in P).

The way context-awareness is modelled in CAC is by assuming that actions are interpreted differently in different contexts. A context of an agent in CAC is the nesting of ambient definitions giving a meaning to the atoms appearing in this agent. This resembles a model of dynamic binding, where the components of the system are loaded at run-time from the environment it is operating in. Although this is a characteristic of context-aware systems, CAC does not directly model a context-aware system that makes choices based on its context. Instead, CAC models a situation where agents have no awareness of their context as the enclosing environment is what controls their context-aware behaviour. This way of modelling context-awareness is similar to the approach discussed in section 3.2 that is taken by some middleware designs such as Satin [69]. In these designs, context-adaptation is done at the middleware level, so the application is largely unaware of changes in context.

Context change is not straightforward to represent in CAC. For example, we can model a mobile phone running in a context where the user is in a meeting as follows:

$$meeting(ring\langle\rangle \triangleright vibrate\langle\rangle)[phone()[P]]$$

To indicate that the context has changed, process P must either migrate to a context with a different name say *nomeeting*, or change the definition part of ambient *phone*. In the first case, the system should be modelled as containing both ambients *meeting* and *nomeeting* as follows:

$$meeting(ring\langle\rangle \triangleright vibrate\langle\rangle)[phone()[go(\uparrow nomeeting, Q)]] \mid \\ nomeeting(ring\langle\rangle \triangleright melody\langle\rangle)[\,]$$

In the second case, P can be of the form: $def(ring\langle\rangle \triangleright melody\langle\rangle)\ in\ Q$ which, after making a transition, changes the definition of ambient *phone*. In both cases, the change is initiated by the process whereas in reality the context change is usually outside the control of the system.

3.3.3 CONAWA Calculus

The CONAWA calculus [36] is another attempt to formally model context-awareness. The context is modelled as a collection of trees each of which represents a type of context information. For example a tree can be used to represent a hierarchy of `locations` while another one can be used for `printer types` as in figure 3.6.

An entity in CONAWA models an agent as an ambient with pointers into nodes in the context trees. This is used to indicate the current value of context information for this entity. For example, in figure 3.6, `P1` is an entity representing a printer of type `Postscript Colour` and located in the `Emergency Room` of the `Hospital`. Syntacticly, the two context trees are written as:

```
Location: [ EmergencyRoom[#P1] | OperatingRoom[] ]
PrinterType: [ PS[Colour[#P1] | BW[]] ]
```

Whereas, the entity `P1` is written for instance as:

```
P1: [out{Location}.in{Location}OperatingRoom]
```

The printer agent `P1` changes its location by moving in the context trees using the ambient capabilities *in* and *out*. Agents can move in more than one context tree at once by specifying the names of trees to perform the capability on.

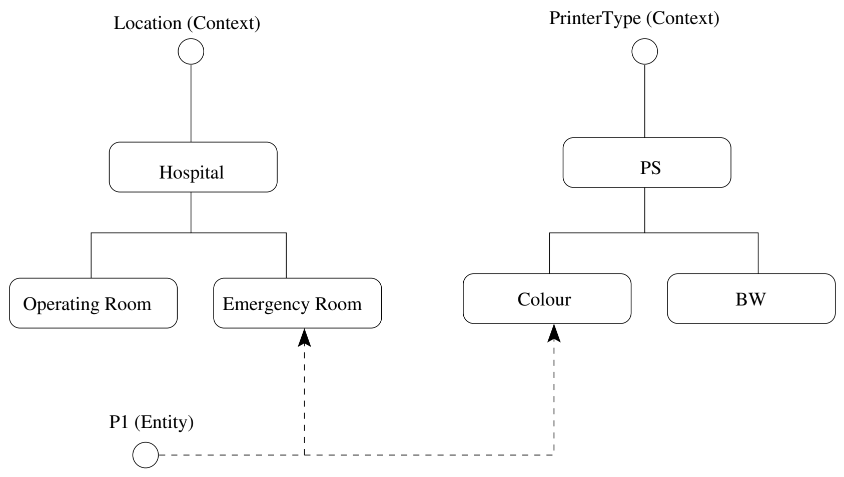


Figure 3.6: An example of trees represeting different types of context information in the CONAWA calculus [36]

For example, `out{Location&PrinterType}` will move the pointer of the agent outside of its current position in both the `Location` and `PrinterType` context trees at once.

Trying to model some types of context information in CONAWA is not straightforward. For example, context information that can take an infinite number of values such as `temperature` is not feasibly representable as ambients. To have an entity such as `patient1` with a pointer to context information of type `temperature`, one option is to have an infinite number of ambients each of which representing a different temperature value. A change in the entity's temperature can then be modelled as exiting the current ambient and entering the one representing the new value for temperature. Another option is to have only one ambient representing the current value of temperature in which case changing the temperature requires opening this ambient and creating a new one, which is not possible in CONAWA. Also, changes to context information such as temperature are outside the control of the application and therefore should be sensed but not affected by the entity itself.

3.3.4 Contextual Reactive Systems

In [9], the authors generalise Reactive Systems [41] into what they call *Contextual Reactive Systems*. In this generalisation, it is possible to specify as part of the interaction rules the contexts in which these rules can be applied and the contexts in which they cannot. A condition specifying when a rule is enabled is called an *enabler*, whereas one specifying when a rule is disabled is called an *inhibitor*. As an example, consider the following process expression consisting of an agent trying to print a document and two printers, one of type laser $\langle pr : las \rangle$

and the other of type inkjet $\langle pr : ink \rangle$:

$$\langle pr : las \rangle \mid \langle pr : ink \rangle \mid print(txt).0$$

This expression has two possible transitions each of which creates a job for one of the two available printers as follows:

$$\langle pr : las \rangle \mid \langle pr : ink \rangle \mid print(txt).0 \to \langle pr : las \rangle \mid \langle pr : ink \rangle \mid 0 \mid \langle job, txt, ink \rangle$$
$$\langle pr : las \rangle \mid \langle pr : ink \rangle \mid print(txt).0 \to \langle pr : las \rangle \mid \langle pr : ink \rangle \mid 0 \mid \langle job, txt, las \rangle$$

How do we specify that the agent prints to the inkjet printer only in case no laser printer is available? This is not possible in conventional reactive systems. In contextual reactive systems, however, we can specify that the first transition can only occur in case the context (terms running in parallel) does not contain a laser printer. This is done by writing:

$$\{- \mid \langle v \rangle . v \neq pr : las\}^* \quad print(txt).P \mid \langle pr : ink \rangle \to P \mid \langle job, txt, las \rangle \mid \langle pr : ink \rangle$$

Contextual reactive systems were used by the authors to model the LIME coordination middleware [52]. Modelling applications using these reactive systems requires specifying different rules for different applications. We think it would be easier to specify one set of rules which serves as the basis for a model of context-awareness. Applications would then be modelled using this set of rules.

3.3.5 Bigraphical Models of Context-Aware Systems

The authors in [5] discuss *Plato-graphical models* as a model of context-awareness. A plato-graphical model consists of three bigraphical reactive systems $\mathbf{C}, \mathbf{P}, \mathbf{A}$[1]. $\mathbf{C}$ is called a context and models the environment of the system. $\mathbf{P}$ is the *proxy* and is the system's representation of its context. Sensing and affecting the environment of the system is done only through the proxy. $\mathbf{A}$ represents the agents or the computational component of the system. Definition 3.3.1 is a formal definition of a plato-graphical model.

Definition 3.3.1 (Plato-graphical Model)**.** *A plato-graphical model is a triple* $(\mathbf{C}, \mathbf{P}, \mathbf{A})$ *of bigraphical reactive systems such that* $\mathbf{C} \cup \mathbf{P} \cup \mathbf{A}$ *is itself a bigraphical reactive system and* $\mathbf{C} \perp \mathbf{A}$[2].[5]

As an example of how a context-aware system is represented using a plato-graphical model, consider a print server that manages a collection of postscript and raw printers[3]. A job is sent to a raw printer only if there is no postscript printer in the collection of managed printers. The context $\mathbf{C}$ consists of nested locations *loc* within which printers reside. In modelling $\mathbf{C}$ we need rules for

[1]See section 2.1.3 for a discussion of bigraphical reactive systems

[2]$\mathbf{C} \perp \mathbf{A}$ means that the set of controls of $\mathbf{C}$ is disjoint from that of $\mathbf{A}$. Controls are the types of nodes in a bigraph

[3]Example taken from [5]

adding new printers to the context, removing existing ones, and removing serviced jobs. These rules are as follows:

$loc(-_0) \to loc(-_0 \mid /x.prt_x(raw))$	raw printer added
$loc(-_0) \to loc(-_0 \mid /x.prt_x(ps))$	postscript printer added
$loc(-_0 \mid prt_x(-_1)) \to loc(-_0) \mid x/$	printer removed
$prt_x(dat_z \mid -_0) \to prt_x(-_0) \mid z/$	job serviced

The proxy **P**, on the other hand, tracks the list of known printers in tables *prts*(*raw*) and *prts*(*ps*) and the list of pending jobs in *jobs*. It sends jobs to postscript printers by default, but in case there is no postscript printer, it sends the job to a raw one. The reaction rules are as follows:

$$\begin{aligned} &jobs(doc_z \mid -_0) \parallel prts_y(ps) \parallel prt_y(ps) \to \\ &\qquad jobs(-_0) \parallel prts_y(ps) \parallel prt_y(ps \mid dat_z) \end{aligned} \tag{3.1}$$

$$\begin{aligned} &jobs(doc_z \mid -_0) \parallel /x.prts_x(ps) \mid prts_y(raw) \parallel prt_y(raw) \to \\ &\qquad jobs(-_0) \parallel /x.prts_x(ps) \mid prts_y(raw) \parallel prt_y(raw \mid dat_z) \end{aligned} \tag{3.2}$$

$$/x.prt_x(ps) \parallel prts_y(ps) \to prt_y(ps) \parallel prts_y(ps) \tag{3.3}$$

$$/x.prt_x(raw) \parallel prts_y(raw) \to prt_y(raw) \parallel prts_y(raw) \tag{3.4}$$

Rule 3.1 represents printing to a postscript printer when one is known to exist. Rule 3.2 models the case when no postscript printer is known to exist and at least one raw printer exists. Finally, rules 3.3 and 3.4 model detecting new printers and adding them to the list of known ones.

The agents part of the model **A** has only one rule, which allows adding new jobs as follows:

$$jobs(-_0) \to jobs(-_0 \mid /z.doc_z)$$

The essence of plato-graphical models is the separation of the context dependent behaviour (the proxy **P**) from the context-independent one (the agents part **A**). In fact, it is possible to model both the proxy and the agents parts as one bigraph.

Plato-graphical models require the specification of different rules for each context-aware application. We think that bigraphical reactive systems should be used to specify calculi for context-awareness in the same way contextual reactive systems were used in [9] to model middleware for context-awareness. This should simplify the task of specifying context-aware systems by having one set of rules representing the semantics of context-awareness.

Also, plato-graphical models do not take into account the interaction between context-aware systems. In other words, the theory does not model how context information is exchanged between context-aware systems and how the context model of each system is updated. In the print server example, it is not clear how information about adding and removing printers would be obtained and how a system that provides such information would interact with the print server.

3.4 Conclusion

In this chapter, various issues arising in research on context-aware systems were discussed. In terms of how context information is represented, many middleware approaches have taken RDF and XML as the format of choice. We share this view mainly because XML and RDF have proved to be very useful in the exchange of information between heterogeneous systems in addition to being ubiquitous, platform-independent, and extensible. Although choosing a context model is an important decision, the method by which context information is exchanged between computational entities is equally important. A number of approaches were discussed, which can be grouped into:

1. **Publish-subscribe**: agents subscribe to events representing changes in context (eg. person x entered room y). Notifications are then sent to subscribers to announce the occurrence of these events.

2. **Shared data-based**: agents share a data-structure (eg. tuple space) in which information is stored by some agents and consumed by others.

Some of the issues not treated by these middleware approaches include: privacy, uncertainty, and the varying quality of context information. Also, these approaches lack any formal basis, which makes them unsuitable for reasoning about the properties of context-aware systems.

A number of formal models of context-awareness were also presented. In terms of how context information is represented, the CAC calculus uses processes embedded in the rewrite rules. So, For example, the rule $a\langle x\rangle \triangleright P$ can be thought of as representing the current value of context information $a\langle x\rangle$ using the process P. The CONAWA calculus represents information as multiple trees each having a different type. This explicit representation of context information is in our opinion better than an implicit one using processes because our aim is to model context-awareness at a high-level of abstraction in which information plays a central role. As to the way information is updated, this is done through migration of ambients in CAC and the manipulation of pointers to tree nodes in CONAWA. In both of these methods, updates are *not* external to the context-aware application. We argue that some context information is more intuitive to think of as originating from some external entities. Examples include temperature, nearby people, and physical location. Other types of information (eg. screen resolution and font size) can be thought of as being controlled by the application itself.

Overall, a need can be identified for high-level formal models where context information and context update are represented explicitly in order to make the task of modelling context-aware applications easier. In the next chapter, we aim to address this need by giving a formal model in the form of a process calculus.

Chapter 4

A Formal Model of Context-Awareness

A context-aware system is usually defined as a system which reacts to changes in its environment. This reaction should have the goal of providing the best possible service to the user. Many people have researched context-aware systems in practical scenarios. However, little attention has been paid to formalising such a concept. A formalism of context-awareness will not just give us a better understanding of what context-awareness is about, but will also provide us with a tool to verify properties of context-aware systems. In this chapter, we describe a model of context and context-awareness using a new process calculus based on the π-calculus and called $C\pi$ (Context Pi).

4.1 Design of $C\pi$

When trying to model context-awareness, the most important parts to model are: context representation, how a context is updated, and how an agent reacts to changes in its context. A representation of context is called a context model. A number of context models were discussed in section 3.1. EgoSpaces (section 3.2.2) and Fulcrum (section 3.2.6) use sets of $(name, type, value)$ triples as a model of context. Whereas, Carmen (section 3.2.4) uses XML files to encode device configurations and user preferences.

A context-aware application is usually not interested in all information about its environment, but only in a subset of this information. Therefore, a method is required for an application to specify which information is relevant to its behaviour. We call this method a *context specification.* Using a context specification, a middleware can take care of updating the knowledge of applications. This is done either by maintaining a data-structure holding context information satisfying the context specification or by delivering *events* to applications notifying them of the changes that have occurred. In EgoSpaces, a context specification is called a *view*, whereas in Fulcrum, it is the set of *filters* used in subscribing to events. EgoSpaces maintains the set of tuples satisfying each view[1]. In Fulcrum, notifications are sent to applications based on their subscriptions. Applications might then want to store the current values of context information in some form of data-structure.

Applications react to context changes by querying the data-structure maintained by the middleware (as in EgoSpaces) or by reacting to events. Reacting to events could be either programmed as part of the behaviour of the application or specified separately using `(event, condition, action)` rules. The former approach is taken by Fulcrum and the Context Toolkit (section 3.2.1), while the latter is taken by Carmen (where rules are called *policies*) and the Active Badge system [61].

$C\pi$ models context as `(information, subject, value)` triples. The underlying model is graph-based and is similar to how RDF [42] represents information using `(subject, property, value)` triples[2]. The main difference is that in $C\pi$ there is no support for anonymous (or blank) nodes and values can be complex (eg. sets, lists, functions, ...). Also, unlike $C\pi$, RDF allows multiple triples with the same subject and property. The model of context in $C\pi$ is also similar to the one used in EgoSpaces. The only difference is that, in this latter, the subject of a tuple `(name, type, value)` is always the agent publishing the information to the tuple space, which makes talking about properties of properties difficult.

An agent in $C\pi$ models a context-aware system, which specifies context information it is interested in knowing and performs actions based on this information. An agent in $C\pi$ is composed of a context part C and a process part P. The context part holds all the context information currently known to the agent. As mentioned above, this information is modelled as triples (i, s, v). By including a triple (i, s, v) in its context, an agent is specifying interest in receiving updates to information i of subject s. So, a context in $C\pi$ serves two purposes: the first is to hold the current values of context information and the second is to specify which information the agent is interested in knowing. Section 4.2 formally defines what a context is.

4.2 A model of context

In order to talk about awareness of a context in a formal way, we need to formally define what a context is. The context of a particular agent is its view of the physical and computational environments, which consists primarily of identities, locations and activities as well as any other data elements such as temperature and pressure. As a consequence, context data is of various types. In particular, each context information element is usually attached to some entity and is in most situations meaningless without a reference to the entity

[1]Actually, EgoSpaces only computes the set of tuples satisfying a view when a query is performed on the view, but this is transparent to the application

[2]A property in RDF is called information in $C\pi$

(subject) it describes. For example, the location information is only meaningful if attached to an entity such as a person or a car.

Definition 4.2.1. *A Context C is a partial function $\mathcal{I} \times \mathcal{S} \rightharpoonup \mathcal{V}$, where:*

- *$\mathcal{I}$ is a set of context information names ranged over by $i, i', i'', i_1, i_2, \ldots$ (eg. temperature)*
- *$\mathcal{S}$ is a set of entities or subjects of information names ranged over by $s, s', s'', s_1, s_2, \ldots$ (eg. patient$_1$)*
- *$\mathcal{V}$ is a set of values of context information ranged over by v, v', v'', $v_1, v_2, \ldots$ (eg. 37)*

with the property that $\mathcal{I} \cup \mathcal{S} \subseteq \mathcal{V}$ and $\mathcal{I} \subseteq \mathcal{S}$.

Assuming $\mathcal{I} \cup \mathcal{S} \subseteq \mathcal{V}$ and $\mathcal{I} \subseteq \mathcal{S}$ gives the ability to describe multi-level properties (properties of properties) and to discover and refer to entities. For example:

- `(nearestPrinter,` $agent_1$`,` $printer_1$`)` and `(status,` $printer_1$`, ready)`
- `(problem,` car_1`, temperature)`, `(temperature,` car_1`, 100)`, and `(unit,` *temperature*`, celsius)`.

We associate with each element i of $\mathcal{I}$ a type t and write elements of C as triples $(i : t, s, v)$ optionally omitting the type t when it can be inferred from the value v. The value $\perp \in \mathcal{V}$ is reserved to indicate an unknown value for information i of entity s.

A context (as defined above) represents a repository of the information currently known to an agent. The idea is that an agent is interested in observing the values of some properties. These values are provided by other agents in the environment who *broadcast* changes to interested agents. The following are examples of elements of a context.

1. `(location:`Int`,` a_1`, 5)`
2. `(relativeloc:`$Agent \to Int$`,` a_1`,` $\{a_2 \mapsto 10, a_3 \mapsto 4\}$`)`
3. `(nearbyagents:`$\mathbb{P}Agent$`,` a_1`,` $\{a_2, a_3\}$`)`
4. `(nearestPrinter:`$Printer$`,` a_1`,` p_1`)`
5. `(status:`$Status$`,` p_1`,` $ready$`)`
6. `(colour:`Int`,` car_1`,` $\perp$`)`

Triple 1 above defines the location of entity a_1 as being 5. Whereas, triple 2 defines the location of this entity relative to the locations of other entities as a function *Agent* $\to$ *Int*. Triple 6 says that the colour of car_1 is currently unknown by giving it the value $\perp$.

4.3 A calculus of context-awareness

In this section, the Calculus of context-awareness $C\pi$ is discussed. We give the syntax and semantics of the calculus as well as a number of examples. The syntax is given in table 4.1. The calculus represents a context-aware application as a *network* of *agents* executing in parallel. Agents can contain *processes*, which are capable of performing two types of actions. The first type includes the usual π-calculus (polyadic) input and output actions: $a(\tilde{y})$ and $a\langle\tilde{x}\rangle$ and the silent action τ.

The second type includes actions for manipulating contextual information. A process performing action $\langle i, s, v\rangle$ announces to the environment that the context information i of entity s has value v. The contexts of all agents in the environment will be updated to reflect this new value. Action (i, s, x) is performed by a process to query the value of (i, s) from its context. $\hat{a}\langle x\rangle$ is an action that is interpreted according to a condition on the context. Whenever $\hat{a}\langle x\rangle$ is to be performed, in presence of a process $(\phi, \hat{a}\langle y\rangle, E)$ executing in parallel, it is substituted with the action sequence E provided the context satisfies ϕ. The two actions $reg(i, s, v)$ and $dereg(i, s)$ manipulate the definition of the context by adding respectively removing interest in receiving updates to the value of context information (i, s).

We assume the existence of the following sets in addition to the sets $\mathcal{I}, \mathcal{S}, \mathcal{V}$ discussed above.

- $a, b, c, \cdots \in \mathcal{A}$: a countably infinite set of names, which includes the sets $\mathcal{I}$ and $\mathcal{S}$ i.e. $\mathcal{I} \cup \mathcal{S} \subseteq \mathcal{A}$ and is included in the set of values $\mathcal{V}$ ($\mathcal{A} \subset \mathcal{V}$). So, elements of the two sets $\mathcal{I}$ and $\mathcal{S}$ are also considered as *names*.
- $x, y, z, \cdots \in \mathcal{X}$: a countably infinite set of variables. Variables can be substituted by any element of the set $\mathcal{V}$.
- $P, Q, R, \cdots \in \mathcal{P}$: the set of all processes generated by the syntax.
- $m, n, o, \cdots \in \mathcal{G}$: the set of all agent terms generated by the syntax.
- $M, N, O, \cdots \in \mathcal{N}$: the set of all networks generated by the syntax.
- $C, C', C_1, \cdots \in \mathcal{C}$: the set of all contexts.

The following definitions are also needed.

- We write $\tilde{v}$ and $\tilde{x}$ to stand for tuples of values $v_1, \ldots, v_n$ and variables $x_1, \ldots, x_n$ respectively.
- The set of *free* names for process and network terms are given by the function *fn* defined in table 4.2. The function n gives the set of all names occurring in a term.
- If K and L are two process (or network) terms differing only by a substitution of bound names, we say K is the *α-conversion* of L and vice versa.

Table 4.1: Syntax of $C\pi$

Contexts:		
$C ::=$	$[S]$	(Context)
$S ::=$	$(i, s, v), S$	(Element sequence)
	$\mid \varnothing$	(Empty sequence)
Actions:		
$\alpha ::=$	$\langle i, s, v \rangle$	(Publish context information)
	$\mid \hat{a}\langle \tilde{v} \rangle$	(Interpreted action)
	$\mid reg(i, s, v)$	(Register interest)
	$\mid dereg(i, s)$	(De-register interest)
	$\mid \pi$	
$\pi ::=$	$a\langle \tilde{v} \rangle \mid a(\tilde{x}) \mid \tau$	(π-calculus actions)
$E ::=$	$\alpha.E \mid \alpha$	(Action sequence)
Processes:		
$P ::=$	$E.P$	(Prefix)
	$\mid P \vert P$	(Parallel composition)
	$\mid P + P$	(Choice)
	$\mid \{\tilde{x}\}.\phi \hookrightarrow P$	(Conditional process)
	$\mid (\phi, \hat{a}\langle \tilde{x} \rangle, E)$	(Interpretation rule)
	$\mid\ !P$	(Replication)
	$\mid (\nu a)P$	(Restriction)
	$\mid P(\tilde{x})$	(Process instantiation)
	$\mid \mathbf{0}$	(nil Process)
Networks:		
$m ::=$	$C \triangleright P$	(Agent)
$M ::=$	m	(Agent)
	$\mid M \parallel N$	(Network parallel)
	$\mid (\nu i; s)M$	(Context information restriction)
	$\mid (\nu a)M$	(Network restriction)
	$\mid M(\tilde{x})$	(Network instantiation)
	$\mid \mathbf{0}$	(Empty network)
Definitions:		
$P(\tilde{x}) =$	Q	(Process Definition)
$M(\tilde{x}) =$	N	(Network Definition)

$\beta ::= \langle i, s, v \rangle \mid \pi$

$\gamma ::= reg(i, s, v) \mid dereg(i, s)$

Table 4.2: Definition of free names in $C\pi$

Processes:	
$fn(P \mid Q) = fn(P) \cup fn(Q)$	$fn(\langle i, s, v\rangle.P) = \{i, s\} \cup n(v) \cup fn(P)$
$fn(P + Q) = fn(P) \cup fn(Q)$	$fn(\hat{a}\langle\tilde{v}\rangle.P) = \{a\} \cup n(\tilde{v}) \cup fn(P)$
$fn(\{\tilde{x}\}.\phi \hookrightarrow P) = fn(P)$	$fn(reg(i, s, v).P) = \{i, s\} \cup n(v) \cup fn(P)$
$fn((\phi, \hat{a}\langle\tilde{x}\rangle, E)) = fn(E) - \{\tilde{x}\}$	$fn(dereg(i, s).P) = fn(P)$
$fn(!P) = fn(P)$	$fn(a\langle\tilde{v}\rangle.P) = \{a\} \cup n(\tilde{v}) \cup fn(P)$
$fn((\nu a)P) = fn(P) - \{a\}$	$fn(a(\tilde{x}).P) = \{a\} \cup (fn(P) - \{\tilde{x}\})$
$fn(\mathbf{0}) = \emptyset$	
Networks:	
$fn(C \triangleright P) = fn(P)$	
$fn(M \parallel N) = fn(M) \cup fn(N)$	
$fn((\nu a)M) = fn(M) - \{a\}$	
$fn((\nu i; s)M) = fn(M)$	
$fn(\mathbf{0}) = \emptyset$	

In this case, K and L are considered to be equivalent and we write $K \equiv L$ (the relation $\equiv$ is called *structural congruence* and is discussed in section 4.3.1).

- We usually omit any trailing $\mathbf{0}$ (null) processes or networks. Therefore, $a(x).\mathbf{0}$ is written as $a(x)$ and $C \triangleright P \parallel \mathbf{0}$ is written as $C \triangleright P$.
- A simple type system is assumed on variables and channels to ensure values sent over channels have the correct number and type. We assume that substitutions conform to this typing.

In $C\pi$, a system is composed of a number of agents running in parallel (a network). An agent has the form $C \triangleright P$ where C is the context and P is the process term modelling the behaviour of the agent. In addition to being able to perform the above actions, a process term can be:

- The parallel composition of two processes $P \mid Q$. P and Q execute in parallel and may communicate using input and output actions: $a(x)$ and $a\langle v\rangle$.
- A non-deterministic choice between two processes $P + Q$. This represents a process that can act as either P or Q.
- A conditional process $\{\tilde{x}\}.\phi \hookrightarrow P$, which acts as P when inside an agent whose context C satisfies property ϕ for some substitution of variables

$\tilde{x}$ (the syntax of condition ϕ is discussed in section 4.4). Conditional processes are used to query the context and perform context-dependent behaviour. It should be noted here that variables $\{\tilde{x}\}$ bind both ϕ and P. Also, the first action of P is only enabled when ϕ is true in the context. For example, the process:

$$\{x\}.(carsnear, agent_1, has(x)) \hookrightarrow x\langle m\rangle.P$$

models a system that sends message m to *a car* near $agent_1$ only when such car becomes available.

- An interpretation $(\phi, \hat{a}\langle x\rangle, E)$, which allows a process $\hat{a}\langle v\rangle.P$ (running in parallel) to execute context-dependent behaviour by acting as $E\{v/x\}.P$ in case it is in a context satisfying ϕ. These interpretation rules allow the separation of the context-dependent behaviour of the agent from the context-independent behaviour. Using interpretation rules, we can model the second approach to the design of context-awareness middleware discussed in section 3.2. In other words, an agent can be split into a part that is ignorant of the changes to context and another part that models the context dependency using interpretation rules.

- A replicated process $!P$: used to model infinite executions. The process $!P$ can be thought of as generating an infinite number of copies of P running in parallel. This allows, for instance, specifying services that (continuously) handle requests on port a as processes of the form $!a(x).Q$.

Three types of restriction are used to give scopes to names:

1. Name restriction on processes as in $(\nu a)P$ meaning that the name a is local to the process P.

2. Name restriction on networks as in $(\nu a)M$ indicating that a is local to the network of agents M.

3. Context information restriction on networks as in $(\nu i; s)M$ meaning that information i about entity s is only exchanged between agents of M. This allows specifying who gets access to context information in context-aware systems.

It should be noted here that the restriction $(\nu i; s)M$ is different from the restriction of the two names i and s using $(\nu i)(\nu s)M$. The former is meant to disallow the provision of context information from agents outside of the restriction even though these agents possibly know about both names i and s. In other words, names i and s are not private to the network M in $(\nu i; s)M$ and neither actions $\langle i, s, v\rangle$ from outside M reach agents inside M, nor such actions from inside M reach agents outside it.

Context information restriction allows the specification of trust between agents. Two agents trust each other on some context information (i, s) if they can exchange values for this information. In context-aware applications, context information is usually obtained from other systems that are trusted to supply correct and accurate information. For example, we might want to specify that a mobile device obtains location data from a specific location service.

4.3.1 Operational semantics

The semantics of $C\pi$ is given in terms of the structural congruence relation $\equiv$ defined in table 4.3 and the structural operational semantic rules in table 4.4. The structural congruence relation is the smallest congruence satisfying the rules in table 4.3. It is used to identify expressions having the same *structure*. This allows the syntactic manipulation of terms when applying derivations and therefore simplifies the presentation of the operational semantics.

Operational semantics is a method by which a meaning can be given to the syntax of a formal language. A set of inference rules are defined over the possible structures in the language to represent valid transformations (transitions) a term can have in different execution contexts. The rules in table 4.4 define a *Labelled Transition System* (LTS) on process and network terms. An LTS is a tuple $(\mathcal{K}, \mathcal{L}, \rightarrow)$, where $\mathcal{K}$ is the set of processes and networks, $\mathcal{L}$ is a set of labels, and $\rightarrow$ is a relation $\mathcal{K} \times \mathcal{L} \times \mathcal{K}$. An element (K_1, L, K_2) of $\rightarrow$ is usually written as $K_1 \xrightarrow{L} K_2$ to mean that the term K_1 makes a transition L to term K_2. This can be inferred using the rules defining the LTS.

The purpose of function $\mathbf{U} : \mathcal{C} \times Act \rightarrow \mathcal{C}$ ($\mathcal{C}$ is the set of all contexts) used in the rules Ag Act1 and Ag Act2 is to update a context $C \in \mathcal{C}$ with the effects of some action α. To define this function, we first give definitions to the operators $\oplus$, $\ominus$, and the context update $C[(i,s) \mapsto v]$. The context defined by $C \oplus (i,s,v)$ is the same context as C with the element (i,s,v) added in case $(i,s) \notin dom(C)$. In case $(i,s) \in dom(C)$, the value of element (i,s) is updated to v. The context $C \ominus (i,s)$ is the context C with element (i,s) subtracted from its domain. In case $(i,s) \notin dom(C)$, $C = C \ominus (i,s)$. Finally, the context $C[(i,s) \mapsto v]$ is C with the value of (i,s) updated to v. If $(i,s) \notin dom(C)$, then $C[(i,s) \mapsto v] = C$. Formally, the definitions are as follows:

$$C \oplus (i,s,v) = \begin{cases} C \ominus (i,s) \cup \{(i,s,v)\} & \text{if } (i,s) \in dom(C) \\ C \cup \{(i,s,v)\} & \text{if } (i,s) \notin dom(C) \end{cases} \tag{4.1}$$

$$C \ominus (i,s) = \{(x,y,z) \in C \mid x \neq i \vee y \neq s\} \tag{4.2}$$

$$C[(i,s) \mapsto v] = \begin{cases} C & \text{if } (i,s) \notin dom(C) \\ C \ominus (i,s) \cup \{(i,s,v)\} & \text{if } (i,s) \in dom(C) \end{cases} \tag{4.3}$$

The function $\mathbf{U}$ is then defined as follows:

$$\mathbf{U}(C,\alpha) = \begin{cases} C \oplus (i,s,v) & \text{if } \alpha = reg(i,s,v) \\ C \ominus (i,s) & \text{if } \alpha = dereg(i,s) \\ C[(i,s) \mapsto v] & \text{if } \alpha = \langle i,s,v \rangle \\ C & \text{otherwise} \end{cases} \tag{4.4}$$

Table 4.3: Structural congruence for $C\pi$

Processes:

If P is α-convertible to Q or vice versa, then $P \equiv Q$

$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$

$P \mid Q \equiv Q \mid P$

$P \mid \mathbf{0} \equiv P$

$P + (Q + R) \equiv (P + Q) + R$

$P + Q \equiv Q + P$

$P + \mathbf{0} \equiv P$

$(\nu a)\mathbf{0} \equiv \mathbf{0}$

$(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$

$P \mid (\nu a)Q \equiv (\nu a)(P \mid Q) \quad \text{if } a \notin fn(P)$

$P \equiv Q \Rightarrow (\nu a)P \equiv (\nu a)Q$

$P \equiv P' \Rightarrow P \mid Q \equiv P' \mid Q$

$P(\tilde{x}) = Q \Rightarrow P(\tilde{y}) \equiv Q(\tilde{y}/\tilde{x})$

Networks:

If M is α-convertible to N or vice versa, then $M \equiv N$

$M \parallel (N \parallel O) \equiv (M \parallel N) \parallel O$

$M \parallel N \equiv N \parallel M$

$M \parallel \mathbf{0} \equiv M$

$(\nu a)\mathbf{0} \equiv \mathbf{0}$

$(\nu a)(\nu b)M \equiv (\nu b)(\nu a)M$

$M \parallel (\nu a)N \equiv (\nu a)(M \parallel N) \quad \text{if } a \notin fn(M)$

$M \equiv N \Rightarrow (\nu a)M \equiv (\nu a)N$

$M \equiv M' \Rightarrow M \parallel N \equiv M' \parallel N$

$M(\tilde{x}) = N \Rightarrow M(\tilde{y}) \equiv N(\tilde{y}/\tilde{x})$

The set of bound names in label α is given by $bn(\alpha)$. The only label with bound names is $a(\tilde{x})$ in which $\tilde{x}$ are bound.

The rule Cond requires a process $\{\tilde{x}\}.\phi \hookrightarrow P$ to be in a context C satisfying condition ϕ for some substitution of $\tilde{x}$ before progressing. This is a method for the agent to query its context and can therefore be used to adapt the agent's behaviour to its context. If the sequence $\tilde{x}$ is empty, the condition ϕ is assumed to be closed (no free variables) and the process is written as $\phi \hookrightarrow P$ for short. The transition rule Cond in this special case becomes as follows:

$$\frac{C \triangleright P \xrightarrow{\alpha} C' \triangleright P' \qquad C \models \phi}{C \triangleright \phi \hookrightarrow P \xrightarrow{\alpha} C' \triangleright P'} \quad (\textsc{Cond}^-)$$

For instance, a device "*dev*" that reveals patient information only in the surgery can be written as:

$$C \triangleright (loc, dev, = surgery) \hookrightarrow a\langle info\rangle.P$$

The calculus is parameterised on the syntax of ϕ and a satisfaction relation $\models$ is used to determine when a context C satisfies a condition ϕ, writing this as $C \models \phi$. An example of a notation and a satisfaction relation for conditions is given in section 4.4.

Table 4.4: Operational semantics of $C\pi$

$$\frac{}{\alpha.P \xrightarrow{\alpha} P} \quad (\textsc{Pref}) \qquad\qquad \frac{C \triangleright P \xrightarrow{\alpha} C' \triangleright P' \qquad \exists\tilde{v}.C \models \phi\{\tilde{v}/\tilde{x}\}}{C \triangleright \{\tilde{x}\}.\phi \hookrightarrow P \xrightarrow{\alpha\{\tilde{v}/\tilde{x}\}} C' \triangleright P'\{\tilde{v}/\tilde{x}\}} \quad (\textsc{Cond})$$

$$\frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P \mid !P} \quad (\textsc{Rep})$$

$$\frac{P \xrightarrow{\beta} P' \qquad C' = \mathbf{U}(C,\beta)}{C \triangleright P \xrightarrow{\beta} C' \triangleright P'} \quad (\textsc{Ag Act1}) \qquad\qquad \frac{P \xrightarrow{\gamma} P' \qquad C' = \mathbf{U}(C,\gamma)}{C \triangleright P \xrightarrow{\tau} C' \triangleright P'} \quad (\textsc{Ag Act2})$$

$$\frac{M \xrightarrow{\langle i,s,v\rangle} M' \qquad N \xrightarrow{?(i,s,v)} N'}{M \parallel N \xrightarrow{\langle i,s,v\rangle} M' \parallel N'} \quad (\textsc{Upd})$$

$$\frac{M \xrightarrow{\langle i,s,v\rangle} M' \qquad i' \neq i \vee s' \neq s}{(\nu i'; s')M \xrightarrow{\langle i,s,v\rangle} (\nu i'; s')M'} \quad (\textsc{Res Pub})$$

$$\frac{M \xrightarrow{?(i,s,v)} M' \qquad N \xrightarrow{?(i,s,v)} N'}{M \parallel N \xrightarrow{?(i,s,v)} M' \parallel N'} \quad (\textsc{Par Upd})$$

continued on the next page

Table 4.4: Operational semantics of $C\pi$

$$\frac{M \xrightarrow{?(i,s,v)} M' \qquad i' \neq i \vee s' \neq s}{(\nu i'; s')M \xrightarrow{?(i,s,v)} (\nu i'; s')M'} \ (\textsc{Res Upd}) \qquad \frac{}{(\nu i; s)M \xrightarrow{?(i,s,v)} (\nu i; s)M} \ (\textsc{No Upd})$$

$$\frac{C' = C[(i,s) \mapsto v]}{C \triangleright P \xrightarrow{?(i,s,v)} C' \triangleright P} \ (\textsc{Ctxt Upd}) \qquad \frac{K \xrightarrow{\alpha} K' \qquad x \notin n(\alpha)}{(\nu x)K \xrightarrow{\alpha} (\nu x)K'} \ (\textsc{Res})$$

$$\frac{P \xrightarrow{a\langle\tilde{x}\rangle} P' \qquad Q \xrightarrow{a(\tilde{y})} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\{\tilde{x}/\tilde{y}\}} \ (\textsc{Proc Com}) \qquad \frac{M \xrightarrow{a\langle\tilde{v}\rangle} M' \qquad N \xrightarrow{a(\tilde{y})} N'}{M \parallel N \xrightarrow{\tau} M' \parallel N'\{\tilde{v}/\tilde{y}\}} \ (\textsc{Net Com})$$

$$\frac{C \models \phi}{C \triangleright \hat{a}\langle\tilde{v}\rangle.P \mid (\phi, \hat{a}\langle\tilde{x}\rangle, E) \xrightarrow{\tau} C \triangleright E\{\tilde{v}/\tilde{x}\}.P' \mid (\phi, \hat{a}\langle\tilde{x}\rangle, E)} \ (\textsc{Interp})$$

$$\frac{P \xrightarrow{\alpha} P' \qquad dis(\alpha, Q)}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \ (\textsc{Proc Par}) \qquad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \ (\textsc{Sum})$$

$$\frac{M \xrightarrow{\pi} M' \qquad dis(\pi, N)}{M \parallel N \xrightarrow{\pi} M' \parallel N} \ (\textsc{Net Par}) \qquad \frac{C \triangleright P \xrightarrow{\alpha} C' \triangleright P' \qquad dis(\alpha, Q)}{C \triangleright P \mid Q \xrightarrow{\alpha} C' \triangleright P' \mid Q} \ (\textsc{Ag Par})$$

$$\frac{C \triangleright P \xrightarrow{\alpha} C' \triangleright P'}{C \triangleright P + Q \xrightarrow{\alpha} C' \triangleright P'} \ (\textsc{Ag Sum}) \qquad \frac{K \equiv L \xrightarrow{\alpha} L' \equiv K'}{K \xrightarrow{\alpha} K'} \ (\textsc{Struct})$$

Note: K and L range over either $\mathcal{P}$ or $\mathcal{N}$. $dis(\alpha, K) \overset{def}{=} bn(\alpha) \cap fn(K) = \emptyset$

The rule Ag Act1 allows a process to perform an action β that is either a π-calculus action or the information broadcast action $\langle i, s, v\rangle$. These are the only actions visible to other agents. All other actions performed by processes inside an agent are observed by other agents in the network as τ actions. The action $\langle i, s, v\rangle$ changes the context of the agent by updating the value of information (i, s) to value v. π actions do not affect the context of the agent (see definition 4.4).

The rule Ag Act2 covers the actions *reg* and *dereg*, which are not visible outside the agent. These actions update the context by adding or removing triples according to definition 4.4. The action $reg(i, s, v)$ registers interest in receiving information (i, s), whereas $dereg(i, s)$ removes interest in information (i, s). The *dereg* action has no effect if (i, s) is not in the context. Action *reg* requires the specification of an initial value for the information.

Actions *reg* and *dereg* allow an agent to control the definition of its context, which makes this definition dynamic. This is useful when the agent does not know a priori which information it requires. The agent generally registers interest in new information after performing some initial communication steps. For example, the following agent receives the id of a user, then registers interest in knowing the activity of this user.

$$C \triangleright a(userid).reg(activity, userid, \bot).P$$

The rules Upd, Par Upd, and Ctxt Upd specify how context information is published. Publishing context information is done through broadcast communication resulting in the instantaneous update of the contexts of all agents running in parallel. This approach is inspired by the *Calculus of Broadcasting Systems (CBS)* of Prasad [54]. By publishing information, an agent also updates its own context. This is covered by the rule Ag Act1. Rules Res Pub, Res Upd, and No Upd disallow the publishing of context information (i, s) outside a network M where this information is made private using $(\nu i; s)M$ and also disallow any updates to this information from agents outside the network M. As an example of information publishing and restriction, let's consider the transition[3]:

$$\begin{array}{l}(\nu i; s)((\nu i'; s')C_1 \triangleright P_1 \parallel C_2 \triangleright P_2) \parallel C_3 \triangleright \langle i', s', v\rangle.P_3 \\ \xrightarrow{\langle i', s', v\rangle} \\ (\nu i; s)((\nu i'; s')C_1 \triangleright P_1 \parallel C_2' \triangleright P_2) \parallel C_3' \triangleright P_3\end{array}$$

with the assumptions that $i \neq i' \vee s \neq s'$, $C_2' = C_2[(i', s') \mapsto v]$, and $C_3' = C_3[(i', s') \mapsto v]$. An inference tree can be obtained by first applying the rule Upd as follows:

$$\frac{C_3 \triangleright \langle i', s', v\rangle.P_3 \xrightarrow{\langle i', s', v\rangle} C_3' \triangleright P_3 \qquad \begin{array}{c}(\nu i; s)((\nu i'; s')C_1 \triangleright P_1 \parallel C_2 \triangleright P_2) \\ \xrightarrow{?(i', s', v)} \\ (\nu i, s)((\nu i'; s')C_1 \triangleright P_1 \parallel C_2 \triangleright P_2)\end{array}}{\begin{array}{l}(\nu i; s)((\nu i'; s')C_1 \triangleright P_1 \parallel C_2 \triangleright P_2) \parallel C_3 \triangleright \langle i', s', v\rangle.P_3 \xrightarrow{\langle i', s', v\rangle} \\ \qquad\qquad (\nu i; s)((\nu i'; s')C_1 \triangleright P_1 \parallel C_2' \triangleright P_2) \parallel C_3' \triangleright P_3\end{array}}$$

The transition on the left is obtained in a straightforward way from rules Pref and Ag Act1. The transition on the right is inferred using the following instance of rule Res Upd:

$$\frac{(\nu i'; s')C_1 \triangleright P_1 \parallel C_2 \triangleright P_2 \xrightarrow{?(i', s', v)} (\nu i'; s')C_1 \triangleright P_1 \parallel C_2' \triangleright P_2}{\begin{array}{l}(\nu i; s)((\nu i'; s')C_1 \triangleright P_1 \parallel C_2 \triangleright P_2) \xrightarrow{?(i', s', v)} \\ \qquad\qquad (\nu i, s)((\nu i'; s')C_1 \triangleright P_1 \parallel C_2 \triangleright P_2)\end{array}}$$

[3]We allow restriction to bind more tightly than $\parallel$ and $|$

Making use of the rule Par Upd, the transition in the premise of the above rule is obtained as follows:

$$\frac{(\nu i'; s')C_1 \rhd P_1 \xrightarrow{?(i',s',v)} (\nu i'; s')C_1 \rhd P_1 \quad C_2 \rhd P_2 \xrightarrow{?(i',s',v)} C_2' \rhd P_2}{(\nu i'; s')C_1 \rhd P_1 \parallel C_2 \rhd P_2 \xrightarrow{?(i',s',v)} (\nu i'; s')C_1 \rhd P_1 \parallel C_2' \rhd P_2}$$

The transition on the left is obtained through the base case No Upd, while the one on the right is a straightforward application of the rule Ctxt Upd.

We can see in the above example that information (i', s') was delivered to agents $C_2 \rhd P_2$ and $C_3 \rhd P_3$ without affecting the agent $C_1 \rhd P_1$ in which this information was made private using the restriction $(\nu i'; s')$.

The rule Net Com allows point to point channel communication between agents. The rule Interp substitutes any interpreted action $\hat{a}\langle\tilde{v}\rangle$ by a sequence of actions E in case an interpretation rule $(\phi, \hat{a}\langle\tilde{x}\rangle, E)$ exists in parallel. The condition ϕ of the interpretation rule must also be satisfied in the current context of the enclosing agent for the substitution to occur. Interpretation rules allow an agent to adapt to its context, but unlike the conditional expressions $\phi \hookrightarrow P$, they represent a way of separating context adaptation from the actual behaviour of the context-aware application. For example, the following agent prints to a postscript printer if one is available, otherwise it prints to a raw one.

$$C \rhd \widehat{print}\langle file\rangle.P \mid (\exists x.(type, x, = ps), \widehat{print}\langle x\rangle, printps\langle x\rangle) \mid$$
$$(\forall x.\neg(type, x, = ps) \wedge \exists x.(type, x, = raw), \widehat{print}\langle x\rangle, printraw\langle x\rangle)$$

Where $\exists x.(type, x, = ps)$ is true only if the context C contains a triple whose first element is "*type*" and whose value is equal to "*ps*". $\forall x.\neg(type, x, = ps)$ is true only if C cotains *no* triple whose first element is "*type*" and whose value is equal to "*ps*". Assuming the context C contains a triple $(type, p_1, ps)$, the above agent could perform a τ action to become:

$$C \rhd printps\langle file\rangle.P \mid (\exists x.(type, x, = ps), \widehat{print}\langle x\rangle, printps\langle x\rangle) \mid$$
$$(\forall x.\neg(type, x, = ps) \wedge \exists x.(type, x, = raw), \widehat{print}\langle x\rangle, printraw\langle x\rangle)$$

4.4 A logic for context

In this section, an example syntax and satisfaction relation are given for the conditions ϕ used in the process calculus.

Syntax

A syntax for the formulae ϕ used in the process calculus can be defined as follows:

$$\phi ::= (i : t, s, \psi_t) \mid \exists\tilde{x}.\phi \mid \forall\tilde{x}.\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \Rightarrow \phi \mid T \mid F \mid \neg\phi$$

We associate with each type t of context information a formula syntax ψ_t. For example, with the integer type Int, we associate the formula syntax:

$$\psi_{Int} ::= \; > n \;\mid\; < n \;\mid\; = n \mid \neg\psi_{Int}$$

In particular, for each type t we have *at least* formulae "$= v$" and "$\neg\psi_t$" to denote equivalence of values and negation of formulae.

We also associate a satisfaction relation $\models_t$ with each type t and formula syntax ψ_t. For the example of integers, we have:

$$\begin{array}{lll} v \models_{Int} \; > n & \text{iff} & v > n \\ v \models_{Int} \; < n & \text{iff} & v < n \\ v \models_{Int} \; = n & \text{iff} & v = n \\ v \models_{Int} \neg\psi_{Int} & \text{iff} & v \not\models_{Int} \psi_{Int} \end{array}$$

where the relation symbols $>, <, =$ are the usual relations on integers.

We can define a formula syntax ψ_L for lists of elements of type X as follows:

$$\psi_{L(X)} ::= \mathit{head}\ \psi_X \mid \mathit{tail}\ \psi_{L(X)} \mid \; = l \; \mid \neg\psi_{L(X)}$$

A syntax for lists of lists of elements of type X can defined by:

$$\psi_{LL(X)} ::= \mathit{head}\ \psi_{L(X)} \mid \mathit{tail}\ \psi_{LL(X)} \mid \; = ll \; \mid \neg\psi_{LL(X)}$$

A syntax for sets can be as follows:

$$\psi_{\mathbb{P}(X)} ::= \mathit{has}\ \psi_X \mid \; = v \; \mid \neg\psi_{\mathbb{P}(X)}$$

where "*has* ψ_X" means the set has an element (with type X) that satisfies ψ_X.

Satisfaction relation $\models$

The satisfaction relation used in the semantics of the calculus is defined as follows:

$$\begin{array}{lll} C \models (i : t, s, \psi_t) & \text{iff} & \exists v \in \mathcal{V}.(i : t, s, v) \in C \wedge v \models_t \psi_t \\ C \models \forall\tilde{x}.\phi & \text{iff} & \forall\tilde{v} \in \mathcal{V}^n.C \models \phi\{\tilde{v}/\tilde{x}\} \\ C \models \exists\tilde{x}.\phi & \text{iff} & \exists\tilde{v} \in \mathcal{V}^n.C \models \phi\{\tilde{v}/\tilde{x}\} \end{array}$$

The definition of $\models$ for the other types of formulae is straightforward.

4.5 Abbreviations

In this section, a number of abbreviations are defined to make writing examples simpler.

- **Read:**
$$(i, s, x).P ::= \{x\}.(i, s, = x) \hookrightarrow P$$
This is a shorthand for reading the current value of context information (i, s).

- **Conditional read:**
$$(i : t, s, \psi_t \curvearrowright x).P ::= \{x\}.(i, s, \psi_t) \wedge (i, s, = x) \hookrightarrow P$$
This allows reading the value of information i of entity s into variable x only in case condition ψ_t is satisfied on the value of the triple. The process blocks until condition ψ_t is satisfied.

- **if-then-else:**
$$\{\tilde{x}\}.\phi \hookrightarrow P \; else \; Q ::= \{\tilde{x}\}.\phi \hookrightarrow P + \forall \tilde{x}.\neg\phi \hookrightarrow Q$$
This means: behave as P in case ϕ is true for some substitution of x, else behave as Q.

- **Wait for a change (z is fresh in P):**
$$(i, s, \sim x).P ::= (i, s, z).\tau.(i, s, \neg = z \curvearrowright x).P$$
This means: wait for the value of (i, s) to change and set x to the new value. The process blocks until there is at least one change in the value of (i, s). After the statement $(i, s, \sim x)$ completes, the agent knows there has been *at least* one change in the value of (i, s).

4.6 Examples

Following is a list of examples of context-aware systems and how they are encoded in $C\pi$.

A driver is warned if an emergency vehicle is nearby

$$\begin{aligned}
P &\overset{def}{=} (emer, car, \; = on) \hookrightarrow \widehat{warning}\langle on\rangle.(emer, car, \; = off) \hookrightarrow \\
&\quad \widehat{warning}\langle off\rangle.P \\
car &\overset{def}{=} [(emer, car, off), (road, car, \perp)] \triangleright \\
&\quad P \mid ((road, car, \; = B), \widehat{warning}\langle x\rangle, \langle flashlight, car, x\rangle) \mid \\
&\qquad ((road, car, \; = A), \widehat{warning}\langle x\rangle, \tau)
\end{aligned}$$

The car agent waits for the emergency to have value "*on*". When this occurs, the driver is warned using the action $\widehat{warning}\langle on\rangle$, which is an interpreted action controlled by two interpretation rules. These rules determine how the warning is issued depending on the type of road the car is currently in. After the warning is enabled, the agent waits for the emergency to finish using a condition $(emer, car, \; = off)$ then turns the warning off.

A device prints to the nearest printer

$$\begin{aligned} device \overset{def}{=}\ & [(nearestp, device, \bot)] \triangleright \\ & !print(x).\hat{p}\langle x\rangle \mid \big((nearestp, device, \neg = \bot), \hat{p}\langle y\rangle, \\ & \qquad (nearestp, device, z).z\langle y\rangle\big) \end{aligned}$$

The device agent specifies interest in context information "*nearestp*", which holds the name of the printer nearest to this agent. This information could be provided by some service (in the environment) that registers interest in knowing the location of the device agent and continuously updates the value of the nearest printer information. Printing is performed by sending the file to be printed (received from a client through channel "*print*") on the name of the printer assuming this name represents a channel on which files can be sent to the printer. This is again done using an interpretation rule, which requires printing to be done only in a context where the nearest printer is known.

A mobile phone vibrates instead of ringing if the user is in a meeting

$$\begin{aligned} phone \overset{def}{=}\ & [(meeting, user, no)] \ \triangleright \\ & phone(x).\widehat{ring}\langle on\rangle.(userpick(y).\widehat{ring}\langle off\rangle + \\ & \qquad usercancel(y).\widehat{ring}\langle off\rangle.x\langle voicemail\rangle) \mid \\ & ((meeting, user, \ = yes), \widehat{ring}\langle x\rangle, \langle vibration, phone, x\rangle) \mid \\ & ((meeting, user, \ = no), \widehat{ring}\langle x\rangle, \langle melody, phone, x\rangle) \end{aligned}$$

The *phone* agent receives a call on the channel *phone* then performs the action $\widehat{ring}$ which is interpreted depending on whether the user is in a meeting. After the phone rings, it communicates with the user who can either pick up the call or cancel it (using channels *userpick* and *usercancel* respectively). In both cases, the phone stops ringing.

A mail client downloads new messages when a wireless connection is detected

$$\begin{aligned} P &\overset{def}{=} (wireless, device, = on) \hookrightarrow s\langle inbox\rangle.wait\langle t\rangle.P \\ Q &\overset{def}{=} (newmsgs, inbox, \sim x).display\langle x\rangle.Q \\ client &\overset{def}{=} (\nu\ inbox)[(wireless, device, off), (newmsgs, inbox, [])] \triangleright\ P \mid Q \\ server &\overset{def}{=} [\,] \triangleright\ !(\nu\ conn)\ s(x).db\langle conn\rangle.conn\langle x\rangle. \\ &\qquad conn(msgs).\langle newmsgs, x, msgs\rangle \end{aligned}$$

The server process receives requests on channel s, retrieves the new messages from its database and then updates the context of the client using the action $\langle newmsgs, inbox, msgs\rangle$. Process Q of the client updates the display when the value of *newmsgs* changes. On the other hand, process P issues a request for new messages to the server then waits for time t before issuing the next request. Notice that the name *inbox* is private to the client process but after communication with the server, it becomes known to it as well.

A web-service performs a search for nearby attractions based on the mobile device's location

$$\begin{aligned} service \overset{def}{=}\ & [\,] \triangleright\ !s(c).reg(loc, c, \bot).reg(browser, c, \bot). \\ & (loc, c, \sim x).(browser, c, \sim y).search\langle x, db\rangle.db(result).style\langle y\rangle. \\ & style(xslfile).c\langle result\rangle.c\langle xslfile\rangle.dereg(loc, c).dereg(browser, c) \end{aligned}$$

Initially, the service's context is empty. A client device connects to the service through channel s sending a (private) channel c. This channel is used by the service to identify the context information specific to this client and is used by the client to send context information to a specific service. Channel c can be thought of as a session identifier. The service registers interest in being notified about the location and web-browser of the device, waits for this information to be updated and services the request. The service performs a search on the database for attractions near the location of the device, retrieves the appropriate style-sheet file for the type of browser and returns this file as well as the search result to the client. Finally, the service removes interest in obtaining the client's browser and location information.

4.7 Behavioural equivalence

In this section, we give a definition of a bisimulation relation on the LTS defined in table 4.4. As a general definition, a bisimulation is a symmetric binary relation $\mathbf{R}$ on terms satisfying:

$$K\mathbf{R}L \text{ and } K \xrightarrow{\alpha} K' \text{ implies } \exists L'.\ L \xrightarrow{\alpha} L' \wedge K'\mathbf{R}L'$$

Informally, this means that if K can do some action α, then L can do the same action and the resulting terms are in the same relation. We say two terms are *bisimilar* if there exists a bisimulation relating them.

We define bisimulation over networks of agents and consider the set of observables to be the labels $\langle i, s, v\rangle$, $a(x)$, $a\langle v\rangle$, τ, and $?(i, s, v)$. The definition is as follows.

Definition 4.7.1 (Network bisimulation)**.** *A (strong)* network *bisimulation is a symmetric binary relation* $\mathbf{R}$ *on agents such that* $M\mathbf{R}N$ *and* $M \xrightarrow{o} M'$ *then:*

1. *if* $o = a(\tilde{x})$ *then* $\exists N'.N \xrightarrow{a(\tilde{y})} N' \wedge \forall \tilde{v}.M'\{\tilde{v}/\tilde{x}\}\mathbf{R}N'\{\tilde{v}/\tilde{y}\}$
2. *otherwise,* $\exists N'.\ N \xrightarrow{o} N' \wedge M'\mathbf{R}N'$

Where $o ::= \langle i, s, v\rangle \mid a(x) \mid a\langle v\rangle \mid ?(i, s, v) \mid \tau$.
We write $M \sim_n N$ *to say that* M *and* N *are bisimilar.*

Informally, two networks are considered to be equivalent if they provide the same context information (using actions $\langle i, s, v\rangle$), are affected in the same way by context changes (actions $?(i, s, v)$), and encode the same communication protocol (π-actions). This definition of bisimulation is strong because we do not ignore τ actions.

To see that the context update actions $?(i, s, v)$ are necessary for equivalence, notice that omitting them from the above definition of bisimulation will cause some problems. Specifically, the agent $C \triangleright \phi \hookrightarrow P$ would be considered equivalent to the $\mathbf{0}$ network if $C \not\models \phi$ as both are incapable of doing any observable action. However, when placed in a network of other agents, the agent $C \triangleright \phi \hookrightarrow P$ can have its context C updated causing the condition ϕ to be satisfied and process P enabled. A similar problem also occurs with interpretation rules.

Equivalence on processes could be defined in terms of network bisimulation. Two processes P and Q can be considered equivalent with respect to some specific context C or with respect to all contexts. In other words, two processes P and Q are equivalent with respect to a context C if:

$$C \triangleright P \sim_n C \triangleright Q$$

4.8 Limitations

Representation of structured values: Context in $C\pi$ is represented as triples of the form (i, s, v). This allows environmental information to be modelled as a directed graph where s and v are nodes and i is an edge. This is similar to how RDF models information. A useful feature of RDF is anonymous or blank nodes, which allow one to represent structured values. For example, the following set of triples represent the address of the staff member with id `exstaff:85740` [42]:

```
exstaff:85740   exterms:address         _:johnaddress .
_:johnaddress   exterms:street          "1501 Grant Avenue" .
_:johnaddress   exterms:city            "Bedford" .
_:johnaddress   exterms:state           "Massachusetts" .
_:johnaddress   exterms:postalCode      "01730" .
```

This effectively says: let's consider the address of the staff member with id `exstaff:85740` to be the *private* name `_:johnaddress`. This name has properties `street`, `city`, and so on. We can use this approach in $C\pi$ to simplify the representation of structured context information. So, instead of announcing the following piece of information:

$$\langle \text{address}, \text{staff85740}, \{\mathit{street} \mapsto s, \mathit{city} \mapsto c, \dots\}\rangle$$

an agent would use something like:

$$\langle (\text{address}, \text{staff85740}, \nu x), (\text{street}, x, s), \dots\rangle$$

Expressiveness of context specifications: Agents specify what information they are interested in by including a triple (i, s, v) in their context. After that, this information is continuously updated. Often in applications, a more powerful specification is required. For instance, an agent might want to know the locations of all cars without specifying what all the cars are. This can be accomplished by allowing simple *pattern matching* on context specifications as in the agent:

$$[(location, * : CAR)] \triangleright P$$

This approach is better then using a complex value to communicate the locations of all cars as in:

$$[(locations, cars, \{car_1 \mapsto l_1, car_2 \mapsto l_2, \dots\})] \triangleright P$$

This is because context information can be updated using actions $\langle location, car_1, l\rangle$ where the agent does not need to know about the existence of car_1. In other words, this new approach would give agents the ability to discover resources in the environment.

Context information restriction: Another downside of *Cπ* is the fact that context information restriction $(\nu i; s)M$ is not *dynamic*. In other words, the scope is specified initially and cannot be changed during run-time. A dynamic scope is required in many applications. An example is when we want to model a system which, based on some initial communication, decides which agents it can accept context information from and which ones it can communicate information to.

Interpretation rules: Interpretation rules can be created dynamically by prefixing them with a number of actions as in:

$$\alpha.\beta.(\phi, \hat{a}\langle x\rangle, E) \mid \hat{a}\langle v\rangle.P$$

However, these rules cannot be discarded after being created. This is inflexible because, in this case, the rules can only be used to modify the behaviour at a *global* level. One way to solve this is to consider interpretation rules as actions and not processes as in:

$$\alpha.(\phi, \hat{a}\langle x\rangle, E).P \mid \hat{a}\langle v\rangle.Q$$

This way, a rule can be discarded as soon as it is used. A rule with a global effect could be written using replication as follows:

$$P \mid !(\phi, \hat{a}\langle x\rangle, E).\mathbf{0}$$

Communication: Communication in *Cπ* is possible between agents as well as between processes using π-calculus input and output primitives. Two agents in a network or two processes inside the same agent can exchange values through channel a if one performs an output action $a\langle\tilde{v}\rangle$ and the other an input action $a(\tilde{x})$. However, a communication action performed by a process P inside agent m can either synchronise with an action from another agent or with one from a process running in parallel with P. This can be seen in the following network:

$$C \triangleright a(x).P \mid a\langle v\rangle.Q \parallel C' \triangleright a(x).R$$

which can perform a τ to become:

$$C \triangleright P\{v/x\} \mid Q \parallel C' \triangleright a(x).R$$

where communication happened between processes or:

$$C \triangleright a(x).P \mid Q \parallel C' \triangleright R\{v/x\}$$

where communication happened between agents.

Part of the problem can be accounted for using restriction as follows:

$$C \triangleright (\nu a)(a(x).P \mid a\langle v\rangle.Q) \parallel C' \triangleright a(x).R$$

which makes channel a private to the process $a(x).P \mid a\langle v\rangle.Q$ and therefore disables the communication between agents. However, a process cannot choose to send a value to (or receive a value from) an external agent without interference from an internal process. A possible solution might be to specify external communication explicitly using actions $\downarrow a(x)$ and $\uparrow a\langle v\rangle$, which synchronise only with external agents.

Summary

In this chapter, a formal model of context-awareness was described. The model is in the form of a process calculus where applications are represented as networks of agents running in parallel. An agent specifies the context information it requires, which is used to adapt to the changing state of the environment. The calculus has a number of limitations which can be addressed as part of future research.

Chapter 5

Context-Awareness Logic

In this chapter, we talk about a logic for expressing the properties of context-aware systems. The logic is based on the μ-calculus [63, ch.5] with modalities for talking about context changes and the behaviour of the system in relation to these changes. We start this chapter by justifying the choice of μ-calculus based logic. Section 5.2 then discusses the syntax of the logic. Section 5.3 details the semantics. Section 5.4 describes some connectives derived from the main ones in the logic. Finally, section (5.5) goes through some examples. The reader is referred to [63, ch.5] for a good introduction to the μ-calculus.

5.1 μ-calculus based logic

The μ-calculus, introduced in section 2.2, is a very powerful modal logic which encompases many other logics such as LTL, CTL, and CTL*. In attempting to find a logic for context-awareness, it was decided to investigate the following three options.

1. Extend HML (Hennessy Milner Logic).
2. Use a temporal logic such as LTL, CTL, or CTL*.
3. Use μ-calculus.

The first attempt was not successful as we found that most of the useful properties of context-aware systems involve describing recurring behaviour such as "whenever an emergency vehicle is detected, the driver is warned". This quickly led to the second choice. Some progress was made in coming up with a LTL-based context-awareness logic. However, expressing the semantics of the formulas based on a LTS was awkward given that labelled transitions do not have a natural correspondence to formulas in LTL (and CTL/CTL*). Finally, the mu-calculus was chosen because it extends HML with the ability to express temporal properties and the fact that its semantics are based on a LTS model.

In defining a logic for context-awareness, we extend the μ-calculus with the modality IN[ϕ]. This modality allows linking states to context properties. This means we can write properties like *"The car is coloured"* and give context-awareness properties to systems such as *"Whenever a non-coloured car enters the paint shop, it will eventually become coloured"*. Also, in many context-aware systems, the fact that an action will possibly happen is not enough. We want to guarantee that some action will be performed by a system given a number of circumstances (context information). This is captured by a μ-calculus macro defined in our logic as $\mathbf{wh}\,\alpha@\psi$ meaning that action α is guaranteed to happen at some point in the future from a state which satisfies ψ.

No formal study was attempted to find the subset of the μ-calculus adequate for expressing context-awareness properties. Nonetheless, based on the examples covered, we think that the LTL fragment of this powerful logic together with the properties $\mathbf{wh}\,\alpha@\psi$ and $\mathbb{X}\psi$ (explained below) would be enough in most cases.

5.2 Syntax

$\psi ::=$	Agent formula
$\quad \mathrm{IN}[\phi]$	In context
$\quad \mid \langle K \rangle \psi$	Action
$\quad \mid Z$	Propositional variable
$\quad \mid \mu Z.\psi$	Least fixed point
$\quad \mid \exists x.\psi$	Existential quantification
$\quad \mid \psi \wedge \psi'$	Conjunction
$\quad \mid \neg\psi$	Negation
$\quad \mid tt$	True
$\phi ::=$	Context formula
$\quad (i, s, \rho)$	Context element
$\quad \mid \phi \wedge \phi'$	Conjunction
$\quad \mid \neg\phi$	Negation
$\quad \mid tt$	True

The syntax of the logical formulas is defined above. There are two types of formulas describing agents and contexts. The idea is to link the behaviour of the agent described in terms of agent formulas to the state of the context described in terms of context formulas. The meaning of the formulas will be given in section 5.3 using a satisfaction relation.

Following is a brief explanation of the non-trivial formulas in the syntax of the logic.

- IN[ϕ] indicates that the agent is currently in a context which satisfies ϕ.

- $\langle K\rangle\psi$ the agent can perform an action in the set K and reduce to an agent that satisfies ψ.
- The formula Z is satisfied by an agent if this agent is an element of $V(Z)$. V is a valuation function which maps each variable Z to the set of agents satisfying this variable. Its purpose is similar to that of the valuation function used in the semantics of first-order logic. To interpret a formula containing free variables $X, Y, Z, \ldots$, we need to know the meaning of these variables. When the formula has no free variables (closed), the valuation function is implicit (i.e does not have to be explicitly provided for the interpretation of the formula).
- In $\mu Z.\psi$, free occurrences of Z in ψ are bound by μZ. The formula $\mu Z.\psi$ is a recursive one which is satisfied by an agent if this latter satisfies $\psi\{\mu Z.\psi/Z\}$. It represents the least fixed point of the function $\psi(Z)$. This function must be monotonic to guarantee the existence of fixed points. To guarantee monotonicity, it is enough to impose the syntactic restriction that any variable in the scope of a fixed point operator must be preceded by an even number of negations [63, ch.5]. Section 2.2 introduces the μ-calculus fixed point operators.
- $\exists x.\psi$ is the usual existential quantification over the set of values $\mathcal{V}$ where a value v that makes $\psi\{v/x\}$ satisfied is required for the formula to be satisfied.
- (i, s, ρ) is satisfied by a context if this latter contains a triple (i, s, v) where v satisfies ρ. The logic is parameterised on the definition of formulas for values ρ. We make use of the definition in section 4.4 where ρ is named ψ_t.

5.3 Semantics

We model context-aware systems using a labelled transition system

$$M = (\mathcal{S}, \rightarrow: \mathcal{S} \times (\mathcal{A} \cup \{u\}) \times \mathcal{S}, \mathit{cont} : \mathcal{S} \rightarrow \mathcal{C})$$

Where $\mathcal{S}$ is the global set of agents and $\mathcal{A}$ represents the set of observable actions ranged over by a while u denotes the occurrence of a context update. We let α range over the set $\mathcal{A} \cup \{u\}$. The relation $\rightarrow$ determines the possible labelled transitions between the states, whereas the function *cont* gives for each state s the context of the system in this state $\mathit{cont}(s)$.

The formulas of the logic are interpreted with respect to a state s of a transition system M. We write $M, s \models \psi$ to mean that a state s of M satisfies formula ψ. We usually omit M and write $s \models \psi$ when M is clear from the context. Table 5.1 defines the satisfaction relation $\models$. This latter is defined (as is the case for the μ-calculus [63]) relative to a valuation V which is used for the interpretation of variables Z and recursive formulas $\mu Z.\psi$.

$M, s \models_V \mathrm{IN}[\phi]$	iff	$cont(s) \models_c \phi$
$M, s \models_V \langle K\rangle\psi$	iff	$\exists s', \alpha.\ s \xrightarrow{\alpha} s'$ and $\alpha \in K$ and $M, s' \models_V \psi$
$M, s \models_V Z$	iff	$s \in V(Z)$
$M, s \models_V \mu Z.\psi$	iff	$s \in \bigcap\{S \subseteq \mathcal{S}.\ \lVert\psi\rVert_{V[S/Z]} \subseteq S\}$
$M, s \models_V \exists x.\psi$	iff	$\exists v : \mathcal{V}.\ M, s \models_V \psi\{v/x\}$
$C \models_c (i, s, \rho)$	iff	$\exists v : \mathcal{V}.\ (i, s, v) \in C$ and $v \in \lVert\rho\rVert$

Table 5.1: Definition of the satisfaction relation relative to valuation V

The property $IN[\phi]$ is satisfied by state s representing an agent if the context of this state (given by $cont(s)$) satisfies ϕ. $\langle K\rangle\psi$ is satisfied by a state s of the transition system if there exists a transition from s to some state s' labelled with a label in K and s' satisfies ψ.

The definition of satisfaction for $\mu Z.\psi$ basically says that s is a member of the least fixed point which is defined as the intersection of prefixed points for the function $f[\psi, Z] : 2^{\mathcal{S}} \to 2^{\mathcal{S}}$ in turn defined as follows:

$$f[\psi, Z](S) \overset{def}{=} \lVert\psi\rVert_{V[S/Z]} = \{s \in \mathcal{S} : s \models_{V[S/Z]} \psi\}$$

Where S is a subset of the set of all states $\mathcal{S}$. $\lVert\psi\rVert_V$ represents the set of all processes satisfying ψ under valuation V.

5.4 Derived Connectives

Following is a list of modalities and connectives derived from the main ones in the previous section. Derived connectives are used later in examples.

$\psi \vee \psi' = \neg(\neg\psi \wedge \neg\psi')$
$\psi \to \psi' = \neg\psi \vee \psi'$
$\phi \vee \phi' = \neg(\neg\phi \wedge \neg\phi')$
$\phi \to \phi' = \neg\phi \vee \phi'$
$\forall x.\psi = \neg\exists x.\neg\psi$
$\nu Z.\psi = \neg\mu Z.\neg\psi\{\neg Z/Z\}$
$\mathbf{ev}\,\psi = \mu Z.(\psi \vee (\langle -\rangle tt \wedge [-]Z))$
$\mathbf{al}\,\psi = \nu Z.(\psi \wedge [-]Z)$
$\mathbf{wh}\,\alpha @\psi = \mu Z.((\langle -\rangle tt \wedge \psi \wedge [-\alpha]Z) \vee (\langle -\rangle tt \wedge \neg\psi \wedge [-]Z)$
$\mathbb{X}\psi = \nu Z.([u]Z \wedge (\langle -u\rangle tt \to \psi))$
$[K]\psi = \neg\langle K\rangle\neg\psi$

The first four connectives are the usual disjunction $\vee$ and implication $\rightarrow$ on agent and context formulas. Universal quantification $\forall x.\psi$ and greatest fixed point $\nu Z.\psi$ formulas are defined as duals of $\exists x.\psi$ and $\mu Z.\psi$ respectively. The ν modality is used to express safety properties whereas μ is used to express liveness. The modalities $\mathbf{ev}, \mathbf{al}, \mathbf{wh}$ are defined in terms of the modalities ν and μ to have the following meanings:

- **ev** is short for "eventually" meaning that any maximal sequence of transitions starting with state s includes a state s' satisfying ψ.
- **al** is short for "always" meaning that ψ is satisfied in every state reachable from state s.
- **wh** is short for "will happen" meaning that every maximal sequence of transitions starting with s includes a transition $s' \xrightarrow{\alpha} s''$ (labelled with action α) s.t. s' satisfies ψ. Here, we take $-\alpha$ as the set $\mathcal{A} - \alpha$ and $-$ to be the set of all actions $\mathcal{A}$. So,
  - $\langle - \rangle tt$ means there is a next action.
  - $[-]Z$ means after any next action, Z holds.
  - $[-\alpha]Z$ means after any next non-α action, Z holds.
  - $\mu Z.(\langle - \rangle tt \wedge \psi \wedge [-\alpha]Z) \vee (\langle - \rangle tt \wedge \neg\psi \wedge [-]Z)$ means if ψ holds, then we continue looking at states after the non-α actions only. If ψ does not hold, then we continue looking at the states following any action. For example, if $a \xrightarrow{\alpha} b$ and $a \xrightarrow{\beta} c$ then if $a \models \psi$, we continue with c only. Otherwise, we continue with both b and c.

  This modality is important when we want to guarantee the execution of action α in the future.

The property $\mathbf{wh}\,\alpha@\psi$ is not equivalent to $\mathbf{ev}(\langle\alpha\rangle tt \wedge \psi)$ as this latter only captures the fact that action α must eventually become enabled and not that it is part of every execution path. For example, if

$$a \xrightarrow{\alpha_1} b \xrightarrow{\alpha_2} c \quad \text{and} \quad b \xrightarrow{\alpha_3} d$$

and if $b \models \psi$, then $\mathbf{ev}(\langle\alpha_3\rangle tt \wedge \psi)$ is true because α_3 is enabled in state b and ψ holds in this state. However, α_3 is not part of the path $a \xrightarrow{\alpha_1} b \xrightarrow{\alpha_2} c$. If the above transitions represented a system, we cannot guarantee that α_3 will be performed in the future. The use of the formula $\mathbf{wh}\,\alpha_3@\psi$ achieves this guarantee. In examples, we take $\mathbf{wh}\,\alpha$ as an abbreviation for $\mathbf{wh}\,\alpha@tt$.

The property $\mathbb{X}\psi$ means that after skipping all context updates u, the property ψ is satisfied. This is useful when we want to talk about the next behaviour of an agent ignoring updates to context. Finally, the property $[K]\psi$ is the dual of $\langle K\rangle\psi$ and means that after all next transitions labelled with a label in K property ψ holds.

5.5 Examples

A driver is warned if an emergency vehicle is nearby

This desirable characteristic might be captured by the property:

$$\mathbf{al}\,(\text{emergency} \rightarrow \mathbf{wh}\, w\langle on\rangle) \tag{5.1}$$

which indicates that throughout the execution of the system in its context, whenever the property *emergency* is satisfied, the system warns the driver at some point in the future. The property *emergency* might be:

$$\exists x.\text{IN}[(type, x, = emerveh) \wedge (emer, x, = on)]$$

Property 5.1 has a problem in that the system can warn the driver at a time when no emergency vehicle is nearby. This is because the state of the context can change at any point. We can try imposing the following property in addition to 5.1:

$$\mathbf{al}\,(\langle w\langle on\rangle\rangle tt \rightarrow \text{emergency})$$

This ensures that the driver can never be warned in the context of no emergency. However, the system is still allowed to respond with *one* warning for several "non-emergency→emergency" changes in the context.

Figure 5.1 shows an example transition system where each state in the system $(s_1, s_2, \dots)$ has a corresponding context represented by a circle. Black circles denote emergency situations while white ones denote non-emergency situations. In the execution paths starting with $(s_1, s_2, s_5, s_6, s_7, \dots)$, a warning 'W' action is not performed until the system had passed through two emergency contexts (in states s_2 and s_6).

Although the driver eventually receives a warning whenever there is an emergency, there is no guarantee that the number of warnings matches the number of emergencies. There is also no guarantee that warnings come within a reasonable amount of time. These are known limitations of the logic that could be addressed in future work.

A device prints to the nearest printer

We can describe two properties of such a system as follows:

1. When the system receives a request to print file x, if the nearest printer is y, then x should eventually be sent to y.

$$\mathbf{al}\,([print(x)](\exists y.\text{IN}[(nearestp, d, = y)] \rightarrow \mathbf{wh}\, y\langle x\rangle))$$

2. Whenever the system sends a print job to printer y, y must be the current nearest printer.

$$[print(x)]\exists y.\mathbf{ev}\,(\langle y\langle x\rangle\rangle tt \wedge \text{IN}[(nearestp, d, = y)])$$

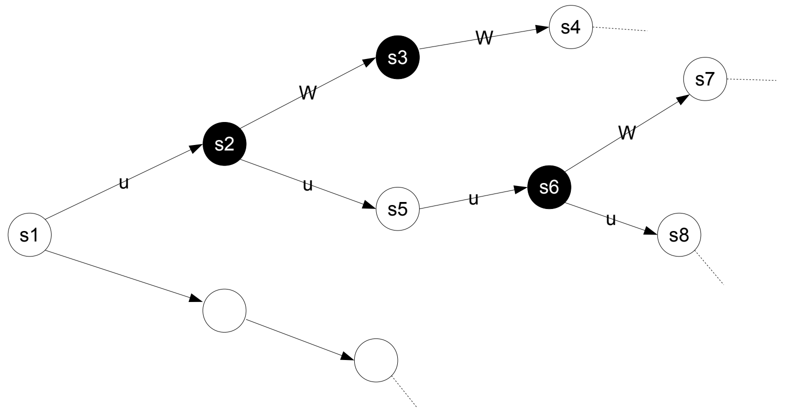


Figure 5.1: Part of a transition system for a context-aware driver warning system

The difference between the two properties is that in the first one the printer is determined at the time of receiving the print request (`print(x)`) whereas in the second it is determined at the time of performing the print action (sending the file to the printer using action $y\langle x\rangle$).

As the state of the context (value of nearest printer) can change at any moment, the second property describes a more accurate context-aware system.

A mobile phone vibrates instead of ringing if the user is in a meeting

$$\mathbf{al}\,(\mathrm{m} \rightarrow [ring]\mathrm{ff})$$

Whenever the user is in a meeting (the system satisfies property m), the action *ring* is not enabled. This is a safety condition. We are not guaranteed that the phone will vibrate at all. For this, we need the following property:

$$\mathbf{al}\,[call]\ \mathbf{ev}\,((\langle vib\rangle tt \wedge \mathrm{m}) \vee (\langle ring\rangle tt \wedge \neg\mathrm{m}))$$

Which ensures that, after receiving a call, the phone will eventually either vibrate when in a meeting or ring when not in a meeting.

A mail client downloads new messages when a wireless connection is detected

Whenever a wireless connection is detected (status of wireless connection changes from *off* to *on*), the retrieval of new messages is initiated.

$$\mathbf{al}\,(\mathrm{IN}[\mathrm{woff}] \rightarrow ([-](\mathrm{IN}[\mathrm{won}] \rightarrow \mathbf{wh}\, \mathit{retnew})))$$

A web-service performs a search for nearby attractions based on the mobile device's location

After receiving a request from a device d, eventually the service sends back to d the list of attractions t nearest to the location l of d.

$$\mathbf{al}\,[\mathit{request}(d)]\mathbf{ev}\,\exists t, l.(\langle d\langle t\rangle\rangle tt \wedge \mathrm{IN}[(\mathit{loc}, d, l) \wedge (\mathit{attractions}, l, t)])$$

Summary

In this chapter, a logic for context-awareness based on the μ-calculus was given. The logic includes powerful modalities for the description of the agent behaviour which can be linked to the state of the context using a modality IN[ϕ]. a number of examples were given to show how the logic is used in modelling.

As future work, it will be interesting to investigate the link between the context-awareness logic given in this chapter and the process calculus $C\pi$ given in chapter 4. In particular it will be helpful to know if two equivalent agents in the calculus satisfy the same set of formulas in the logic and vice versa. We can also enlarge the scope of the logic to talk about networks of agents using a spatial modality $\|$. This can be useful in describing a context-aware system from a higher level point of view.

Chapter 6

Case study: The BluScreen System

In this chapter we demonstrate the use of the $C\pi$ calculus and its associated logic using an example of a ubiquitous computing system called BluScreen. BluScreen [35] is a system created at the University of Southampton for managing adverts shown on a public electronic display. It works by tracking the presence of devices near the display using Bluetooth and deciding which advert to show next depending on the exposure of the detected devices to each advert. To maximise the exposure to adverts, each advert is associated with an advertising agent which bids in an auction representing the right to display an advert in the current advertising cycle (figure 6.1). In our version we assume that a screen can have more than one *advertising space* allowing for more than one advert to be displayed on a single screen.

The following properties could be used to describe this system.

1. For each advertising space *sp*, the advertising agent with the highest bid on the auction of *sp* is allowed to advertise on *sp*.

2. If no (new) devices are currently nearby, the advertising agent does not display any advert.

3. At the end of each advertising cycle, only one agent gets to advertise on each advertising space.

4. An advertising agent cannot bid more than once in a cycle.

5. The bigger the number of new devices, the bigger the amount an agent bids for an advertising space.

In our model, we simplify by assuming that agents place random amounts as bids. In reality, a value function based on the number and value of nearby devices is used to decide on the amount to bid. Therefore, the last property is neither correct with random amounts nor with the value function. This is

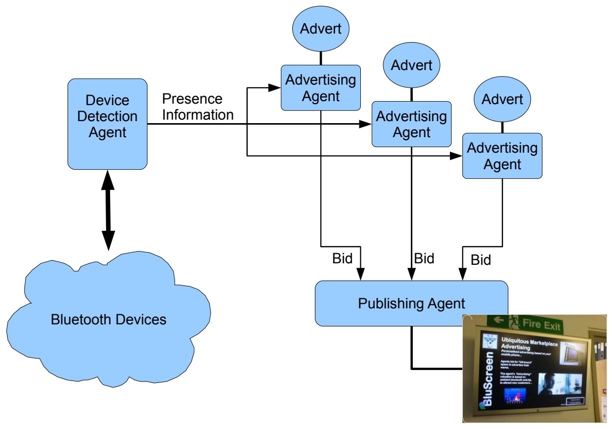


Figure 6.1: The BluScreen architecture

because an agent could choose to bid less in the presence of say two new devices as opposed to one new device if the value of the two devices is less. For example, an agent advertising a hotel in Paris might place more value on a device whose owner is planning to visit Paris in the near future.

In modelling the BluScreen system, some shortcomings were found in the calculus. For instance, process definitions taking parameters and recursive definitions did not exist in the calculus but were required in this case study. Also, some notation is assumed in the case study that could easily be added to the calculus. In particular, the actions $\langle i, s, +v\rangle$ and $\langle i, s, -v\rangle$ allow updating the current value of some context information through addition or subtraction. The behaviour is dependent on the type of the context information. In case it is a set, $+$ means set union and $-$ means set subtraction. This could be extended to any type of operation on context information.

BluScreen Model

Below, we model the BluScreen system using $C\pi$. To make the model more readable, we use the following abbreviations:

- "$if\ \{\tilde{x}\}.\phi\ then\ P\ else\ Q$" equivalent to $\{\tilde{x}\}.\phi \hookrightarrow P\ else\ Q$
- "$if\ \{\tilde{x}\}.\phi\ then\ P$" equivalent to $\{\tilde{x}\}.\phi \hookrightarrow P$

We note that the expression P in "$if\ \{\tilde{x}\}.\phi\ then\ P$" becomes enabled only when $\{\tilde{x}\}\phi$ becomes true.

Advertiser

$$
\begin{array}{lll}
A^i(s,ad) & \overset{def}{=} & [(\text{start},i,no),(\text{present},s,\emptyset),(\text{auctions},s,\emptyset),(\text{seenby},ad,\emptyset)] \triangleright \\
& & (\text{start},i,=yes). \\
& if & \{x,y\}.(\text{present},s,has(x)) \wedge (\text{seenby},ad,\neg has(x)) \wedge \\
& & \quad (\text{auctions},s,has(y)) \\
& then & (\nu c)(\\
& & \quad y\langle am,c\rangle.c(sp).sp\langle ad\rangle.(\text{present},s,x).\langle \text{seenby},ad,+x\rangle \\
& & \quad + \\
& & \quad if\ (\text{auctions},s,\neg has(y))\ then\ \mathbf{0} \\
& &)
\end{array}
$$

Each advertising agent is parameterised on the cycle number i, the screen s, and the advert it is associated with ad. The context of an advertising agent includes the devices that have been detected around the screen `(present, s)` and the history of device exposure to the advert `(seenby,ad)`. In addition, the context includes the current auctions available for advertising on screen s `(auctions,s)`. Agents know when to start bidding on auctions by the use of the context information `(start,i)`.

An advertising agent first checks whether any of the detected devices has not been exposed to the advert and selects an auction for an advertising space to show the advert on. After that, it bids on the selected auction. If successful, the agent receives the name sp representing the channel on which to send the advert to be shown on the screen.

Publisher

$$
\begin{array}{ll}
P^i(sc,sp,au) \overset{def}{=} & (\nu\ \text{maxbidder, maxbid, timer}) \\
& [(\text{start},i,no),(\text{timer},au,10),(\text{maxbidder},au,\bot),(\text{maxbid},au,0)] \triangleright \\
& \quad (\text{start},i,=yes).\langle \text{auctions},sc,+\{au\}\rangle.Q(sc,sp,au) \mid T(au) \\
Q(sc,sp,au) \overset{def}{=} & if\ (\text{timer},au,<0) \\
& then\ (\text{maxbidder},au,x).x\langle sp\rangle.\langle \text{auctions},sc,-\{au\}\rangle \\
& else\ au(bid,c). \\
& \quad if\ (\text{maxbid},au,<bid) \\
& \quad then\ \langle \text{maxbidder},au,c\rangle.\langle \text{maxbid},au,bid\rangle.Q(sc,sp,au)) \\
& \quad else\ Q(sc,sp,au) \\
T(au) \overset{def}{=} & \tau.\langle \text{timer},au,-1\rangle.T(au)
\end{array}
$$

Each advertising space is associated with a publishing agent. This agent announces an auction to advertising agents and determines the winner at the end of the auction. Therefore, it keeps track of the agent with the maximum bid so far in its context `(maxbidder,au)`. While the auction is active, the agent accepts new bids, which are then compared to the current maximum bid and the context updated accordingly. When the auction finishes, the publishing agent sends the channel representing the advertising space to the winning advertising agent. A process T keeps decrementing the value of timer. When this latter reaches below 0, the auction is considered to have finished.

Device detection

$$\begin{aligned} D^i(sc) &\stackrel{def}{=} [\,] \triangleright start^i().\langle \text{present}, sc, \emptyset\rangle.E^i(sc) \\ E^i(sc) &\stackrel{def}{=} findnext\langle t, f\rangle.(\\ &\quad t(x).\langle \text{present}, sc, +\{x\}\rangle.E^i(sc) + f().\langle \text{start}, i, yes\rangle \\ &\quad) \end{aligned}$$

The device detection agent scans for new devices around the screen and updates the context information `(present,s)` to reflect the changes. At the start of each cycle, the set of devices nearby the screen is reset using the action $\langle \text{present}, sc, \emptyset\rangle$, then scanning is performed through communication on channel `findnext`. A reply on channel t represents the detection of a new device, whereas a reply on channel f represents the failure to find any new device.

System

$$\begin{aligned} B_{n,m}(sc) &\stackrel{def}{=} [\,] \triangleright start^0\langle\rangle \parallel S^0_{n,m}(sc) \\ S^i_{n,m}(sc) &\stackrel{def}{=} D^i(sc) \parallel \prod_{j\in 1..n} A^i(sc, ad_j) \parallel \prod_{j\in 1..m} P^i(sc, sp_j, au_j) \parallel \\ &\qquad [\,] \triangleright sp_1(x).\ldots.sp_m(x).start^{i+1}\langle\rangle \parallel S^{i+1}_{n,m}(sc) \end{aligned}$$

The BluScreen system is modelled by the network of agents $B_{n,m}$. This network consists of an infinite number of networks $S^i_{n,m}(sc)$ each of which models an advertising cycle. For each cycle, there is one device detection agent $D^i(sc)$, n advertising agents $(\prod_{j\in 1..n} A^i(sc, ad_j))$, and m publishing agents $(\prod_{j\in 1..m} P^i(sc, sp_j, au_j))$ running in parallel. The notation $\prod_{j\in 1..n} A_i$ is short for $A_1 \parallel A_2 \parallel \cdots \parallel A_n$. The agent $[\,] \triangleright start^0\langle\rangle$ kick starts the first cycle and the agent $[\,] \triangleright sp_1(x).\ldots.sp_m(x).start^{i+1}\langle\rangle$ starts off the next cycle.

Properties

The following properties describe examples of the context-aware behaviour modelled by agents in the BluScreen system.

- In case no device is currently nearby, the advertising agent does not display any advert.

$$A^i(s, ad) \models \mathbb{X}(\text{IN}[(\text{present}, s, = \emptyset)] \rightarrow \forall x.\mathbf{al}\,[x\langle ad\rangle]\text{ff})$$

- In case no *new* devices are currently nearby, the advertising agent does not display any advert.

$$A^i(s, ad) \models \mathbb{X}(\text{IN}[\forall z.(\text{present}, s, has(z)) \rightarrow (\text{seenby}, ad, has(z))] \rightarrow \forall x.\mathbf{al}\,[x\langle ad\rangle]\text{ff})$$

- Only the advertising agent with the maximum bid for an auction gets to advertise.

$$P(sc, sp, au) \models \mathbf{al}\,(\forall x.\langle x\langle sp\rangle\rangle tt \rightarrow \text{IN}[(maxbidder, au, = x) \wedge (timer, sc, = 0)])$$

Summary

The BluScreen system is a context-aware system built at the University of Southampton. A collection of advertising agents represent adverts competing for space on a public display. These agents place bids depending on whether devices near the display have been exposed to their adverts before. We have attempted to model this system using $C\pi$. We think that our model successfully captures the essential elements of the context-aware behaviour in the BluScreen system.

Chapter 7

Conclusions

Entities in a ubiquitous computing environment must be aware of the context in which they operate. This is achieved through the use of sensors that provide the entity with appropriate data. Such data can be used by the system to reconfigure itself and possibly change its strategy and goals. For example, a mobile phone can change its profile to "silent" when entering a building where mobile phones are not allowed.

Many of the applications of context-awareness involve dependable systems which affect the daily life of users. Correctness of implementation is crucial and failures can have disastrous consequences (eg. in health care and military). Therefore, it is important to develop context-aware applications using a formal framework allowing the verification of correctness and safety properties.

Our attempt at modelling context-awareness serves as a first step towards a more powerful theory that takes into account many of the aspects of context-aware systems. More work is required especially in the areas of behavioural equivalence and property verification. Some further work is outlined in the next section.

7.1 Future work

Equivalences

Behavioural equivalences are a very useful tool in proving properties of systems. By checking that one agent is equivalent to another, we are asserting that they are exchangeable in any network. An interesting problem is finding the minimum subset C' of a context C for which $C \triangleright P \sim C' \triangleright P$. This can indicate the minimum required information to achieve the desired behaviour. Another equivalence (on processes) is $P \sim_\phi Q$ which means that under any context C satisfying property ϕ, processes P and Q are equivalent. In other words,

$$P \sim_\phi Q \quad \text{if} \quad \forall C : \mathcal{C}.\ C \models \phi \quad \text{implies} \quad C \triangleright P \sim C \triangleright Q$$

It would also be interesting to investigate the link between the context-awareness logic and the process calculus $C\pi$. In particular it would be helpful to know if two equivalent agents in the calculus satisfy the same set of formulas in the logic and vice versa.

Privacy and reliability of context information

Privacy is the inability to discover or misuse the identity of the user. The Common Criteria [32] lists four aspects of privacy namely: anonymity, pseudonymity, unlinkability, and unobservability.

Anonymity means that a piece of information cannot be attributed to the identity of the person to which it belongs. In $C\pi$, this means that a process does not differentiate between information (i, s, v) and (i, s', v) for all s, s'. For example, if system P behaves the same whether it knows $(location, user_1, r101)$ or $(location, user_2, r101)$ where $user_2$ is any user different from $user_1$, then we can say that the user is anonymous when in location $r101$. As an example, P could be the following process, which switches the light on in case a person is detected in location $r101$.

$$\exists x.(location, x, = r101) \hookrightarrow light\langle on\rangle$$

Another view of anonymity is that information (i, s, v) is not differentiated from (i, s, v') for any values v, v'. For example, if an electronic voting system behaves the same when $(vote, p_1, yes)$ and when $(vote, p_1, no)$, then we can say that the vote of person p_1 is anonymous.

Conclusion

Ubiquitous systems need to be aware of their surrounding context in order to integrate seamlessly with the user's tasks and environment. A system which has context-dependent behaviour with the aim of providing useful functionality to its user in different contexts is called a context-aware system.

In this thesis, we aimed to deliver a formal model of context-awareness that would make the following concepts clearer.

1. How can context be represented?
2. How is contextual information accessed?
3. How can we model the effects of context on the system's behaviour?

We think that we have achieved this objective by defining the new calculus $C\pi$ and a modal logic for context-awareness. We demonstrated this through various examples and models. The main characteristics of $C\pi$ include:

- Ability to model a system behaviour and how it affects and is affected by its context.

- Ability to define context-dependent behaviour using interpretation rules as well as conditional terms. The interpretation rules define context-dependency as an external effect whereas the conditional construct is internal to the process.
- Context information restriction can be used to define some degree of context information privacy. This is because it limits the scope in which the restricted information is accessible.

On the other hand, the modal logic for context-awareness has the following features:

- A distinction between the state of a system and its context. We can describe the properties of each part as well as the dependency between them.
- Ability to express guaranteed execution of an action.
- Logic is based on μ-calculus so it inherits its expressive power.

However, a lot of research remains to be done in this area some of which was outlined in the previous section. We hope the research we have undertaken and the results we have achieved will be of benefit to the context-awareness and ubiquitous computing research communities. Further research can build on our results by enhancing the logic and the calculus or implementing tools to simulate the execution of models built using the calculus and validate them using the logic.

Bibliography

[1] V. Akman and M. Surav. Steps toward formalizing context. *AI Magazine*, 17:55–72, 1996. 22

[2] BBC(news). *Drive safely, pay less.* 2000. Published: `http://news.bbc.co.uk/1/hi/sci/tech/861240.stm`. 4

[3] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli. Context-aware middleware for resource management in the wireless internet. *Software Engineering, IEEE Transactions on*, 29:1086–1099, 2003. 25, 29

[4] P. A. Bernstein. *Middleware: a model for distributed system services*, volume 39. ACM Press, 1996. 23

[5] L. Birkedal, S. Debois, E. Elsborg, T. Hildebrandt, and H. Niss. Bigraphical Models of Context-Aware Systems. *FOSSACS 06: Proceedings of 9th International Conference on Foundations of Software Science and Computation Structures*, 3921, 2006. 31, 37

[6] P. Blackburn, M. de Rijke, and Y. Venema. *Modal logic.* Cambridge University Press New York, NY, USA, 2001. 15

[7] R. Boyer and W. Griswold. Fulcrum - an open-implementation approach to internet-scale context-aware publish / subscribe. In *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on*, page 275a, 2005. 23, 25, 30

[8] J. Bradfield and C. Stirling. Modal logics and mu-calculi: an introduction. *Handbook of Process Algebra*, pages 293–330, 2001. 18, 19

[9] P. Braione and G. P. Picco. On calculi for context-aware coordination. *COORDINATION 04: Proceedings of the 6th International Conference on Coordination Models and Languages*, 2949:38–54, 2004. 31, 36, 38

[10] P. Brown, W. Burleston, M. Lamming, O. Rahlff, G. Romano, J. Scholtz, and D. Snowdon. Context-awareness: Some compelling applications. *Proceedings the CH12000 Workshop on The What, Who, Where, When, Why and How of Context-Awareness*, 2000. 5

[11] S. Buvac and I. A. Mason. Propositional logic of context. *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 412–419, 1993. 22

[12] L. Cardelli and A. D. Gordon. Anytime, anywhere: modal logics for mobile ambients. *Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 365–377, 2000. 34

[13] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240:177–213, 2000. 8, 9, 11

[14] L. Cardelli and A. D. Gordon. Ambient logic. 2003. Microsoft Research. 19

[15] D. Chalmers et al. *Ubiquitous Computing: Experience, Design and Science.* 2006. Published: `http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/Manifesto/manifesto.pdf`. 4

[16] A. Chan and S. Chuang. Mobipads: a reflective middleware for context-aware mobile computing. *IEEE Transactions on Software Engineering*, 29:1072–1085, 2003. 25

[17] B. Clarkson, K. Mase, and A. Pentland. Recognizing user context via wearable sensors. In *Wearable Computers, 2000. The Fourth International Symposium on*, pages 69–75, 2000. 20

[18] B. Clarkson, N. Sawhney, and A. Pentland. Auditory context awareness via wearable computing. *Energy*, 400:600. 20

[19] G. Conforti, D. Macedonio, and V. Sassone. Bilog: Spatial logics for bigraphs. *Computer Science Report*, 2, 2005. 19

[20] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. *Lecture Notes in Computer Science*, 1995:18–38, 2001. 29

[21] A. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5:4–7, Feb. 2001. 20

[22] A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. *CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness*, 2000. 20

[23] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16:97–166, 2001. 5

[24] A. K. Dey, D. Salber, G. D. Abowd, and M. Futakawa. The conference assistant: combining context-awareness with wearablecomputing. *Wearable Computers, 1999. Digest of Papers. The Third International Symposium on*, pages 21–28, 1999. 20

[25] Exif.org. an unofficial site dedicated to exif. `http://exif.org/`. Last accessed 20 Aug. 2007. 21

[26] P. Fahy and S. Clarke. Cass-middleware for mobile context-aware applications. *Mobisys 2004 Workshop on Context Awareness*, 2004. 25

[27] C. Fournet and G. Gonthier. The join calculus: a language for distributed mobile programming. *Applied Semantics. International Summer School, APPSEM*, page 268?332, 2000. 34

[28] K. Fujinami and T. Nakajima. Sentient artefacts: Acquiring user's context through daily objects. *Lecture Notes in Computer Science*, 3823:335, 2005. 20

[29] A. Held, S. Buchholz, and A. Schill. Modeling of context information for pervasive computing applications. *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI)*, 2002. 22

[30] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger. *Context-Awareness on Mobile Devices - the Hydrogen Approach*. IEEE Computer Society, 2003. 27

[31] P. Holleis, M. Kranz, M. Gall, and A. Schmidt. Adding context information to digital photos. In *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, pages 536– 542, 2005. 20

[32] ISO. Common criteria, iso/iec/15408, part 2 security functional components, 2006. `http://www.niap-ccevs.org/cc-scheme/cc_docs/CCPART2V3.1R1.pdf`. Last accessed 02 Jun. 2007. 74

[33] C. Julien and G. Roman. *EgoSpaces: Facilitating Rapid Development of Context-Aware Mobile Applications*, volume 32. IEEE Press, 2006. 23, 25, 27, 28

[34] L. Kagal, V. Korolev, H. Chen, A. Joshi, and T. Finin. Centaurus: A framework for intelligent services in a mobile environment. *Proceedings of the International Workshop on Smart Appliances and Wearable Computing (IWSAWC)*, 2001. 22

[35] M. Karam, T. Payne, and E. David. Evaluating BluScreen: Usability for Intelligent Pervasive Displays. *Pervasive Computing and Applications, 2007. ICPCA 2007. 2nd International Conference on*, pages 18–23, 2007. 68

[36] M. B. Kjaergaard and J. Bunde-Pedersen. A formal model for context-awareness, 2006. `http://www.brics.dk/RS/06/2/BRICS-RS-06-2.pdf`. Last accessed 23 Oct. 2007. 31, 35, 36

[37] M. Korkea-aho. Context-aware applications survey. *Helsinki University of Technology, Helsinki, Internetworking Seminar April*, 25, 2000. 20

[38] A. Kurz. *Coalgebras and Modal Logic*. 2001. Published: Available online at http://www.cs.le.ac.uk/people/akurz/CWI/public_html/cml.ps. Last accessed 23 Oct. 2007. 16

[39] M. Kwiatkowska, R. Milner, and V. Sassone. Science for global ubiquitous computing. *Bulletin of the EATCS*, 82:325–333, 2004. 4

[40] M. G. Lamming, P. J. Brown, et al. The design of a human memory prosthesis. *Computer Journal*, 37:153–163, 1994. 6

[41] J. J. Leifer. *Operational Congruences for Reactive Systems*. University of Cambridge, Computer Laboratory, 2001. PhD thesis. 36

[42] F. Manola and E. Miller. Rdf primer. *W3C Recommendation*, 10, 2004. http://www.w3.org/TR/REC-rdf-syntax/. Last accessed 23 Oct. 2007. 41, 57

[43] J. McCarthy. Notes on formalizing context. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1:555–560, 1993. 22

[44] S. Meyer and A. Rakotonirainy. A survey of research on context-aware homes. *Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003-Volume 21*, pages 159–168, 2003. 20

[45] R. Milner. *Communicating and Mobile Systems: The pi-calculus*. Cambridge University Press, 1999. 9

[46] R. Milner. Bigraphical reactive systems: basic theory, 2001. http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-523.pdf. Last accessed 23 Oct. 2007. 12

[47] R. Milner. Pure bigraphs: Structure and dynamics. *Information and Computation*, 204:60–122, 2006. 12, 14

[48] R. Milner et al. Theories for ubiquitous processes and data. Platform for a 15-year Grand Challenge, 2005. 4

[49] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, i. *Information and Computation*, 100:1–40, 1992. 8

[50] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, ii. *Information and Computation*, 100:41–77, 1992. 8

[51] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993. 19

[52] A. L. Murphy, G. P. Picco, and G. C. Roman. Lime: A middleware for physical and logical mobility. *Proceedings of the 21 stInternational Conference on Distributed Computing Systems*, page 524?533, 2001. 37

[53] J. Parrow. An introduction to the pi-calculus. *Handbook of Process Algebra*, 543, 2001. 8

[54] K. V. S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25:285–327, Dec. 1995. 51

[55] G.-C. Roman, C. Julien, and J. Payton. *Modeling adaptive behaviors in Context UNITY*, volume 376. Elsevier Science Publishers Ltd., 2007. 31, 33

[56] G.-C. Roman and P. McCann. *A Notation and Logic for Mobile Computing*, volume 1 of *Formal Methods in System Design*. 2002. 31

[57] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces. *Pervasive Computing, IEEE*, 1:74– 83, 2002. 25

[58] N. Ryan. Contextml: Exchanging contextual information between a mobile client and the fieldnote server. *Project Homepage. http://www. cs. ukc. ac. uk/projects/mobicomp/fnc/ConteXtML. html*. 22

[59] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. pages 434–441, 1999. 25, 26

[60] Sansatex. *Sansatex Smartshirt*. `http://www.sensatex.com`. 4

[61] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on*, pages 85–90, 1994. 5, 21, 41

[62] A. Schmidt, M. Beigl, and H. W. Gellersen. There is more to context than location. *Computers & Graphics*, 23:893–901, 1999. 20

[63] C. Stirling. *Modal and Temporal Properties of Processes*. Texts in Computer Science. Springer, 2001. 60, 62

[64] T. Strang and C. Linnhoff-Popien. *A Context Modeling Survey*. 2004. 23

[65] W3C. N-triples. `http://www.w3.org/2001/sw/RDFCore/ntriples/`. Last accessed 14 Aug. 2007. 21

[66] W3C. Resource description framework (rdf), 2004. `http://www.w3.org/RDF/`. Last accessed 14 Aug. 2007. 21

[67] W3C. Composite capabilities/preferences profile, 2007. `http://www.w3.org/Mobile/CCPP/`. Last accessed 14 Aug. 2007. 22

[68] Wireless Application Forum. *WAG UAProf, Wireless Application Protocol*, 2001. `http://www.openmobilealliance.org/tech/affiliates/wap/wap-248-uaprof-20011020-a.pdf`. Last accessed on 30 Jun. 2008. 22

[69] S. Zachariadis, C. Mascolo, and W. Emmerich. Satin: A component model for mobile self-organisation. *International Symposium on Distributed Objects and Applications (DOA), Agia Napa, Cyprus, October*, 2004. 25, 35

[70] E. N. Zalta. *Basic Concepts in Modal Logic*. 1995. Published: Online at `http://mally.stanford.edu/notes.pdf`. Lecture notes. Last accessed 23 Oct. 2007. 15, 16

[71] P. Zimmer. *A Calculus for Context-Awareness.* BRICS Report Series, 2005. Draft. Available at `http://citeseer.ist.psu.edu/728725.html`. Last accessed 21 Aug. 2007. 31, 34