Basic Research in Computer Science

A Formal Model for Context-Awareness

Mikkel Baun Kjærgaard
Jonathan Bunde-Pedersen

BRICS Report Series **RS-06-2**

ISSN 0909-0878 **February 2006**

See back inner page for a list of recent BRICS Report Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
IT-parken, Aabogade 34
DK–8200 Aarhus N
Denmark

Telephone: +45 8942 9300
Telefax: +45 8942 5601
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/06/2/`

Conawa: A Formal Model for Context Awareness

Mikkel Baun Kjærgaard Jonathan Bunde-Pedersen

February 2, 2006

Abstract

There is a definite lack of formal support for modeling realistic context-awareness in pervasive computing applications. The Conawa calculus presented in this paper provides mechanisms for modeling complex and interwoven sets of context-information by extending ambient calculus with new constructs and capabilities. In connection with the calculus we present four scenarios which are used to evaluate Conawa. The calculus is a step in the direction of making formal methods applicable in the area of pervasive computing.

1 Introduction

In the area of pervasive computing the great vision is moving computation from the desktop computer to a number of devices embedded in the environment of the user. The applications put forward for these devices depend heavily on the notion of context awareness.

However, previous research in this field has failed to consider the true nature of context by only incorporating very simple notions of it. It would thus be of interest to define a calculus which captures the context as described in [10, 5]. Earlier attempts as [2, 9] have only dealt with very limited notions of context.

The modeling of context is both an issue when working with a practical approach to pervasive computing as well as the more theoretical approach labeled among other things as 'Global Computing´. This report describes a context-model based on 'Ambient Calculus´ [3] which

is more realistic and capable of describing the scenarios given as part of the case in section 1.1.

Ambient calculus situates an ambient in a tree and permits it only to interact with nearby[1] ambients. If context-information should be made available to an ambient it must be positioned near the ambient in the tree – but this is not necessarily a good idea. The problem is the very nature of context which can be anything from the actual physical location of the ambient to logical information completely independent of anything else. These two independent pieces of context is therefore hard to model in a single tree. Modeling context in a flat structure is possible (see [2, 9]) but this approach suffers from the simple structure, which means that it is difficult to express and navigate the contextual information.

The main contribution of this report is to define the Conawa calculus which incorporates a rich notion of context. In addition it is shown how this calculus can be applied to applications in the area of pervasive healthcare. The structure of this report is as follows: Firstly, requirements for a rich notion of context are outlined and related work is presented. Secondly, the Conawa Calculus is defined first by presenting the conceptual elements, afterwards by defining syntax and semantics of the calculus. Thirdly, it will be shown how the calculus can be applied to applications in the area of pervasive healthcare. Finally, design choices, discussion and a conclusion will be given.

1.1 Evaluation

To evaluate the applicability of a calculus for modeling Context-aware computing applications we want to outline some basic requirements for it. These requirements are based on the four kinds of Context-Aware Computing Applications presented in [10]. We want to express these four requirements in four concrete scenarios. This makes it easier to evaluate if a calculus can be used to model context appropriately. The four categories in [10] are the product of two points along two orthogonal dimensions. One of the dimensions describes if a task at hand is getting information or is carrying out a command; the other dimension describes if it is effected manually or automatically (see figure 2). The four concrete scenarios are inspired from the idea of an Aware-Phone described in [1]. The Aware-Phone is an application implemented on a mobile phone which supports context-mediated social awareness.

[1] Parents or siblings

1.1.1 Context information

This requirement represents the category of getting information which is manually effected. The idea is that when a query is executed the answer is adapted based on the current context. This is summarized in the following scenario in terms of the Aware-Phone application.

A young doctor or a nurse needs to contact a more experienced doctor to consult him on some issue. So the Aware-Phone is queried for where the nearest doctor which is not occupied by some other work task is located. The application then returns the best suited doctor in the current context of the location and activities of the doctors on duty.

1.1.2 Context Commands

This requirement represents the category of carrying out a command which is manually effected. The idea is that when some command is executed the execution depends on the current context. This is summarized in the following scenario:

A doctor using the Aware-Phone application receives a message containing an X-ray image which is to large to be shown on the mobile-phone's small display. The doctor then commands the application to show the image at the nearest available wall-sized display.

1.1.3 Automatic Contextual Reconfiguration

This requirement represents the category of getting information which are automatically effected. The idea is that a system automatically reconfigures its structure or behavior based on the context. In practice reconfiguration can be done by adding new components, removing existing components, or altering the connections between components. This is summarized in the following scenario:

A doctor has approached a wall-sized display and carried out some login procedure. Based on the current context the system have been reconfigured to show information based on his current activities. For example it could show an X-ray which the doctor have indicated in the Aware-Phone application that he wants to examine.

1.1.4 Context-Triggered Actions

This requirement represents the category of carrying out a command which is automatically effected. The idea is that when the context en-

ters a certain state some special command is to be executed. This is summarized in the following scenario:

A doctor approaches a wall-sized display where he is automatically confronted with some login step. After using the display the doctor walks away from the display whereby he is automatically logged out.

1.2 Ambients in multiple trees

With this calculus it is our idea to model context using several ambient like trees. There will be one context-tree for each category of context-information, for instance one tree for location-information, one for activities and one for types of printers. An ambient which represents an entity or an application will have a presence in one or more trees, for instance a printer is both present in the location tree and the printer-types tree as shown in figure 1. We realize that an entity cannot physically be part of more than one tree, but propose to use a solution inspired from bi-graphs [6] where links (a la pointers) are used to indicate presence in a context tree. An ambient which were to utilize the context information would then navigate the relevant trees in order to specify preferences and indicate which context it was interested in. An example could be moving in both the location and printer-type tree which would give the ambient a new physically closest printer but also a new preference in printer-type. We propose to extend existing ambient-constructs such as *in* and *out* to enable navigating in several contexts at once and also extending the basic tree-structure to use bi-graph-like (tree)structures.

2 Related work

2.1 On Context and Context-awareness

The term context-awareness was first introduced by the researchers in Xerox Parc laboratories [10] in 1994. They categorize context-awareness along two axis as seen in figure 2.

They furthermore describe context-aware systems as adaptable, distributed and pervasive. This definition is used to present a system which is context-aware in the sense that it is location-aware. We have used the concepts in figure 2 to provide scenarios that encompass their idea of context. We in contrast have not limited ourselves to location, as it was, and is, our goal to provide a richer notion of context-awareness.

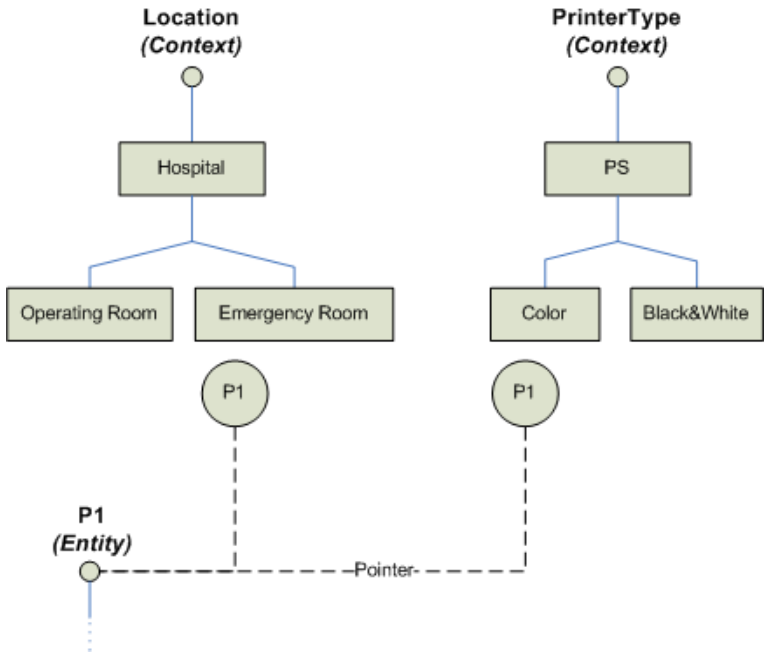

Figure 1: Illustration of one printer (entity) and the contexts which it is part of

	manual	automatic
information	proximate selection & contextual information	automatic contextual reconfiguration
command	contextual commands	context-triggered actions

Figure 2: Context

The conviction that context is more than location was fueled by several surveys on context and context-awareness [11, 4, 8]. The modeling and use of context is the focus of these surveys. Location is an important part of context, but models should allow for more elaborate constructions in which the term of "proximity" is determined by the context to which it refers. We have focused on modeling context in this way, providing a very expressive model.

The AWARE-architecture [1] is a framework for *context-mediated social awareness*. Basically, this concept allows a device (an AwarePhone) to consider context and internal configuration to mediate communication between itself and other devices. For instance, sending an instant message to a doctor is only allowed if the context state is right (ie. the doctor is not operating) and the configuration allows it (ie. the doctor

has set his status to 'free'). We have used the AwarePhone concept as a foundation for our mobile ambients and also as an inspiration for our scenarios.

2.2 On Calculi

A formal model based on the UNITY model which incorporates a notion of context is described in [7]. The paper presents an enhancement of Mobile UNITY called *Context UNITY*. The model describes 'programs' which can obtain context information through variables whose values react to and thereby depend on the context in which the program is executing. Their work results in a framework which enables programmers to write applications which have a 'sense of context'. Furthermore it is firmly grounded in the formalisms of UNITY which allows certain properties of applications to be formally investigated.

Bi-graphs [6] are graph-based structures which have similarities with our structuring technique for context. Two kinds of structures are present in a single graph, hence the name bi-graphs. Firstly, nodes can be nested inside each other providing *depth* and representing *locality*. Secondly, nodes may have ports which are connected to other ports by links. This represents the linked structure of the bi-graph. Furthermore, reaction rules are used to express dynamic properties. Our ambient type which represents entities or applications represents a similar *overlay structure* on graphs (in our case trees) but we use this secondary structure to express complex context constellations.

Furthermore, we have based our syntax and semantics heavily on the "Ambient Calculus" [3] and inspiration on calculi and context from "On Calculi for Context-Aware Coordination" [2].

3 Concepts

In this section the fundamental concepts of our calculus is motivated. The calculus can be seen as a generalization of the ambient calculus[3]. In the ambient calculus computation is situated into a hierarchy of ambients which can be used to represent virtual or physical barriers. But from a modeling perspective this is not enough to model the applications considered in the area of context-aware computing. Here computation is seen as embedded in a number of contexts at the same time. In our calculus the notion of ambients found in the original calculus is used as the

fundamental building block, but it is extended to be situated in contexts.

3.1 Contexts

As motivated above the ambients in this calculus are situated in one or more contexts. The contexts are used to capture virtual and physical barriers as in the original calculus, but also for context-modeling. In figure 3 an example is shown which illustrates the use of ambients embedded in two contexts. To denote that an ambient is part of multiple context the # character is used followed by the name of the ambient which itself is listed below the contexts. When an ambient is situated in a context it is capable, as in the original calculus, of interacting with other ambients in this specific context. In each of the contexts which an ambient is part of, it forms a subtree of that context. To ensure consistency we restrict ambients only to be present in one place in all contexts.

So communication between ambients is as in the original calculus constrained by how the ambient is situated. The context is built as a hierarchy based on ambients which are used to model different type of information. In this calculus we have restricted the contexts to be trees because most relevant context information can in some form be modeled as trees.

Two contexts are used in the example; one to represent the virtual barriers of a network and one to represent physical location. Two ambients are used to model two devices which are situated in the network and at some location. The example illustrates the ideas but is very simple – more complete examples will be given later on.

```
1 Contexts:
2 Network:[subA[#Device1] | subB[#Device2]]
3 Location:[BuildingA[#Device1 | #Device2] BuildingB[]]
4
5 Ambients:
6 Device1:[]
7 Device2:[]
```

Figure 3: Context example

3.2 Actions and capabilities

When ambients are situated in multiple trees the actions and capabilities of ambients as defined in the original calculus need to be reformulated. This reformulation is in our calculus based on adding restrictions to actions and capabilities. The restrictions are simple boolean expressions over contexts. By using these restrictions it is possible to model that an *in* capability should only be done in some specific context and not in others. Another example is that it is possible to output a new ambient only in a specific context.

4 Syntax

In developing our syntax we have focused on constructs which make it possible to describe and navigate context information. Therefore we have limited the syntax in areas such as scope of names and general output paths. The syntax is described formally in table 1 where c denotes a name of a context and n a name.

A description in this calculus consist of two parts. The first part describes a number of contexts which are placeholders for context information. They are ambient structures and represent context information as hierarchical and independent sets of information. The ambient structures of the contexts are restricted in their capabilities as well as structure compared to original ambients in [3]. These restricted ambients called *context ambients* are static in the sense that they do not have capabilities to move, but only to be created or opened. The second part consist of a number of *reference ambients* which can be used to denote entities which are embedded in certain contexts. These ambients can be present in contexts by a reference which is only a "pointer". This means that a reference ambient can be represented in multiple contexts[2]. The reference ambients have capabilities which allows them to move in and out of the different contexts whereby they can navigate the context information.

We have kept the context and reference ambients consistent with those of Cardellis on the syntactic level. The only exception is reference ambients which can be positioned in several contexts. To make this possible a reference is used to position a reference ambient n in a context which is denoted by $\#n$ in the context structure.

[2]But only one representation per context for consistency reasons

A ::=	$C; R$	separation of contexts and reference ambients
C ::=	$c : [CP] \mid C$	context have unique names
R ::=	$n : [RP] \mid R$	reference ambients have unique names
CP ::=	$n[CP]$	context ambient
	$CP \mid CP$	parallel
	0	inactivity
	CM.CP	exercise capability
	$\#n$	reference to reference ambient
CM ::=	coenter N	may enter if N matches
	coexit N	may exit if N matches
	open n	open ambient named n
RP ::=	$n[RP]$	reference ambient
	$RP \mid RP$	parallel
	$!RP$	replication
	0	inactivity
	(n).RP	input locally, bind to n
	$\langle$n$\rangle$	output locally (asynchronous)
	RM.RP	exercise capability
RM ::=	in $\{B\}$ N	enter if B and N matches
	out $\{B\}$	exit where B matches
	enter $\{B\}$ N	observable enter if B and N matches
	exit $\{B\}$	observable exit where B matches
	coenter $\{B\}$ N	may enter if N and B matches
	coexit $\{B\}$ N	may exit if N and B matches
	open $\{B\}$ n	open any n where B matches
B ::=	$* \mid c \mid \neg c \mid c \wedge B$	boolean expression, matches one or more contexts
N ::=	$n \mid ?$	names of ambients

Table 1: Syntax

$C1 : [\#n| \ldots]$
$n : [\ldots]$

Figure 4: n is referenced in *C1*

The syntax for the reference ambient's capabilities *in*, *out*, and *open* have been extended with braces { and } to indicate in which context or contexts the capability should be exercised. The context will not be aware of any ambients entering or leaving it this way. The context ambient can use the cocapabilities *coenter* and *coexit* to be able to detect the coming and going of referenced ambients, if they do so by using *enter* and *exit*. The expression between the braces can be a boolean expression of the form given as B. An example is given in figure 5. The example will match all ambients which are present in the same context in the *Location* and *Activity* contexts but is a different type of device. By not explicitly naming the ambient on which to perform the capabilities and using the wildcard-charater '?', we can model a simple matching mechanism.

$\ldots in\{Location \wedge Activity \wedge \neg DeviceType\}$?

Figure 5: Simple matching mechanism

This section has focused on extending the standard ambient syntax with constructs and capabilities which lets ambients navigate in complex context information. For a more thorough explanation of the standard ambient syntax the reader is referred to [3].

5 Semantics

In this section the semantics of the Conawa calculus will be defined. The calculus has two kinds of ambients for which the semantic has to be defined. There is context ambients which behave as a restricted form of ambients and reference ambients which extend the ambient semantic. Only reduction rules will be defined in this section. Structural congruence will not be defined or discussed in this report but we believe it is straight forward to extend the original definition to our calculus.

The context ambients are restricted mainly in terms of their capabilities. Context ambients have the same semantic as ambients in connection with the following constructs; parallel, inactivity, exercise capability and open. The semantic of the last two capabilities *coenter* and *coexit* is

defined below by the rules for the enter and exit capability of reference ambients.

The semantics of reference ambients extends the ambient semantics of several capabilities. The extended semantic of *in*, *out*, *enter*, *exit*, *coenter* and *coexit* are defined below. The semantic of the constructs will have small variations depending on whether a reference ambient is placed in another reference ambient or in a number of contexts by references. In connection with the following constructs, reference ambients have the same semantics as ambients: parallel, replication, inactivity, input, output, exercise capability and *open*.

5.1 *in* capability of reference ambients

The semantics of *in* will be presented in two rules. These rules do not handle all cases but give the idea of how the full semantic could be defined.

The first rule handles the simple case of one reference ambient which performs and *in* in one context. The ellipsis character "..." is used to indicate that the ambient and references can be arbitrary deeply nested within the context. The rule is shown in figure 6. Generally this rule would need to be extended to handle more contexts and the *?* which makes it possible to make a move into a arbitrary ambient.

$C : [...\#ra|ca[P]...]$
;
$ra : [in\{C\}ca.Q|R]$
$\longrightarrow$
$C : [...ca[\#ra|P]...]$
;
$ra : [Q|R]$

Figure 6: The rule for *in* in one context

The rule in figure 7 defines the semantic for the move of reference ambient into another reference ambient. In the rule the two reference ambient are parallel in contexts *C1* to *Cr* and not in contexts *D1* to *Ds*. Some subset of these contexts are used to limit the move where the ambients should be siblings in *Ci* to *Cj* and not in *Dk* to *Dl*. The rule also reflects that the moving reference ambients references will be removed from the contexts when the *in* is preformed.

$C1 : [...\#ra1|\#ra2|P_{C1}...]$
...
$Cr : [...\#ra1|\#ra2|P_{Cr}...]$
$D1 : [...\#ra1|P_{D1}...]$
...
$Ds : [...\#ra1|P_{Ds}...]$
;
$ra1 : [in\{Ci \wedge ... \wedge Cj \wedge \neg Dk \wedge ... \wedge \neg Dl\}ra2.Q|R]$
$ra2 : [S]$
$\longrightarrow$
$C1 : [...\#ra2|P_{C1}...]$
...
$Cr : [...\#ra2|P_{Cr}...]$
$D1 : [...P_{D1}...]$ ... $Ds : [...P_{Ds}...]$
;
$ra2 : [ra1[Q|R]|S]$

Figure 7: The rule for *in* with multiple contexts and reference ambients

If you replace *ra2* with *?* in the first line you have the rule for the $in\{B\}?$ case. When a reference ambients is embedded in another reference ambient, the original ambient rule can be used with the constraint that the condition have to contain a asterisk (*). If * is used for B then a move in all contexts needs to be made. If a reference ambient is not present in a context and an *in* is made, it will be placed in the top-level of the context.

5.2 *out* capability of reference ambients

In order to define the semantics of the *out* capability, two rules will be given. One considers the case of a reference ambient performing an *out* in a number of contexts. The other handles the case of a reference ambient making an *out* of another reference ambient. In figure 8 the first rule is shown. It defines how a reference ambient *ra1* can make an *out* in context *C1* to *Cr*.

The second rule defines the semantic when a reference ambient embedded in another reference ambient performs an *out* in a number of contexts. The rule is shown in figure 9. There are two things worth noticing. The first is that negation in the condition on the *out* has no effect, because it just means the same as not including the condition

$C1 : [...ca1[\#ra1|P_{C1}]|Q_{C1}...]$
...
$Cr : [...car[\#ra1|P_{Cr}]|Q_{Cr}...]$
;
$ra1 : [out\{Ci \wedge ... \wedge Cj\}.R|S]$
$\longrightarrow$
$C1 : [...\#ra1|ca1[P_{C1}]|Q_{C1}...]$
...
$Cr : [...\#ra1|car[P_{Cr}]|Q_{Cr}...]$
;
$ra1 : [R|S]$

Figure 8: The rule for *out* in multiple contexts

on that specified context. The second thing is, that an *out* cannot be performed on a context with a reference located at the root of the specified context. Therefore a reference ambient cannot remove itself from a context in other ways than doing and *in* on another reference ambient.

5.3 Observable capabilities

The rules for *in* and *out* have been defined above but in the calculus we have also included capabilities which adds observability to the two types of capabilities. This is done with the *enter*, *coenter*, *exit* and *coexit* capabilities. In this section a rule for *enter* and *coenter* will be defined in terms of an extension of the above rule for *in* which is shown in figure 10. For the other capabilities rules need to be extended in the same way as for *enter* and *coenter*.

6 Scenarios

In this section the scenarios in section 1.1 will be formaly described using the proposed Conawa calculus.

6.1 Context information

The scenario described in section 1.1.1 is in figure 11 shown formalized in the Conawa calculus. The formalized scenario is in figure 12 also

$C1 : [...\#ra1|P_{C1}...]$
...
$Cr : [...\#ra1|P_{Cr}...]$
$D1 : [...\#ra1|P_{D1}...]$
...
$Ds : [...\#ra1|P_{Ds}...]$
;
$ra1 : [ra2 : [out\{C1 \wedge ... \wedge Cr \wedge\}.Q|R]S]$
$\longrightarrow$
$C1 : [...\#ra1|\#ra2|P_{C1}...]$
...
$Cr : [...\#ra1|\#ra2|P_{Cr}...]$
$D1 : [...\#ra1|P_{D1}...]$
...
$Ds : [...\#ra1|P_{Ds}...]$
;
$ra1[S]$
$ra2 : [Q|R]$

Figure 9: The rule for *out* in multiple contexts with reference ambients

shown in a graphical depiction. The interpretation of the formalized scenario is as follows:

1. The subambient of AP2 performs an $out\{^*\}$ which places it in all contexts of AP2 (as a sibling)

2. $out\{Person \vee Status \vee Entity\}$ moves FNDAP2 upwards in the given contexts

3. Then the $in\{Person\}$ *Doctor* moves it into the 'Doctor' branch of the context 'Person'

4. Similarly, $in\{Status\}$ *Busy* makes FNDAP2 a child of 'NotBusy'

5. Then the "agent" enters the ambient which matches $Person \vee Status \vee Location \vee \neg Entity$ which happens to be AP1

The scenario goes on a bit further and the result is that FNDAP2 returns to AP2 with the ambient name 'Hans' as the nearest *and* available doctor. We have not depicted this, because the main motivation for the scenario is finding the doctor.

$C : [...\#ra|ca[coenter.P|Q]...]$
;
$ra : [enter\{C\}ca.R|S]$
$\longrightarrow$
$C : [...ca[\#ra|P|Q]...]$
;
$ra : [R|S]$

Figure 10: The rule for *enter* and *coenter* in a single context

6.2 Context commands

The scenario described in section 1.1.2 is in figure 13 shown formalized in the CONAWA calculus. The object of the scenario is to *use* the nearest and largest screen to display an image. We have used an inner ambient (SONS) which act as an 'agent' wrapping the image.

1. *SONS* moves out of *AP2*.
2. It shifts *Status*-context into *Free*. Afterwards in context *Screens* into *Large*.
3. Then we match any ambient in all contexts except *Entity*, which will be *Large1*.
4. *Large1* lets SONS in and opens it.
5. The image is outputted (by *SONS*) and caught by *Large1*, which finally displays it.

6.3 Contextual reconfiguration

The scenario described in section 1.1.3 is in figure 14 shown formalized in the CONAWA calculus. The scenario differs from the previous in that the screen responds to changes in its context, ie. the entering of the AwarePhone. The screen is augmented with the ability to display X-Rays by moving code (display-process) from the AwarePhone.

1. AwarePhone *AP1* logs into Screen *Large1* (use of the enter and coenter capabilities respectively)

```
Entity:[ Awarephones[#AP1 | #AP2] ]
Person:[ Doctor[#AP1] | Nurse[#AP2] ]
Status:[ Busy[#AP2] | NotBusy[#AP1] ]
Location:[ Hospital[ Ward1[#AP1 | #AP2] | Ward2[] ] ]
;
AP1:[!<Hans>]
AP2:
[
 //Find Nearest Doctor
 FNDAP2
 [
  out{*}.out{Person&Status&Entity}.in{Person} Doctor.in{Status} NotBusy
  .in{Person&Status&Location&~Entity} ?
  .GetMsg[out{*}.(name).enter FNDAP2.<name>]
  .coenter{*} GetMsg.open GetMsg.in {Entity} Awarephones.enter {Entity} AP2]
  |
  coenter{*} FNDAP2.open FNDAP2
  |
  !<Grethe>
 ]
]
```

Figure 11: Example based on 1.1.1

2. We have described in the previous scenarios how agents are dispatched to locate other ambients, therefore we omit the details for "goto AP1" (and similarly for "return ...")

3. Having arrived in *AP1* the *TransportAgent* loads a passenger: *DisplayAgent*

4. The transport returns and opens the passenger

5. The display-process runs displaying on the large screen

6.4 Context-Triggered actions

The scenario described in section 1.1.4 is in figure 15 shown formalized in the CONAWA calculus. The scenario shows how certain capabilities can be set to trigger by events in the context of the ambient, ie. when another ambient enters.

1. The *enter* in *AP1* is executed alongside with the *coenter* in *AreaLarge1*; this places *AP1* inside *AreaLarge1*

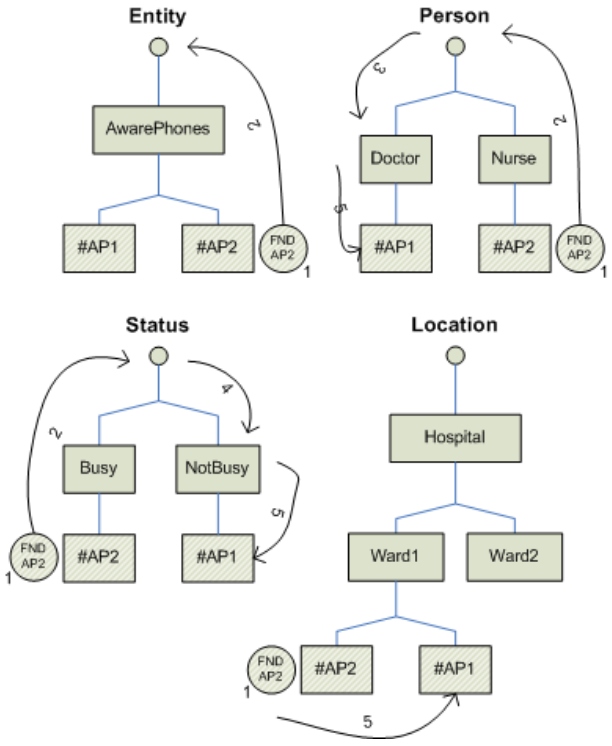

Figure 12: Illustration of scenario 1

```
Entity:[ Awarephones[#AP1 | #AP2] ]
Screens:[ Small[#Small1] | Medium[] | Large[#Large1 | #Large2 ] ]
Status:[ InUse[#Small1] | Free[#Large1 | #Large2] ]
Location:[ Hospital[ Ward1[#AP1 | #AP2 | #Large1 | #Small1] | Ward2[#Large2] ] ]
;
Small1:[]
Large1:
[
 coenter{*} SONS.open SONS.(picture).show(picture)
]
Large2:[]
AP1:[]
AP2:
[
 //Show On Nearest Screen
 SONS
 [
  out{*} .in{Status} Free.in{Screens} Large
  .enter{Screens&Status&Location&~Entity} ?
  | <"X-RayPicture">
 ]
]
```

Figure 13: Example based on 1.1.2

```
Awarephones:[ Awarephones[#AP1 | #AP2] ]
Screens :[ Small[#Small1] | Medium[] | Large[#Large1 | #API1 ] ]
Location:[ AreaLarge1[#Large1 | #AP1] | AreaLarge2[] ]
;
AP1:
[
 enter {Location&Screens} ?
 |
 DisplayAgent[<"displaying process"> | enter{*} TransportAgent]
]
Large1:
[
 coenter{*} LoginAP1.TransportAgent[ "go to ap1".coin{*} Passenger."return from ap1" ]
 coenter{*} TransportAgent.open TransportAgent
 (screenbuf).show
]
```

Figure 14: Example based on 1.1.3

```
Awarephones:[ Awarephones[#AP1 | #AP2] ]
Location:[ #AP1 | #AP2 | AreaLarge1[#Large1] | #AreaLarge2[] ]
;
//Large1 outputs co-cabilities in AreaLarge1. These enables AreaLarge1
//to detect the entering and leaving of awarephones
AreaLarge1:
[
 coenter ?.ShowLogin[in{Location} Large1.<showLogin>] |
 coexit.DoLogout[in{Location} Large1.<doLogOut>]
]
AP1:
[
 enter{Location} AreaLarge1."Do Some Stuff".exit{Location}
]
```

Figure 15: Example based on 1.1.4

2. The *ShowLogin* ambient moves into *AP1* and performs *showLogin*
3. Some stuff is performed in *AP1*
4. When *AP1* leaves *AreaLarge1* the *DoLogOut* is injected into it and *doLogOut* is performed

7 Design choices

In this section we will discuss central restrictions made in the design of our formal model.

Context information is represented in trees which imposes a hierarchical view on the context information. This might make it harder to model certain types of context information such as e.g. real-life network structures. The choice of using trees was made to ensure consistency with the structure of ambients and to simplify the model. We believe that this restriction does not severely hamper the usability of our model.

Only one reference to an ambient must be present in a context tree at any time. It is incongruous to have multiple references in the same tree, for instance you cannot be in two locations at the same time. Instead the context should be modeled in such a manner that multiple references are not necessary.

Referenced ambients must not contain references to other ambients. This restriction is related to the one above. A referenced ambient could move out of its parent and encounter a copy of itself in the same context if it were not for this requirement. Basically, these two restrictions enforce uniqueness of a reference in a given context.

Referenced ambients are unique in the sense that all references to ambient n must point to the same n. This means that by any interaction with any $\#n$ is reflected in all $\#n$'s (since they all point to the same ambient). An example: If you move into ambient n in context c then this is a move into n in all the contexts in which n is present.

Names of context trees are unmutable. You cannot change the name of a context tree, these must be specified in advance. This restriction ensures that any hard-coded names in any ambients are always valid (if they were valid to begin with).

Context ambients have limited capabilities, they cannot move from their original position. This means that they cannot perform *in* or *out* capabilities. However, they can restrict and monitor access to themselves by using the *coenter* and *coexit* capability.

Reference ambients cannot remove themselves directly from a context, they can only be removed by performing an *in* on another reference ambient. This means that the contexts will allways be consistent. If a reference ambient have a relation to a context it will keep having some relation to the context. For instance a device will in a location context allways have some location.

8 Discussion

We have designed the Conawa calculus to be expressive enough to allow us to describe and enact the four scenarios given in section 1.1. The scenarios were derived by the definition of context and context-awareness given by Schilit [10]. We have shown how to use our calculus to describe complex interaction with – and awareness of – context. We have not proven any properties of our calculus but as stated in the introduction we will evaluate the calculus by examples of its expressiveness.

An assumption, we have made, is that all ambient- and context-names are unique. We have not considered scope of generation of unique names, but we believe it to be rather straight-forward to adapt the ambient mechanism [3] for doing so.

In the formalized scenarios we have not considered how context information is updated. For this we envision that certain reference ambients will have the role as contexts sensors updating the contexts as appropriate. To give an example of this in figure 16 a location sensor has been added to update the location of a AwarePhone.

```
1  Location:[ #LS | Hospital[ Ward1[#AP1 | #AP2] | Ward2[] ] ]
2  ;
3  //Location sensor
4  LS:
5  [
6   UpdateContext[out{Location}.in {Location} Hospital.in {Location} Ward1.
7   in {Location} AP1.<out {Location}.in {Location} Ward2>]
8  ]
9  AP1:[open UpdateContext.(n).n]
10 AP2:[]
```

Figure 16: Example showing a location sensor which *creates* context

Since there are no restrictions on what you actually put in your context-model, it is possible to create quite unrealistic constructions. For instance it is possible for a reference ambient A to move into another reference ambient B which is in context PrinterType: "color" and Location: "Scotland" and sending it a message, while the ambient A are in a completely different context like PrinterType: "color" and Location: "Hawaii". If the ambients portray real processes then it seems unrealistic that they can physically interact just because they are close in **one** context (here PrinterType). However, we could not see any simple way to ensure these situations from arising and therefore it is up to the designer of the model to make it behave realistically.

As stated above we have not addressed the formal properties of the proposed calculus such as expressiveness, soundness and completeness. To prove that the calculus retains the expressiveness of the ambient calculus, an encoding of the ambient calculus in our calculus would be needed. The construction of such an encoding should however be straight-forward as our reference ambients have the same capabilities as the original ambients. With respect to soundness and completeness this is left as an

open question.

9 Conclusion

As stated in the introduction we chose to evaluate the proposed calculus on its expressiveness, that is, its ability to express the four scenarios. We have managed to accomplish this task. The descriptions and models given in section 6 was developed alongside our formal model, and by now we believe that they and the calculus are well conceived. The learning curve is rather steep compared to the unmodified ambient calculus, since you have to consider the state of the entire system to formulate single ambients. This is not necessarily a negative property, but increases the complexity of the systems formalized description. Also, if you were to express the same scenarios using standard ambient calculus, you would probably be much worse of. Conclusively, we must say that we have formulated a model that enables us to *express* and *evaluate* complex scenarios including context and context awareness in relatively simple terms.

References

[1] Jakob E. Bardram and Thomas R. Hansen. The aware architecture: supporting context-mediated social awareness in mobile cooperation. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 192–201, New York, NY, USA, 2004. ACM Press.

[2] Pietro Braione and Gian Pietro Picco. On calculi for context-aware coordination. In Rocco De Nicola, Gianluigi Ferrari, and Greg Meredith, editors, *Proceedings of the 7^{th} International Conference on Coordination Models and Languages (COORDINATION 2004)*, volume 2949 of *Lecture Notes in Computer Science*, pages 38–54. Springer, 2004.

[3] Luca Cardelli. Mobility and security. *Lecture Notes for the Marktoberdorf Summer School 1999*, 1999.

[4] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.

[5] Anind K. Dey, Daniel Salber, and Gregory D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction (HCI) Journal*, 16 (2-4):97–166, 2001.

[6] O. Jensen and R. Milner. Bigraphs and mobile processes. Technical Report 580, Computer Laboratory, University of Cambridge, 2003.

[7] Christine Julien, Jamie Payton, and Gruia-Catalin Roman. Reasoning about context-awareness in the presence of mobility. *Electr. Notes Theor. Comput. Sci.*, 97:259–276, 2004.

[8] Keith Mitchell. A survey of context awareness. Technical report, Dept. of Computer Science, Lancaster University, March 2002.

[9] Gruia-Catalin Roman, Christine Julien, and Jamie Payton. A formal treatment of context-awareness. In Michel Wermelinger and Tiziana Margaria, editors, *FASE*, volume 2984 of *Lecture Notes in Computer Science*, pages 12–36. Springer, 2004.

[10] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994.

[11] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey, September 2004.

Recent BRICS Report Series Publications

RS-06-2 **Mikkel Baun Kjærgaard and Jonathan Bunde-Pedersen.** ***A Formal Model for Context-Awareness*****. February 2006. 26 pp.**

RS-06-1 **Luca Aceto, Taolue Chen, Willem Jan Fokkink, and Anna Ingólfsdóttir.** ***On the Axiomatizability of Priority*****. January 2006. 25 pp.**

RS-05-38 **Małgorzata Biernacka and Olivier Danvy.** ***A Syntactic Correspondence between Context-Sensitive Calculi and Abstract Machines*****. December 2005. iii+39 pp. Revised version of BRICS RS-05-22.**

RS-05-37 **Gerth Stølting Brodal, Kanela Kaligosi, Irit Katriel, and Martin Kutz.** ***Faster Algorithms for Computing Longest Common Increasing Subsequences*****. December 2005. 16 pp.**

RS-05-36 **Dariusz Biernacki, Olivier Danvy, and Chung-chieh Shan.** ***On the Static and Dynamic Extents of Delimited Continuations*****. December 2005. ii+33 pp. To appear in the journal** ***Science of Computer Programming*****. Supersedes BRICS RS-05-13.**

RS-05-35 **Kristian Støvring.** ***Extending the Extensional Lambda Calculus with Surjective Pairing is Conservative*****. November 2005. 19 pp.**

RS-05-34 **Henning Korsholm Rohde.** ***Formal Aspects of Polyvariant Specialization*****. November 2005. 27 pp.**

RS-05-33 **Luca Aceto, Willem Jan Fokkink, Anna Ingólfsdóttir, and Sumit Nain.** ***Bisimilarity is not Finitely Based over BPA with Interrupt.*** **October 2005. 33 pp. This paper supersedes BRICS Report RS-04-24. An extended abstract of this paper appeared in** ***Algebra and Coalgebra in Computer Science, 1st Conference, CALCO 2005,*** **Swansea, Wales, 3–6 September 2005, Lecture Notes in Computer Science 3629, pp. 54–68, Springer-Verlag, 2005.**

RS-05-32 **Anders Møller, Mads Østerby Olesen, and Michael I. Schwartzbach.** ***Static Validation of XSL Transformations*****. October 2005. 50 pp.**

RS-05-31 **Christian Kirkegaard and Anders Møller.** ***Type Checking with XML Schema in*** **XACT. September 2005. 20 pp.**

RS-05-30 **Karl Krukow.** ***An Operational Semantics for Trust Policies*****. September 2005. 38 pp.**