

# On Context Awareness in Global Ubiquitous Computing

Doina Bucur

BRICS\*

Department of Computer Science<sup>†</sup>

University of Aarhus

February 1, 2007

## Abstract

Our work covers two different aspects of ubiquitous computing; it is, in part, a sensor network service discovery protocol tailored for a special application framework, in which the workings of the networking layer (routing and service discovery, embedded into one protocol) can be optimized by allowing knowledge of user context even at the networking layer. Furthermore, a context-awareness process calculus builds upon mobile ambients and introduces a modelling of computing context, distributed through the network and expressed by macros; key features include the expressing of contextual commands and context-triggered actions, and a mechanism for extending the visibility of context in the network through explicit context provision and discovery.

**Keywords** Ubiquitous computing, context awareness, context, sensor networks, service discovery, ad hoc routing, process calculi, mobile ambients, macros, contextual commands, context-triggered actions, process behaviour.

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Sensor Networks in Pervasive Systems</b>	<b>6</b>
2.1	ABSN Requirements . . . . .	6
2.2	Protocol Design . . . . .	8
2.3	Discoverability Analysis . . . . .	12
2.4	Evaluation . . . . .	15
<b>3</b>	<b>A Context-Awareness Calculus</b>	<b>19</b>
3.1	Syntax . . . . .	19
3.2	Semantics . . . . .	22
3.3	Expressiveness . . . . .	24
3.4	Encoding asynchronous $\pi$ -calculus . . . . .	28
<b>4</b>	<b>Conclusions and Future Work</b>	<b>30</b>

---

\*Basic Research in Computer Science, [www.brics.dk](http://www.brics.dk)

<sup>†</sup>[www.daimi.au.dk](http://www.daimi.au.dk), IT-Parken, Åbogade 34, DK-8200, Århus N, Denmark



## 1 Introduction

Computer system designs have followed a development path closely matching that of nature itself: from the early sequential computer model of von Neumann (whose behaviour can easily be compared to that of a single-celled organism) to the concurrent, multiprocess designs initiated by operating systems (resembling multi-celled life forms, one step up the evolution chain) and finally to the various scales of networked computing systems, composed of independent, yet highly mobile and interacting computing entities (just like entire ecologies), computer science concepts have also followed with a matching evolution. If Turing's logic of sequential computability still serves as a basis for the computer science system of thought, it is more recently matched by a logic of interaction, among and within computing entities (as argued in Milner's [21]).

The vision of *ubiquitous computing* (Weiser's [28, 29]) calls for silicon-based information technology to become part of the human environment, the same way as writing - the primary means of information technology - now ubiquitous, frees the definition of knowledge from the limits of human memory. Similarly, noting that nowadays computation takes place in a world of shared situations and heterogeneous technologies, and that the desktop computer is isolated and unaware of its environment, ubiquitous computing aims at freeing the user's focus from isolated computers and towards the user's actual goal, by moving the computers (in various new shapes and sizes, as dictated by their tasks) into the environment, as ubiquitous information conveyers. In the same spirit as that of tagging every object of interest in the surroundings with written symbols, it provides to physical entities of interest - humans included - matching computing entities; the power then lies in networking these entities together for a distributed solving of the task at hand, in a combined aim both at having a virtual smart space closely matching the actual physical space and at providing access to computation in all settings. This makes ubiquitous computing<sup>1</sup> a computing paradigm of extreme interaction.

Concept-wise, ubiquitous computing transcends traditional computing, in that its processes describe systems wider than single computing machines and offers to the human presence - included in the system as yet another entity - an inside view of the system's behaviour; ubiquitous software spans ecologies of computing entities, all of them independent, of a wide range of resources and capabilities, and highly mobile. This active behaviour, combined with the countless number of autonomous entities in such systems, leads to each entity's neighbouring environment (or more generally, context) being highly complex and unpredictable, as is the case in natural environments. Hence, the design of the software performing interaction with entities in such contexts will have interaction and dealing with nondeterminism as elementary building atoms.

### Computer Science Issues in Ubiquitous Computing. Context Awareness

In order to build, deploy and understand the variety of devices in the variety of settings required by ubiquitous computing, the computer science of ubicomp readdresses classical issues, such as hardware low-power, sensing platforms, human-to-computer data input methods, routing and privacy, and brings forward incipient topics of research, such as context awareness. A junction is needed among fields of computer science not traditionally linked together: theories of network protocols, process calculi and statistics deal with their own aspects from within ubiquitous design. Design and theory, as different schools of thought upon ubiquitous computing research, collaborate on building experimental applications or middleware, or work separately on issues such as ad hoc routing,

---

<sup>1</sup>Although their dictionary meanings show some fine differences, the terms *ubiquitous computing*, *pervasive computing* and *global ubiquitous computing* are used in practice to denote this same computing vision and its prototype applications, even if from variate points of view.

modelling locality, trust or mobility, and verification of systems (as noted in Milner's [20]).

The ubiquitous computing paradigm encourages a constantly changing execution environment for its computing entities. In such a setting, *context awareness* is a computing paradigm in which announcements and discovery schemes make applications aware of the changes taking place in their computing context, allowing them to gain advantage from context change, instead of employing a middleware layer for hiding the changes from the application. In the following, we give a brief survey (on the coordinates of Chen and Schilit [9, 24]) of the relevant definitions of computing context, of the mechanics of context from acquisition to use, and of the features which context-aware applications engage. We thus establish a dictionary of terms to point back to from the next sections.

Computing *context* is any piece of information that is used in order to infer knowledge about other computing entities (objects, persons or places) which are interesting to the user application. *Primary contexts* – such as a user's location, the set of resources in a user's neighbourhood, the current network overhead or the temperature – can be divided into (at times, overlapping) categories:

- *resources*<sup>2</sup>, such as neighbouring printers, displays or colleagues, together with lower-level contexts such as the status of the network connections;
- *user context*, meaning an object or person's location, or a person's status;
- *physical context*, such as temperature, moment in time, or sound intensity;
- *history of context*, meaning the recording of any primary context over time.

More complex conclusions upon the status of entities can be drawn by combining several pieces of primary context: when a source of information is indexed by one type of context, after which the result gets indexed by another, the result gets defined as *composite context*; an example is the deduction of a person's current activity by first indexing his calendar with the current time to find his current meeting place, for then to check the status of the projector in the meeting room to find if his presentation is ongoing.

In order for applications throughout the network to be able to access context values under continuous change in network conditions, a number of discrete steps are taken:

- *context acquisition* is the sensing or fusion of context from the environment, and is the only software layer to stay in touch with and follow the state of the physical world;
- *context provision* is the dissemination of context from the sensing or fusion entity to a (local) application or network neighbourhood, a step which enlarges the visibility scope of resources; *context discovery* (primarily known in literature as *service discovery* or *resource discovery*) is then the interested applications locating the provided context and it is the precondition for actual *context use*.

Resource provision and discovery are especially difficult tasks under dynamic network conditions, and designing discovery protocols in such environments means balancing the desire to have resources published globally in the network with the ability - for performance reasons - to only have a local provision scheme.

Depending on the physical properties of the primary contexts, sensing the context is an issue in itself, and its result is prone to nondeterminism. Sensing the location in an indoor network of wireless entities, for example, uses in many cases wireless signal

---

<sup>2</sup>The terms *resource*, *service* and *capability* are hence used interchangeably to denote a computational resource.

strength as a measure of the distance between peers; due to both reflection of the signal on the walls and the inherent lack of sensitivity of the signal strength to distance for many frequencies, the result is uncertain. Pre-trained Bayesian networks, Markov chain techniques or sensor fusion (aggregating input data from more than one sensor sets, of different capabilities) are employed to improve the accuracy of sensing.

Furthermore, the way in which an application makes use of the surrounding context is categorized (as in Schilit's [24]) as follows:

- *contextual information and commands* are the application's requests for either data or actions, which produce different results depending upon the specific context in which they are issued;
- *context-triggered actions* are commands guarded by condition clauses; the clauses make certain inferences about the current context, and depending upon their result, one or another command is issued;
- *automatic contextual reconfiguration* is the process of adding or removing components or links between them, as a result of changing context.

All these categories fall under *active* context awareness, meaning the automatic adapting of the application to the updated surroundings; active context awareness is a far more challenging topic than passive awareness, which relies on the user to take decisions based on the present values of context.

**Overview of report** The research in the remaining of this report has both an operational and a computational nature. Section 2 presents work on a sensor network service discovery protocol tailored for a special application framework, in which the workings of the networking layer (comprising routing and service discovery, in our case embedded into one protocol) can be optimized by allowing knowledge of user context (more precisely, a sensor's work activity identifier) even at the networking layer. The research is a slightly unusual mix of disciplines, with ad hoc routing protocols, service discovery concepts, some pervasive healthcare as motivation, and actual sensor platforms, all playing a role; because of this, some previous knowledge in at least one of these disciplines would be helpful.

Section 3, on the other hand, presents a work-in-progress on a context-awareness process calculus which models primary context, context provision and discovery as primitives, and is able to express the basic features of context-aware applications. Prerequisites for this section include process calculi.

## 2 Sensor Networks in Pervasive Systems

This section presents work on a service discovery protocol for sensor networks, published as [6]<sup>3</sup>. It is motivated by the fact that wireless ad hoc sensor networks are playing an increasing role within ubiquitous computing deployments, yet limited research has been done within the field of resource discovery (and networking frameworks in general) in such networks, and most of this previous research either builds on service discovery protocols from a heavyweight IP-based network infrastructure, or is purely theoretical<sup>4</sup> and detached from the actual application domain.

The protocol is part of a wider project called *Activity-Based Sensor Networks* (ABSN), which aims at building software for pervasive sensor networks (with an immediate view towards deploying the network in an ongoing pervasive healthcare application<sup>5</sup>). It is grounded in the reality of pervasive applications by having the concept of *activity-based computing, ABC* [1] as a basis; the core idea in activity-based computing is that computing systems should support applications by organizing computational services and data in logical bundles that match the users' work *activities* (computational activities which logically bind together different sources of information in order to solve a complex task).

Having to build a service discovery protocol for an application domain defined by ABC and for sensor networks, ABSN blends together research from ad hoc routing, service discovery and studies upon the structure of work activities, in order to give a complete network-and-discovery framework for this particular domain of pervasive applications (and states it is the binding of low-level protocols to high-level human activity concepts that makes sensor networks protocols more applicable than generic routing or service discovery protocols). It also has specific the fact that it agrees that the usual separation of software in layers stacked upon eachother doesn't fit the case of sensor networks, in the case of which designing separately for a routing and a discovery protocol is redundant and thus not optimal with regard to power-efficiency (as argued by [27, 18]).

The rest of this section is organized as follows: subsection 2.1 gives an introduction to the protocol design, including some related work on service discovery protocols for ad hoc and transient networks, subsection 2.2 gives a detailed description of ABSN, while subsection 2.3 makes an analysis of the performance parameters of discovery in ABSN. At last, subsection 2.4 gives the evaluation results and concludes. Practically, the rest of the section is an abbreviated version, slightly rewritten for readability in the wider context of this report, of [6]; for the complete related work, in particular, one should refer to the paper.

### 2.1 ABSN Requirements

An ABSN is an ad hoc sensor network network that has no reliance on any infrastructure; each sensor in the deployment possibly holds a set of services, each service being one sensing capability; in addition, gateways (and other such network points of presence) are considered as simple services, and are treated in the same fashion, for a design that allows the sensor network to function both autonomously (as in the case of its deployment on patients in an emergency setting, outside the reach of infrastructure) and while interfaced to an infrastructure (as in the case of the above-mentioned patients being brought to hospitals).

---

<sup>3</sup>The paper is also accepted, pending revisions, to ACM Springer Journal Mobile Networks and Applications (MONET), for a special issue on Pervasive Healthcare.

<sup>4</sup>Take for example the wide field of routing schemes for ad hoc networks, which produces simulated research rarely applied in practice.

<sup>5</sup>A project at the Centre for Pervasive Healthcare, [www.cfph.dk](http://www.cfph.dk), in the Department of Computer Science, University of Aarhus.

The network topology that ABSN expects to cover is the typical case encountered in home, organization and accident settings:

- The sensors are often deployed in dense, relatively localized and connected patches following a person's or object's location.
- There is a logical structuring of the sensors based on the activity they are a part of; we call this logical grouping an *activity cluster*, AC; a typical activity cluster is formed by the set of body sensors on a patient or by the sensors monitoring the number of people present in a ward room.
- Interaction among sensors is, more often than not, bounded inside an activity cluster, but network-wide discovery and data exchange is of no less importance.
- There is a high degree of mobility involving entire activity clusters at a time and sensor unavailability is often a problem (in the example above, sensors might fall off patients while the patients are being moved, and the protocol is required to signal the change to the application layer).

ABSN is a combined low-level data routing and service discovery protocol, providing the way to most efficiently - in terms of response time, network bandwidth and power consumption - route data in the network and to discover and access services within the network. The ABSN discovery protocol allows low-latency data aggregation within the bounds of a sensor bundle, by locally caching on sensors the neighbouring of same-activity routes and services. For the same purpose, it uses the knowledge about the sensor bundles being relatively localized, in order to limit the packet broadcast overhead in the network. This is different from the majority of the ad hoc protocols in the literature - which do not differentiate between nodes in the network - in the fact that ABSN allows the application-level logic to be used at the networking layer, in order to make the network protocols more efficient in practice.

ABSN designs a completely distributed, hybrid discovery protocol which is proactive in a neighbourhood zone and reactive outside, tailored so that any query among the sensors of one activity is routed through the network with minimum overhead, guided by the bounds of that activity. ABSN enhances the generic *Extended Zone Routing Protocol* with logical sensor grouping and greatly lowers network overhead during the process of discovery, while keeping discovery latency close to optimal.

### 2.1.1 Service Discovery in Ad Hoc Environments

In general, for any networked environment, device mobility implies losing connectivity with well-known environments. *Service discovery* software then enables a mobile user to take advantage of resources at any new location. For ubiquitous environments, in which autonomous computing entities interact, service discovery protocols permit the adaptation of devices to network composition and context.

“Classical” service discovery protocols for pervasive environments - like Jini [19], UPnP [11], SLP [10] and others - are tailored for resource-rich, stable-network enterprise-like environments, and as such are not always applicable in pervasive computing, yet they do provide basic protocol design reference points. They rely on stable, LAN-like network connections, do not need to tackle routing issues (since they use IP), are usually centralized - using one or more directories spanning the network - and can employ sophisticated service semantics.

Unlike LAN devices, sensors form unstable ad hoc network connections and are used for extremely mobile applications. Such *Mobile Ad hoc Networks* (MANETs) are autonomous systems of intermittent, energy-bounded nodes collaborating in the absence of

any centralized support. The network diameter is large compared to a node's range, any data transport is multihop, and the nodes do not have a priori knowledge of the topology of the network. Because of the network diameter-to-wireless range ratio, routing and power efficiency are central issues, and each node has to act as a router.

Thus, when trying to port "classical" service discovery protocols to MANETs, the problems arising are that in MANETs mobility and resource limitations disallow static directories, the underlying network is not stable enough to allow centralized, registration-oriented protocols and the protocols cannot be heavyweight with respect to bandwidth and power usage.

When designing both routing and discovery protocols for large-scale ad hoc sensor networks, important decisions to take include the discovery scheme (be it proactive, reactive or a combination of these<sup>6</sup>), the network range of the protocol (multihop or singlehop), and the power-saving schemes.

## 2.2 Protocol Design

This section gives a detailed description of both Zone Routing Protocol (ZRP, [13]) and Extended Zone Routing Protocol (EZRP, [27]) serving as basis for ABSN.

### 2.2.1 ZRP and EZRP

ZRP is a MANET routing protocol whose guidelines fit a sensor network's setting by being flat, fully distributed and only limitedly proactive; it chooses a hybrid proactive and reactive approach and delimits a zone of a certain number of hops around each node (so that, overall, the zones are heavily overlapped), and limits the proactive procedure to this zone. For out-of-zone discovery, instead of plainly broadcasting of the query throughout the network, the *bordercast* message distribution scheme directs queries from a source node towards the edges of the network by only forwarding the query to the nodes on the border of the source node's zone. It is important to note that, despite grouping the nodes into zones, ZRP is not a hierarchical protocol, but remains a flat one, since each node has a zone, so that the grain size of this zoning is one node.

The ZRP and its subprotocols' IETF drafts do not impose specific protocols for the proactive and reactive discovery of routes, and they even give guidelines specific to routing over the heavyweight IP layer. We choose to adapt these drafts' guidelines for use in resource-poor sensor networks, as described below.

*EZRP* [27] recognizes that, in an ad hoc network, service availability is tightly linked to route determination to that service: finding the address of a service is redundantly followed by finding a route to that address. Because of this, the solution employs the piggybacking of service information in routing packets (an idea first found in [18]). It extends ZRP [13], to include service information, simply by adding a service ID to the hello messages. This way, neighbouring nodes find the others' service IDs, together with their simple presence. Furthermore, nodes periodically broadcast their set of neighbours and their service IDs throughout their zones, so that intra-zone routes are extended with service information.

A view over an EZRP network is given in figure 1. We will denote the ZRP zones by the term *network cluster*, NC, for easily relating them to the ACs. In figure 1,

---

<sup>6</sup>A routing or discovery protocol can be categorized as *proactive* or *reactive*: a proactive protocol makes all the required efforts to gain knowledge over routes before any queries are attempted, while a reactive protocol only searches for a route when a query asks for it. The advantage of proactive route discovery schemes is that routes within the network are continuously refreshed, so that a query for a route is answered without latency; they also constantly add overhead on network bandwidth and on a node's cache memory. Reactive protocols require a global flood search procedure, with long delays and heavyweight traffic at each query.

if a link-state<sup>7</sup> protocol is used intra-NC (for example, a simplified OSPF<sup>8</sup>), node A will receive hello packets from its direct neighbours (also containing a service ID field) and will periodically transmit advertisements throughout the NC, announcing its list of neighbours and their services. When node A's NC converges (all nodes have a consistent view upon the network, from a routing point of view), A will have in its local cache a complete view over its NC, route- and service-wise. In order for A to discover services and routes for nodes D or X, ZRP-style bordercast is employed.

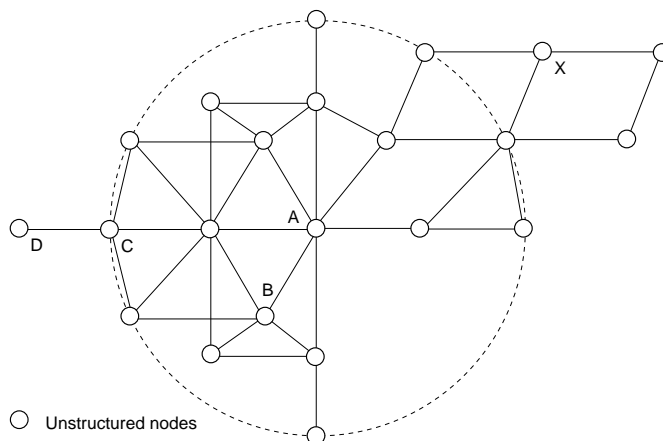


Figure 1: A view over an EZRP discovery network: the nodes are not logically grouped and each node keeps a zone (network cluster, NC) of a 2-hop radius (for this example) centered at itself. The NC for node A is drawn, together with the network connections between nodes.

### 2.2.2 ABSN Discovery Design

As stated in subsection 2.1, the reflection of high-level activities into the sensor network is the logical grouping of the sensors into *activity clusters*, AC, which are deployed in multihop, irregularly-shaped patches throughout the network. Since data communication is more often than not bound inside an AC, we call for a proactive discovery scheme for intra-AC routes and services and a reactive discovery scheme inter-AC.

Although this might immediately recall EZRP as a solution, there is no possibility of identifying ACs with the network clusters in ZRP in an one-to-one fashion. In ZRP, NCs are sets of nodes reachable within a certain radius (number of hops) from any central node and are flatly distributed across the network (for every node in the network there is a NC, and nodes are logically homogeneous). On the other hand, ACs are unique sets of nodes (for example, in the case of a pervasive healthcare application in which patients are monitored by groups of sensors, there is only one AC with patient Hansen's ID) and are deployed in irregular, possibly overlapping and dense geometric patterns around the network.

A view over a basic activity-based network is given in figure 2. Activity clusters are deployed in a relatively localized, connected fashion throughout the network; they might overlap in the same area, so that any node might have in its range other nodes belonging to activities different than its own.

<sup>7</sup>A link-state routing protocol is a protocol in which each node in the network has a full view upon the entire network as a graph, in terms of the identity of the other nodes and the metric of every link between them. A route to a distant node would just be a Dijkstra's shortest path in this graph.

<sup>8</sup>*Open Shortest Path First* (OSPF) is a link-state routing protocol, the most widely used in large networks.

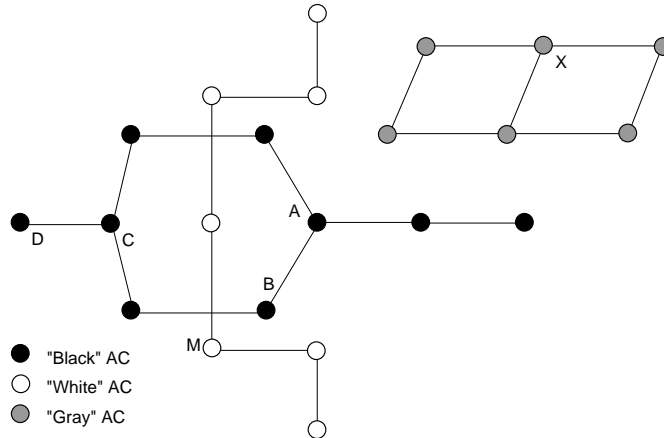


Figure 2: The logical view over an activity-based network: each activity is colour-coded and the intra-AC network connections are drawn; network connectivity also exists among neighbouring nodes of the different, overlapping ACs (and are not drawn in the figure).

The main requirements imposed over the ABSN design are protocol scalability in large networks and low latency at intra-AC discovery: service and route discovery is expected to work throughout the network, yet optimized in such a way that discovery latency is low if the requestor and the service belong to the same activity.

ABSN superimposes the logical AC grouping over the flat EZRP network topology, as in figure 3. To achieve scalability, every node in the network still keeps a NC zone centered at itself, in order to limit the proactiveness of the protocol. Since the nodes within this NC belong to different ACs, and since the logical grouping of sensors in ACs implies a lower probability that a service be requested from a node of a different colour, ABSN has a node only cache the service information of same-AC nodes that lie within this node's NC. Furthermore, a node will proactively cache routing information for all the nodes within its NC.

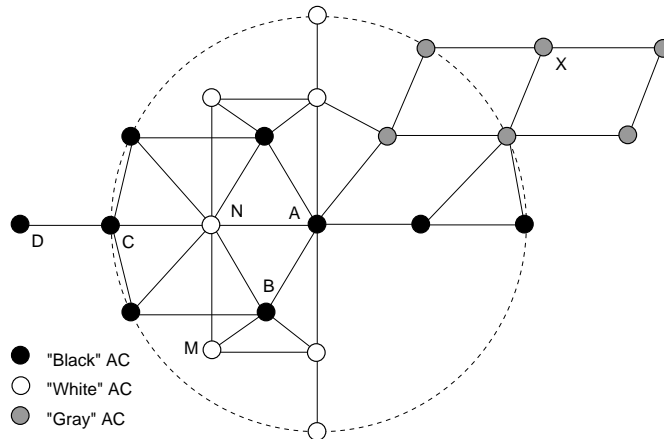


Figure 3: A view over a sample ABSN network: in the EZRP fashion, every node, regardless of its AC, keeps a zone (network cluster, NC) of a 2-hop radius (for this example) centered at itself. The NC for node A is drawn, and it gathers nodes from different ACs, including node A's own AC.

### 2.2.3 Intra-NC Discovery

A lightweight link-state proactive protocol is employed intra-NC. This protocol sends periodic one-hop broadcast hello packets from each node, so that - at all times, and with a maximum latency equal to one hello period - every node knows the addresses of its neighbours and the link quality to each of them, which is used as metric. Also, a node that detects a drastic change in the metric to at least one of its direct neighbours (including a neighbour's arrival or disappearance) triggers a limited-range *link-state advertisement* (LSA), announcing the change within the radius of its own NC. LSAs are forwarded in a broadcast manner for only a number of hops equal to the radius of the NC.

This way, every node keeps a *routing map* of the NC as a weighted graph of nodes that can be reached in a number of hops equal to or less than the NC diameter.

In order to allow nodes to build a *service map* of the near AC, both hello and LSA packets carry, besides addresses, AC and metric information, the service IDs of the nodes these packets advertise: they will both state the service ID of the source of the packet, and - in addition - a LSA will list the neighbours which changed state and, if available, their service information.

For example, at a "black" node (see figure 3), only such incoming packets bearing service IDs for other "black" nodes have their service information cached. In figure 3, this leads to the "black" node A having a service table with the service information of all the "black" nodes in its NC: a hello packet originated at node B advertised the service ID of B, and a LSA triggered by node C (and forwarded over one "white" hop) advertised the service ID of C.

To save computation time and memory, no service information is cached about nodes of other colours.

### 2.2.4 Route and Service Query Solving

With a routing map and a service map in place at each node, the solving of a service query depends on the parameters of the query. The needed service might be either of the same or of a different colour than the source node. Given these two cases, if a certain service of the same colour is needed, then

- the local service table is tried; if there is a match, then the service owner's address is returned;
- if there is no match, the query is bordercast only to the border nodes (or, if none exists, the closest nodes to the border) of the same colour, exploiting the fact that ACs are connected, and thus greatly limiting the network overhead of the query.

If, on the other hand, a service of another colour is needed, then

- the local routing table is checked for the closest node of the searched colour, again relying on the fact that same-colour nodes are more likely to be closer to each other; if such a node (which we call a *gateway* for the entire searched AC) is found, the query is relayed to it, and it will proceed as in the same-colour case above; if more than one gateway is found, ABSN chooses the closest gateway to the query source node;
- if no such gateway exists, the query for the gateway is bordercast to all border nodes

A routing query will be solved exactly as in ZRP, regardless of activity IDs. Routes for all the nodes that can be reached within a number of hops equal to the NC diameter are

read from the proactively built network map. Routes for any other nodes are discovered on-demand, through bordercasting.

This design makes discovery either proactive or reactive, depending on whether the unknown is a route or a service, and on the relation between the AC of the requestor and that of the destination:

- Route discovery is always proactive within the limits of a NC, and reactive outside, regardless of the relation between the colours of the nodes involved.
- Service discovery for same-colour nodes sharing a NC is fully proactive; in all other colour and distance conditions, service discovery is always a mix of proactiveness and reactivity: the query is initially forwarded for certain other nodes to solve, but will eventually reach a node that has discovered the required service information in advance.

### 2.3 Discoverability Analysis

This section analyzes the performance of ABSN compared to the relevant related work protocols. It defines and uses parameters such as the *discoverability* of that service, the *optimality* and latency of the solving of a query and the network overhead added by a query to assess ABSN's characteristics. Furthermore, this section analyzes the set of network topologies that ABSN serves better than other protocols.

The ability of ABSN to discover any service (including a route), if present in the network, is hence called the *discoverability* of that service. ABSN guarantees discoverability in all network settings, provided that any activity cluster is connected; this is like EZRP [27] and GSD [8], and unlike CARD [14].

We denote by the *optimality* of a query that query taking the shortest, lowest-latency route in the network. The optimality of ABSN depends on the NC radius and on the network topology of the searched AC. While having low discovery latency in a large number of network and activity topologies, ABSN only has suboptimal latency in few badly-formed activity cluster topologies, as shown below.

As a routing protocol, ABSN is optimal and guarantees route discoverability, in the fashion of ZRP. The ABSN routing and discovery design following logical activity grouping greatly limits the network and computational overhead that a general-purpose EZRP would imply, if deployed in the pervasive environments ABSN is applicable to.

These statements will be substantiated by the discussion over local and global discovery in the remainder of this subsection.

#### 2.3.1 Neighbourhood Discovery

In regard to route discovery within a node's NC, it is to note that the proactiveness of the lightweight link-state protocol that ABSN uses ensures that changes in the network or in the service provision are propagated in a timely fashion, with a maximum latency directly proportional to the NC radius. Since link-state protocols keep a dynamically updated view over the entire NC and are immune to routing loops, optimality and route discoverability at intra-NC route discovery are ensured.

On the other hand, link-state protocols broadcast any LSA throughout the NC. This implies that the radius of each node's NC must be dynamically updated given the node density in the area, in order to keep network and computation overhead down.

In regard to service discovery, any node will announce throughout the NC any change in the states of its direct neighbours by only mentioning known service information: for example, in figure 3, when "black" node C enters the network, "white" node N will not cache C's service information provided by C's hellos. Thus, a LSA originated from N

and announcing the new “black” node will not provide to A the service information that A could use (since A and C are of the same colour). We call the situation in which the forwarding of service information towards a node is stopped by interposing nodes of another colour by the term “colour shadowing”.

However, within a NC, ABSN is resilient to such apparent “colour shadowing”: even if the “black” AC weren’t connected, the new node C will also trigger a LSA, announcing its new link to node N. Since LSAs are forwarded regardless of the colour of their source and contain the complete service information of their source node, node A will receive a first-hand update about the services provided by C from C itself. In general, within an NC, every node will have a complete image of the same-colour services in the neighbourhood. This resilience is even independent of AC connectivity. Furthermore, still in the example in figure 3, if the “black” AC is connected, LSAs with the service information of the new node will redundantly reach node A on all fully “black” paths from C to A. This way, discoverability of same-colour services within an NC is guaranteed and is optimal - this adds to the optimality of the routing of packets intra-NC.

In the case of different-colour service discovery, on the other hand, while it is ensured that a service will be discovered if present (the service discoverability is also fully guaranteed), the optimality of the query depends on the choice of gateway for the searched AC. Yet, in the worst-case scenario in which a gateway is blindly chosen in the exactly opposite direction from the actual service searched, a maximum of 1 bordercast step will have the query solved (as shown in the example in figure 4).

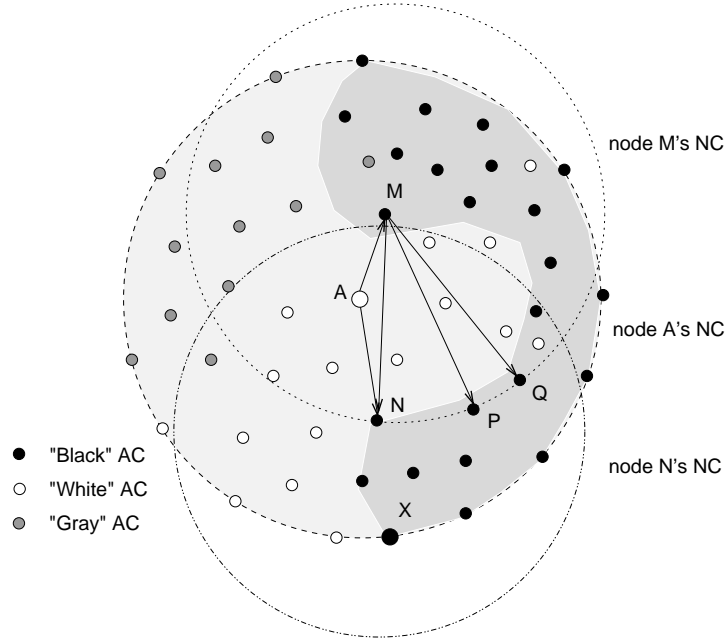


Figure 4: For “white” node A to discover the services of “black” node X, A needs to choose a gateway for the “black” AC. The ABSN solution is to choose the best metric (closest) “black” node as a gateway (in this figure’s case, node M). Yet, for M to discover X it needs to do one bordercast step (forwarding the query to N, P and Q). An optimal choice for a “black” gateway would be node N, which can immediately answer it without latency, since X lies in N’s NC.

### 2.3.2 Global Discovery

At a global level, route discoverability and optimality are secured, by the design of ZRP. An average maximum number of  $\frac{D}{R}$  bordercast steps are employed for the route discovery towards a node which is  $D$  hops away, if the average NC radius of the nodes on the path is  $R$ .

ABSN greatly reduces network overhead by only forwarding service queries for a “black” service within the bounds of the “black” AC. In the case of symmetrical, radially deployed ACs (as in figure 5), the network overhead is reduced to a percentage of  $\frac{\text{black network area}}{\text{total network area}}$  of the network overhead in EZRP (here, we denote by the *area* of an AC or a network a qualitative measure that is proportional to the number of nodes in that AC, the nodes’ degree and to the number of hops this AC occupies).

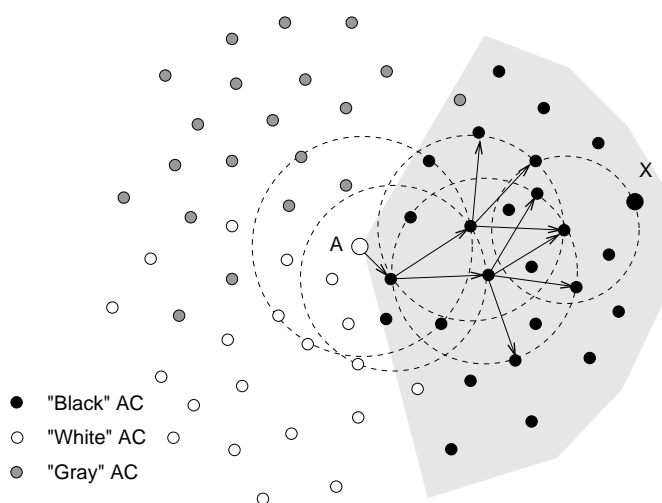


Figure 5: For node A to discover the services of “black” X, a service query will be bordercast repeatedly. Unlike EZRP, which would bordercast the query symmetrically around A to all nodes regardless of colour, ABSN limits the traffic to the bounds of the “black” AC.

Service discoverability in ABSN relies on the activity clusters being connected. If ACs are connected, ABSN guarantees the discoverability and optimality of a different-AC gateway node search, but only the discoverability of a service is guaranteed: the optimality of the path the query takes towards the service depends on the topology of the searched AC: at a global level, “colour shadowing” is in effect, routing a query according to AC topology.

For example, in figure 6 a service query takes a sub-optimal path through the overall network. ABSN argues that this is a small price to pay for the reduction in network traffic. It is to note that this sub-optimality is only true for service requests: service replies are treated as regular data packets and, since routing is optimal, the replies take the optimal path back.

Furthermore, as shown previously, within the limits of a NC, “colour shadowing” does not limit discoverability. This indicates that global “shadows” or disconnections less than the NC radius in width will be overcome by the protocol. In order for the protocol to also cover the general case when ACs are widely disconnected, it only needs to be added a *join* mechanism so that disconnected subsets of one AC can be forwarded data over differently-coloured nodes. Given this, it follows that the preferred AC topologies only exclude disconnections wider than the NC radius.

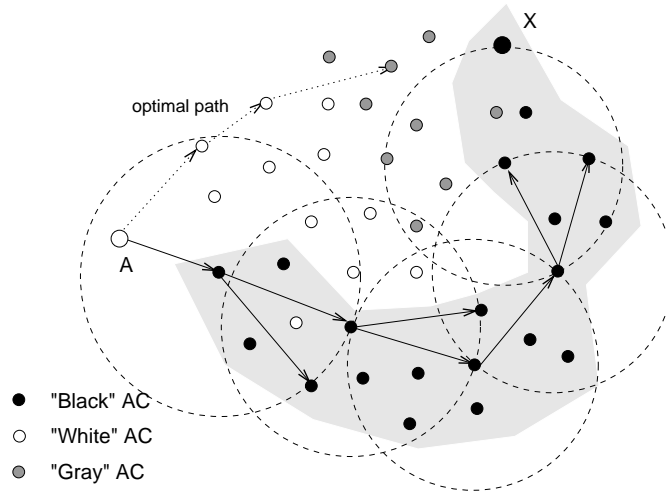


Figure 6: The bordercasting of the “black” query will follow the deployed shape of the “black” AC, instead of taking an optimal path across other ACs.

## 2.4 Evaluation

This section gives an overview of the means of evaluating ABSN. Real-code simulated tests were performed to show the improvements in terms of network and computational overhead of ABSN over EZRP, as well as the scalability of ABSN.

ABSN was implemented on Moteiv’s IEEE 802.15.4-compatible *Tmote skies*, as a set of TinyOS [26] NesC [12] components. NesC and TinyOS (the de facto standard for programming low-resource sensor networks) were used in order to ground ABSN in practice.

For simulation results that would be a correct reflection of the running of the code on sensors, we chose a *real-code* simulator composed of two pieces of software: OMNeT++ [25], a general networking discrete-event simulation environment, and NesCT [17], a language translator from the embedded sensor language NesC into OMNeT++’s C++ classes. NesCT already comes with the translation for OMNeT++ of the basic TinyOS components. Thus, the simulations take into consideration all of TinyOS’s features and limitations.

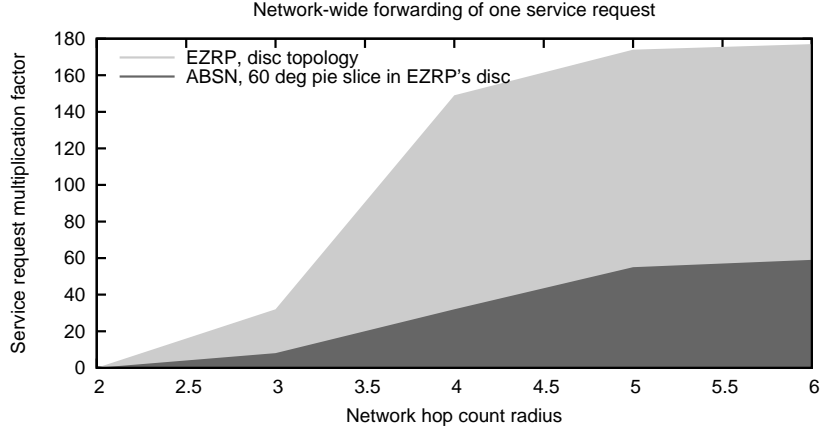
Since ABSN extends EZRP by adapting its basic functionality for use in pervasive activity-based settings, a set of simulations show the decrease in the network overhead triggered by one service query, while being multiplied throughout the network in search of the destination node. Figures 7 (a) and (b) show the network traffic - due to the multiplication of the single service query - as a comparison of two cases in which the service discovery is being performed by EZRP and by ABSN, in the same network. The two sets of tests are performed over extended star networks of varying hopcount radius, composed of nodes with a constant degree equal to 3 and having NCs of constant radius 2. The source of the service query is the central node in the network, the searched service lies on the border of the network, and the two nodes have the same colour.

Two ACs occupy the network: in the case of figure 7 (a), the AC including the source and the destination of the query occupies a non-overlapping 60-degree pie slice in the overall network. The case in figure 7 (b) is extreme, in that which that AC is a chain of nodes running from the central node towards the border destination node.

In figure 7 (a), it can be noted that ABSN uses only a fraction (approximately equal to the pie slice proportion in the 2D network) of the traffic generated by EZRP in order to find the service. Figure 7 (b) is a good example of extreme ABSN efficiency: only a

number of packets that is linear to the distance from source to service is forwarded on the network, compared to EZRP's storm of LSAs.

(a)



(b)

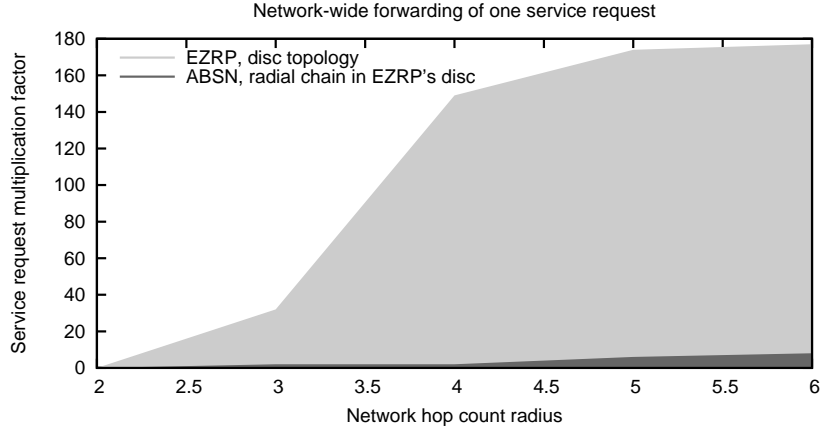


Figure 7: Additional network traffic due to the multiplication of a single service query, in number of packets by the radius of the overall network. The network is a 3-degree extended star with varying radius; all nodes keep NCs of 2-hop radius; the query is intra-AC: it starts at the central node and is destined to a same-colour border node. The searched AC is a non-overlapping, (a) 60-degrees pie slice in the network, (b) chain of nodes from the centre of the star to its periphery.

In the above cases, discovery latency is equal for both EZRP and ABSN, yet ABSN shows a great improvement in network bandwidth usage, by guiding the query along the path of an AC, instead of unconditionally bordercasting it through the network. While these tested topologies do not make a complete usage test, they do give a starting point for extrapolating activity-based network topologies in which ABSN is much more efficient than EZRP.

Furthermore, the tests above only evaluate same-AC queries, and show that intra-AC discovery is much more efficient in terms of use of bandwidth, while only yielding on speed in certain topologies. The more general case of queries originated in one AC and destined to another has a network usage efficiency that is a combination of the performance of EZRP and the efficiency of intra-AC discovery: the discovery of a gateway for the searched AC performs like EZRP, while the discovery of the service starting at the already found gateway performs like the tests above.

One other important test relates to the scalability of the protocol; as stated in subsection 2.2, since ABSN is a flat protocol, the main measure of scalability is the node size of the network clusters. While a large network cluster would improve response times by adding more proactiveness to the overall network, a limit over the NC size is imposed by resource bounds on the nodes and by limited network bandwidth.

Figure 8 shows the network traffic generated by the intra-NC protocol within the bounds of randomly generated, densely connected NCs in extended star topologies. The figure shows both the maintenance traffic (hello packets are sent by each node even in the absence of change) and network building traffic (LSA packets are sent in the process of building the NC, one incoming node at a time). The NC radius is kept constant at 2 hops, while the size of the NC varies to also model the node density. At the same time, the degree of all nodes (the number of links to neighbouring nodes) is kept proportional (by a constant factor) to the NC size. To model a possible real-world case, a number of 3 ACs overlap in this NC, and, at every network change, each node sensing the change triggers, on average,  $\frac{2}{3}$  of a LSA packet (a number given by the relation between the default TinyOS packet size and the size of node addresses and service identification fields).

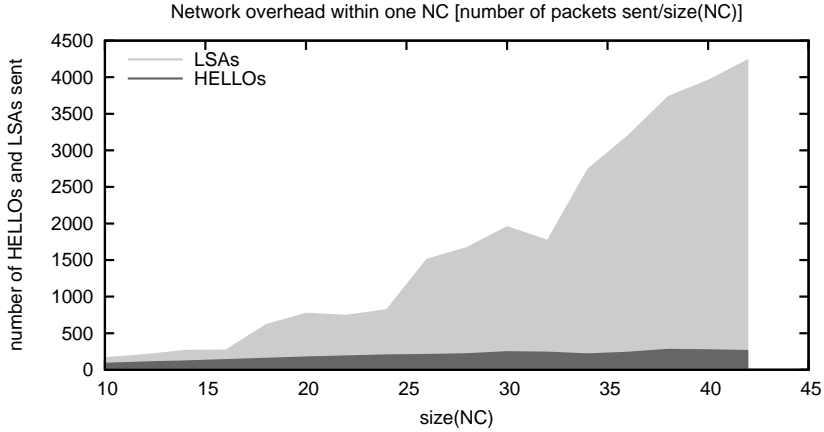


Figure 8: Network traffic, in number of packets by number of nodes composing the NC. The hello packets have been sent out within the bounds of one NC by the link-state proactive intra-NC protocol during a period of time equal to 10 times the hello period. The LSA packets have been sent in the initial process of setting up the same network.

The number of hello packets that are sent on the network only varies slightly below 10 times the size of NC, while the number of LSAs varies significantly with the NC size. This LSA overhead variation is recognizable as being proportional to the square of the NC size, which follows the theory: the forwarding of LSAs (a constant number of such LSAs originate from each node) would trigger their multiplication by a factor proportional to  $N * E$ , where  $N$  is the number of nodes and  $E$  is the number of edges in the network graph (equal to  $\frac{\rho * N}{2}$ , where  $\rho$  is the average degree of the nodes). This gives a multiplication factor proportional to  $N^2$ . Any variation in the test parameters (the number of overlapping ACs and the fraction of one LSA that any network change would trigger) would only change the multiplication factor of forwarded LSAs by a constant.

Another measure of the scalability of the protocol is the size of the data structures needed for each node to act as a router, service provider and discoverer. An overview of the RAM size consumed by these data structures on a TinyOS sensor node is given in figure 9. It is to note in the figure that the adding of service discovery capabilities to nodes running a link-state routing protocol is done at an insignificant memory cost. Thus, such proactive service discovery schemes can be a natural, low-cost extension to

link-state routing protocols in practice. A *Tmote sky* module with 10kB RAM and 48kB programming flash running ABSN will take 17.750 kB of ROM and will accommodate NCs and ACs of as many as 62 nodes, while the sizing down of a NC/AC to a sufficient 50 nodes will occupy 7.077 kB of RAM.

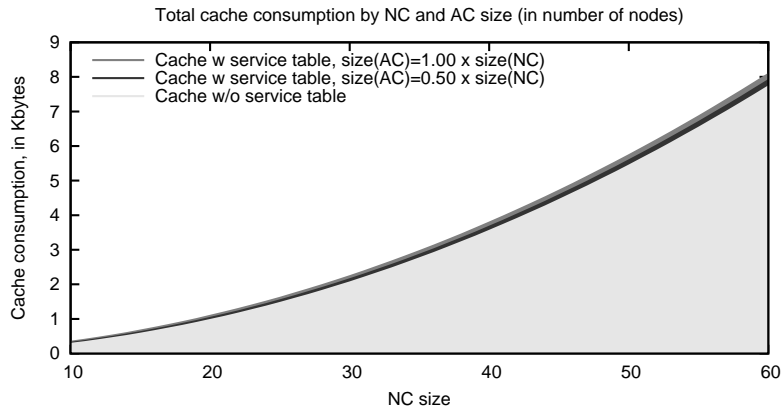


Figure 9: An overview over RAM consumption on a sensor node running ABSN. Memory consumption is static in TinyOS (there is no dynamic allocation), so that total occupied size can be found at build time. The graphic leaves out the footprint of the application (which is constant with the NC size) and only considers the size of all data structures involved in routing and service discovery.

## Conclusion

To summarize our contribution, ABSN is part of an effort to redesign classical network protocols for a better applicability in pervasive computing: it uses the high-level concept of *computational activities* (as logical bundles of data and resources) to give sensors knowledge about their usage even at the network layer and it makes use of this logical structuring of the network for a more effective service discovery scheme. Noting that in practical settings activity-based sensor patches are localized, ABSN designs a completely distributed, hybrid discovery protocol which is proactive in a neighbourhood zone and reactive outside, tailored so that any query among the sensors of one activity is routed through the network with minimum overhead, guided by the bounds of that activity. ABSN enhances the general-purpose *Extended Zone Routing Protocol* (EZRP) with logical sensor grouping and greatly lowers network overhead during the process of discovery, while keeping discovery latency close to optimal. Future work includes generalizing ABSN for use in networks with arbitrarily disconnected activity clusters.

### 3 A Context-Awareness Calculus

This section presents work in progress on a context-awareness calculus, motivated by the need of a computation model for studying the behaviour of computing systems under changing context. *CAC* (*Context-Awareness Calculus*)<sup>9</sup> models primary context and formalizes characteristic properties of context-aware applications, such as the locality of resources, context provision and discovery.

As a model of networks of concurrent processes, our calculus mainly builds upon Cardelli's mobile ambients calculus, [7]; mobile computing entities are modelled by ambients running processes and containing other ambients, and the topology of the network evolves by the ambients' running of *in* and *out* movements. Furthermore, we introduce a modelling of computing context, distributed through the network and expressed by named macros (an idea inspired by Zimmer's context-awareness calculus, [30]), together with the expressing of contextual information or commands and context-triggered actions.

A piece of primary context is a capability modelled by a named macro, which is local to a certain ambient; context might then be made available to a larger set of network entities when being published upwards in the hierarchy of ambients (a scheme which models context provision). The visibility scope of such a published piece of context then comprises all ambients on a downward line in the hierarchy; this way, context is explicitly created and then made available to a chosen region of the network, so that interested computing entities can now react to it (a model of context discovery).

In addition, we state that features of interaction such as generic data communication can be thought of as a kind of context-awareness. Therefore, while such generic features are not primitives in our calculus, we show how they are encoded in *CAC*. Also, we compare the expressivity of our calculus with other concurrency calculi, to find that *CAC* is as expressive as the asynchronous  $\pi$ -calculus, and we offer a qualitative measure of the extent to which *CAC* can model real-world systems and scenarios.

The rest of this section is organized as follows: subsection 3.1 formalizes the syntax of the calculus in terms of processes, gives a brief interpretation of it and discusses the name binding scheme and the substitution mechanism; subsection 3.2 gives the reduction rules for the active processes, while subsection 3.3 gives *CAC* encodings to the context-awareness features of applications and points out a number of encodings for practical scenarios. Subsection 3.4 gives *CAC* encodings to the asynchronous  $\pi$ -calculus, starts the study of the behavioural properties of *CAC* processes and gives pointers for future work.

#### 3.1 Syntax

Processes model the mobile entities, and they evolve through their interactions within themselves or with the environment. The syntax of the processes is formalized in Table 1, in which it is denoted by  $\tilde{x}$  a tuple of zero or more variable names; for simplicity, in the following, the set of the names in  $\tilde{x}$  will also be written as the tuple  $\tilde{x}$ , with the context calling for the correct interpretation.

When comparing the syntax with that of the mobile ambients calculus, it is notable that the *open* mobility primitive has been left out, as inessential for our calculus. Also, so have been the generic data communication primitives: the synchronous input action  $(x).P$  and asynchronous output action  $\langle M \rangle$ ; *CAC* encodes this traditional model of data

<sup>9</sup>This name is a working name, and will almost certainly be rethought before publication.

---

processes	$P ::= 0$ $  P \mid P'$ $  (\nu z)P$ $  f \langle \tilde{v} \rangle$ $  a(D)[P]$ $  def^a E \text{ in } P$ $  in\ a.P$ $  out.P$	no process parallel composition name restriction macro call mobile entity definition for remote entity movement in movement out
definitions	$D ::=$ $  E_1, \dots, E_k$ $E ::= F$ $  !F$ $F ::= f(\tilde{x}) \triangleright P$	no definition set of definitions one-shot definition permanent definition macro definition

---

Table 1: Syntax

communication with a name-based macro call with parameters, a mechanism designed to model computational context, as will be detailed later.

A central concept of the calculus is the usage of *names*: a mobile entity interacts with the context by knowing and calling the names of the capabilities it requires from the environment (which can be thought of in terms of resources or services at remote locations), or by providing new such named capabilities to the environment. An infinitely countable set of names is presupposed, elements of which are written in lower-case letters. Three types of names are employed: *macro names*, which denote capabilities, *variable names* for the macros' arguments, and the regular *ambient names*. All types of names are taken from the same name set mentioned above; the syntax of the calculus will point to one or another name type.

A brief interpretation of the processes in Table 1 is given below.

- $0$ ,  $P \mid P'$ ,  $(\nu z)P$ ,  $in\ a.P$  and  $out.P$  keep their traditional meaning.
- A piece of primary computational context is a macro  $f(\tilde{x}) \triangleright P$ , for example:
  - a *resource* is a public computer's capability to run an editor:  $emacs(file) \triangleright P$  (in which  $P$  is the machine-dependent executable code for emacs), or a meeting room's capability to offer a pointer to the largest display available:  $largest\_display() \triangleright "Board1"$ ;
  - a piece of *user context* is the user's preference towards triggering the running of his favourite editor whenever a suitable screen is in available:  $trigger() \triangleright emacs(file)$ , or the user's status:  $status(id) \triangleright "seated"$ ;
  - a piece of *physical context* is an environmental status:  $weather\_forecast(day) \triangleright http\langle "dmi.dk", day \rangle$ .

Each such piece of context originates at a certain ambient (that ambient which is the provider of the resource, or the entity which acquires a user's status or some physical context); context might then be made public to the network outside the original ambient. A macro is discovered through its name only; we choose to simplify the treatment of macros by embedding the arity of macros in their names, so that a macro name has a fixed number of arguments.

- An ambient is now  $a(D)[P]$ , in which a set of macro definitions  $D$  is carried along. The macros in ambient  $a$ 's set of definitions are explicitly published to  $a$  by the entities which originated them, by running a definition process  $def^a E \text{ in } P$ , which publishes the definition  $E$  to remote ambient  $a$  and continues local execution with process  $P$ . We choose to limit the extent of the network from which ambients publish definitions towards  $a$ , to contain only the ambients in  $P$ , as explained in the semantics of the calculus.
- An inquiry upon the value of a piece of context is a macro call  $f \langle \tilde{v} \rangle$ , a process which - if a macro called  $f$  is available - has the effect of being replaced by the body of the macro, in which the input parameters have been substituted. The replacement takes place only if a suitable macro is publicly available in the environment; similarly, we choose to limit the extent of the network into which a call  $a()[f \langle \tilde{v} \rangle]$  will search for macro definitions to contain only the ancestor ambients of  $a$ .
- Definitions come in two flavours:
  - a one-shot definition  $f(\tilde{x}) \triangleright P$  is a macro definition which is 'consumed' by that macro being called;
  - a permanent definition  $!f(\tilde{x}) \triangleright P$  is one which can be instantiated by any number of macro calls, and is in fact an infinite set of one-shot definitions.

The sets of bound and free names for the processes above are given in Table 2. It is to note that there exist only two name-binding constructions: the usual  $\nu z(P)$ , in which all occurrences of  $z$  within  $P$  become bound, and the definition  $f(\tilde{x}) \triangleright P$ , in which all occurrences of  $\tilde{x}$  in  $P$  become bound. The name of the defined macro,  $f$ , is not a binder for the occurrences of  $f$  within this the scope of this definition.

		bound names, $bn(P)$	free names, $fn(P)$
$P ::=$	$0$	$\emptyset$	$\emptyset$
	$  P \mid P'$	$bn(P) \cup bn(P')$	$fn(P) \cup fn(P')$
	$  \nu z(P)$	$bn(P) \cup \{z\}$	$fn(P) \setminus \{z\}$
	$  f \langle \tilde{v} \rangle$	$\emptyset$	$\{f\} \cup \tilde{v}$
	$  a(D)[P]$	$bn(D) \cup bn(P)$	$\{a\} \cup fn(D) \cup fn(P)$
	$  def^a E \text{ in } P$	$bn(E) \cup bn(P)$	$\{a\} \cup fn(E) \cup fn(P)$
	$  in a.P$	$bn(P)$	$\{a\} \cup fn(P)$
	$  out.P$	$bn(P)$	$fn(P)$
$D ::=$		$\emptyset$	$\emptyset$
	$  E_1, \dots, E_k$	$\bigcup_{i=1..k} bn(E_i)$	$\bigcup_{i=1..k} fn(E_i)$
$E ::=$	$F$	$bn(F)$	$fn(F)$
	$!F$	$bn(F)$	$fn(F)$
$F ::=$	$f(\tilde{x}) \triangleright P$	$\tilde{x} \cup bn(P)$	$\{f\} \cup (fn(P) \setminus \tilde{x})$

Table 2: Bound and free names in processes

A variable substitution in a process,  $\{\tilde{v}/\tilde{x}\} P$  or  $P\sigma$  (we use  $\sigma$  to range over substitutions), denotes the process  $P$  in which the free names  $\tilde{x}$  have been substituted for  $\tilde{v}$ ;

this substitution may involve alpha-conversion to avoid name clashes: if  $bn(P)$  intersects the set of variable names from  $\tilde{v}$  in a non-empty set, then each of the variables in this set will be renamed to a fresh name in  $P$  before the actual substitution.

We adapt the convention: when considering a process and a substitution, we assume that the bound names of the process are different from the free names of the process and from both sets of names in the substitution. We define the process  $P\sigma$  resulted from the application of the substitution  $\sigma$  to  $P$  as in Table 3, in which we denote by  $\tilde{v}\sigma$  the tuple having as elements the elements of the tuple  $\tilde{v}$  upon which the substitution  $\sigma$  was performed.

---

$0\sigma$	$\triangleq$	$0$		
$(P P')\sigma$	$\triangleq$	$P\sigma P'\sigma$		
$(\nu z P)\sigma$	$\triangleq$	$\nu z P\sigma$	$(E_1, \dots, E_k)\sigma$	$\triangleq$ $E_1\sigma, \dots, E_k\sigma$
$(f \langle \tilde{v} \rangle)\sigma$	$\triangleq$	$f\sigma \langle \tilde{v}\sigma \rangle$	$(!F)\sigma$	$\triangleq$ $!F\sigma$
$(a(D)[P])\sigma$	$\triangleq$	$a\sigma(D\sigma)[P\sigma]$	$(f(\tilde{x}) \triangleright P)\sigma$	$\triangleq$ $f\sigma(\tilde{x}) \triangleright P\sigma$
$(def^a E \text{ in } P)\sigma$	$\triangleq$	$def^{a\sigma} E\sigma \text{ in } P\sigma$		
$(\text{in } a.P)\sigma$	$\triangleq$	$\text{in } a\sigma.P\sigma$		
$(\text{out}.P)\sigma$	$\triangleq$	$\text{out}.P\sigma$		

---

Table 3: Substitutions

### 3.2 Semantics

The structural congruence relations allow for manipulation of the internal process structure; the structural congruence of processes is the smallest congruence over the relations in Table 4; more structural congruence relations exist, but they are not essential to our semantics and are thus left out. Only the structural congruence for the list of definitions in an ambient is new:  $D, E, D' \equiv E, D, D'$  states that any single definition  $E$  in a list of definitions can be brought to the first position in the list (and thus, any list of definitions effectively behaves like a set).

---

$P 0$	$\equiv$	$P$	$\nu z 0$	$\equiv$	$0$
$P Q$	$\equiv$	$Q P$	$\nu z \nu w P$	$\equiv$	$\nu w \nu z P$
$(P Q) R$	$\equiv$	$P (Q R)$	$\nu z (P Q)$	$\equiv$	$P \nu z Q$ if $z \notin fn(P)$
			$\nu z (a(D)[P])$	$\equiv$	$a(D)[\nu z P]$ if $z \neq a$ and $x \notin fn(D)$
$P \equiv Q$	$\Rightarrow$	$Q \equiv P$	$P \equiv Q, Q \equiv R$	$\Rightarrow$	$P \equiv R$
$P \equiv Q$	$\Rightarrow$	$\mathbf{A}[P] \equiv \mathbf{A}[Q]$	$D, E, D'$	$\equiv$	$E, D, D'$

---

Table 4: Structural congruence

The one-holed 'active' contexts  $\mathbf{A}$  and their bound and free names are defined in Table 5; as in the case of the bound and free names of a process, only the restriction

operator  $\nu$  and the macro parameters add to the set of bound names of such a context. Note that the hole can only be found in 'active' places, a fact which allows an eventual process placed in the hole to start reactions immediately: a hole can be found neither in the body of a definition, nor guarded by an *in* or *out* action.

Also, we define the set of macro definitions of a context  $\mathbf{A}$ ,  $Def(\mathbf{A})$ , as being the collected set of definitions in the ambients above the hole (the interesting aspect of a definition being its name  $f$ ); we consider  $Def(D)$  as being the set including each unique name  $f$  extracted from the set of definitions  $D$ . On the same line, the set of ambient names in a context  $\mathbf{A}$ ,  $Amb(\mathbf{A})$ , is intuitively defined as the collected names of the ambients above the hole.

contexts	$bn(\mathbf{A})$	$fn(\mathbf{A})$	$Def(\mathbf{A})$
$\mathbf{A} ::= [\cdot]$	$\emptyset$	$\emptyset$	$\emptyset$
$  \mathbf{A} P$	$bn(\mathbf{A})$	$fn(\mathbf{A})$	$Def(\mathbf{A})$
$  (\nu z)\mathbf{A}$	$bn(\mathbf{A}) \cup \{z\}$	$fn(\mathbf{A}) \setminus \{z\}$	$Def(\mathbf{A})$
$  a(D)[\mathbf{A}]$	$bn(\mathbf{A}) \cup bn(D)$	$\{a\} \cup fn(D) \cup fn(\mathbf{A})$	$Def(D) \cup Def(\mathbf{A})$

Table 5: Active contexts

We then formally specify the reduction relations in Table 6, in which the (*Struct*) relation states that manipulation of the internal structure of processes, in accordance with the structural congruence rules, can take place before and after reduction, and the (*Context*) relation states that a process in an 'active' spot of a context can reduce at any moment.

$\frac{}{a(D)[in\ b.P Q] \mid b(D')[R] \longrightarrow b(D')[R \mid a(D)[P Q]]} (In)$	
$\frac{}{a(D)[b(D')[out.P Q] \mid R] \longrightarrow a(D)[R \mid b(D')[P Q]]} (Out)$	
$\frac{a \notin Amb(\mathbf{A})}{a(D)[\mathbf{A}[def^a\ E\ in\ P] \mid Q] \longrightarrow a(D, E)[\mathbf{A}[P] \mid Q]} (Definition)$	
$\frac{f \notin Def(\mathbf{A})}{a(f(\tilde{x}) \triangleright P, D)[\mathbf{A}[f \langle \tilde{v} \rangle]] \longrightarrow a(D)[\mathbf{A}[\{\tilde{v}/\tilde{x}\} P]]} (Reaction)$	
$\frac{f \notin Def(\mathbf{A})}{a(!f(\tilde{x}) \triangleright P, D)[\mathbf{A}[f \langle \tilde{v} \rangle]] \longrightarrow a(!f(\tilde{x}) \triangleright P, D)[\mathbf{A}[\{\tilde{v}/\tilde{x}\} P]]} (!Reaction)$	
$\frac{P \equiv P' \quad P \longrightarrow Q \quad Q \equiv Q'}{P' \longrightarrow Q'} (Struct)$	$\frac{P \longrightarrow Q}{\mathbf{A}[P] \longrightarrow \mathbf{A}[Q]} (Context)$

Table 6: Semantics

The *(In)* and *(Out)* relations formalize movement in the fashion of mobile ambients: only the relative positions of the ambients change. A process calling for the publishing of a macro definition  $E$  to the remote ambient  $a$ ,  $def^a E \text{ in } P$ , adds the definition  $E$  to the set of definitions of the closest ambient named  $a$ , which lies on the ancestor line in the context  $\mathbf{A}$ , for then to continue execution with process  $P$ .

The *(Reaction)* relation replaces a macro call  $f \langle \tilde{v} \rangle$  with the body  $P$  of the macro  $f(\tilde{x}) \triangleright P$ , in which variables  $\tilde{v}$  have substituted  $\tilde{x}$ . The replacing only takes place if there is an ambient on the ambient ancestor line in the context of the call, which holds a suitable definition for  $f(\tilde{x})$ . The only difference between the rules for *(Reaction)* and *(!Reaction)* is that in the first case, the found definition is a one-shot, and is subtracted from its host definition list after reduction; in the second case, the definition is permanent, and so remains for further instantiation.

### 3.3 Expressiveness

A discussion upon the expressivity of CAC, with regard to its basic elements of interaction and in comparison with the  $\pi$ -calculus [23] and mobile ambients [7], is now in order.

In CAC, knowledge of names forms the base of the interaction mechanism. Just as  $\pi$ , but unlike mobile ambients, data communication in CAC relies on the knowledge of names; we state that this reliance is a good match for the real world, in which interaction is based on protocols, and, in order to communicate, devices must be aware of things like protocol ports and frame formats. In our case, names do not point to communication channels, but to macros (in essence, functions of any arity). If  $\pi$  has the basic interaction constructions  $\bar{x}y$  for output and  $x(z)$  for input, CAC has in turn macro calls,  $f \langle \tilde{v} \rangle$ , for output and macro definitions carried along by ambients,  $f(\tilde{x}) \triangleright P$ , for input.

Furthermore, the choice towards explicitly having both input  $f(\tilde{x}) \triangleright P$  and banged input  $!f(\tilde{x}) \triangleright P$  is grounded in practice: intuitively, pieces of context range between being 'consumable' (a unicast network packet can only be received once) and being 'permanent' (the status of a person, once determined, remains constant and can be queried an infinite number of times before being redetermined).

Like mobile ambients and the asynchronous variant of  $\pi$ , but unlike the standard, fully synchronous  $\pi$ , CAC is input-synchronous and output-asynchronous: the CAC macro call  $f \langle \tilde{v} \rangle$  is like the output action  $\langle M \rangle$  in mobile ambients, and unlike the  $\pi$ -style  $\bar{x}y.P$ , while the CAC macro definition  $f(\tilde{x}) \triangleright P$  is a synchronous input. This design has been chosen for minimality: it is known from Boudol [4] and Honda [16] that asynchronous  $\pi$  can encode the full  $\pi$ , and CAC encodes the asynchronous  $\pi$  without difficulty, as discussed below.

The mobile ambients-style data communication for general purposes is eliminated from CAC; instead, CAC focuses on the communication of data only in relation to context provision and usage. Furthermore, we state that general data communication can be seen as a feature of context, and we encode examples of simple, generic network protocols in CAC, in the rest of this section.

Replication of processes in CAC is also not fully-fledged, but only focuses on the context awareness-related constructs; we offer replication to the input process by allowing macro definitions to become permanent; replicated output is also easily encoded using the replicated input: in  $a(!r \langle \rangle \triangleright f \langle \rangle | r \langle \rangle)[r \langle \rangle]$ , the call for  $r$  in the body of the ambient effectively behaves like  $!f \langle \rangle$ . The scheme also works more generally: in  $a(!r \langle \rangle \triangleright P | r \langle \rangle)[r \langle \rangle]$ , the call for  $r$  in the body of the ambient effectively behaves like  $!P$ .

In regard to name scoping, CAC does not add much heavyweight to the mobile ambients, since the names of the macro definitions carried by ambients are not binders. The way a macro call gets substituted varies with the dynamics of the new definitions being published in the network, so that the actual substitution will originate in the definition closest to the call, regardless of the age of the definition.

The topology of a network model in CAC matches that of mobile ambients: computing entities are placed in a tree of ambient inclusions, and ambient movement is guided by *in* and *out* actions. The mechanism of providing and discovering context builds upon the topology of the network. A piece of context only needs to be published to that entity in the network which the interested users can query for that context. In practice, a large number of context-aware applications have their logical topology layed out hierarchically, with computing entities at the top of the hierarchy publishing context of general interest, while entities at the bottom of the hierarchy publish context of local interest. This matches the mobile ambients topology, which leads to our design calling for the publishing of context to ambients upwards in the ambient hierarchy, and for the use of context downwards from these ambients.

Finally, the macro calls in CAC directly encode contextual information (which are, as defined in section 1, calls for context data which get different results in different contexts) and contextual commands (actions which proceed differently, depending on context). Abstracting away, we state that – given the limited expressivity power of mobile ambients, which cannot model links between agents – automatic contextual reconfiguration is essentially the same as context-triggered actions, and we look over the encoding of the latter in the following.

### 3.3.1 Expressing context-triggered actions

Depending on the complexity of the real-world task to model, process calculi do need certain expressive extensions. In the following, we explore the extent of our calculus being able to express context-triggered actions, meaning - as defined in the introduction - condition clauses which make certain inferences about the current context, and depending upon their result, sprout one or another process.

Let a simple scenario be that in which the condition about the current context tests only one variable; in this case, the presence of a user: a screen  $screen()[]$  in a room  $room()[]$  needs to react to the coming in the same room of a person  $tag()[]$  by starting his favourite program (an editor,  $emacs \langle \tilde{v} \rangle$ ). The screen is always willing to trigger programs for incoming people, and can run emacs (the process  $P$  being the executable code for it, and the input variables being start preferences, such as the file to open):

$$screen \triangleq screen(!react () \triangleright trigger \langle \rangle | react \langle \rangle), emacs (\tilde{x}) \triangleright P [react \langle \rangle]$$

behaving, in effect, like  $screen(emacs (\tilde{x}) \triangleright P) [!trigger \langle \rangle]$ , and thus modelling replicated output with only replicated input. The call for  $trigger \langle \rangle$  is fed by tags entering the room; the tags might themselves specify the preferred application to be started on the screen:

$$tag \triangleq tag()[def^{room} trigger () \triangleright emacs \langle \tilde{v} \rangle in 0]$$

and so, after the tag moves in the room, which is now  $room \triangleq room()[screen|tag]$ :

$$\begin{aligned} room &= room()[screen(emacs (\tilde{x}) \triangleright P) [!trigger \langle \rangle] | tag()[def^{room} trigger () \triangleright emacs \langle \tilde{v} \rangle in 0]] \\ &\longrightarrow room(trigger () \triangleright emacs \langle \tilde{v} \rangle) [screen(emacs (\tilde{x}) \triangleright P) [!trigger \langle \rangle] | tag() []] \\ &\longrightarrow room()[screen(emacs (\tilde{x}) \triangleright P) [emacs \langle \tilde{v} \rangle | !trigger \langle \rangle] | tag() []] \\ &\longrightarrow room()[screen()[P \{ \tilde{v} / \tilde{x} \} | !trigger \langle \rangle] | tag() []] \end{aligned}$$

In the same way, the application to be migrated on the screen can be one residing on a server in the network; the tag will then only publish to the room its owner's identification data, so that the screen can call the application from the network based on this data.

In the general case in which more context inputs are checked in the condition clause, for a natural encoding CAC needs practical extensions, such as the match operator on input in the early  $\pi$ -calculus.

### 3.3.2 Scenario<sup>10</sup>: display on largest display available

Let an operating room *OR7* be configured with a (permanent) capability for pointing incoming *display* calls to the largest display locally available in the room:

$$OR7 \triangleq OR7(!display(x, y) \triangleright \nu h [def^x h() \triangleright move \langle y, "Board1" \rangle in h \langle \rangle])[]$$

In *display(x, y)*, *x* is the identity of the caller and *y* is the file to display; *h* is an intermediate handler which sets things in motion, and is made private for safety.

The tag *TagA* enters the operation room and calls for the displaying of a chart; a way to interpret the display call is unknown to the tag before entering *OR7*:

$$TagA \triangleq TagA()[display \langle TagA, chart \rangle]$$

After the move in, *TagA* can execute the displaying of the chart; *OR7* provides the necessary data by moving down a definition to the calling application (hence the need for the tag to send its identity with the call). It is implied that *TagA* knows a definition solving a call for *move(x, y)* into an actual process:

$$\begin{aligned} OR7 & \quad (!display(x, y) \triangleright \nu h [def^x h() \triangleright move \langle y, "Board1" \rangle in h \langle \rangle]) \\ & \quad [TagA()[display \langle TagA, chart \rangle]] \\ \rightarrow OR7 & \quad (!display(x, y) \triangleright \nu h [def^x h() \triangleright move \langle y, "Board1" \rangle in h \langle \rangle]) \\ & \quad [TagA()[\nu h [def^{TagA} h() \triangleright move \langle chart, "Board1" \rangle in h \langle \rangle]]] \\ \rightarrow OR7 & \quad (!display(x, y) \triangleright \nu h [def^x h() \triangleright move \langle y, "Board1" \rangle in h \langle \rangle]) \\ & \quad [(\nu h) TagA(h() \triangleright move \langle chart, "Board1" \rangle)[h \langle \rangle]] \\ \rightarrow OR7 & \quad (!display(x, y) \triangleright \nu h [def^x h() \triangleright move \langle y, "Board1" \rangle in h \langle \rangle]) \\ & \quad [TagA()[move \langle chart, "Board1" \rangle]] \end{aligned}$$

### 3.3.3 Scenario: get remote status

Keep a *TabA* in *OR7* and let now a similar tag, *TagB*, be in a remote operation room, *OR8*, in a hospital covered by a network infrastructure, *Net*:

$$Net \triangleq Net()[OR7()[TagA] \mid OR8()[TagB]]$$

The status of the mobile tags is published by the tags themselves to the infrastructure, so that it can be queried from any location within the hospital. *TagA* publishes its status as a permanent definition:

$$TagA \triangleq TagA()[def^{Net} !stateA(x) \triangleright \nu h [def^x h() \triangleright "operating" in h \langle \rangle] in 0]$$

and the network becomes 'aware' of it:

<sup>10</sup>The following three scenarios are inspired by the local AWARE project, [2].

$$\begin{array}{l}
Net \quad () \\
\quad [OR7()[TagA()][def^{Net} !stateA(x) \triangleright \nu h [def^x h() \triangleright \text{"operating" in } h \langle \rangle] \text{ in } 0]] \\
\quad | OR8()[TagB]] \\
\rightarrow Net \quad (!stateA(x) \triangleright \nu h [def^x h() \triangleright \text{"operating" in } h \langle \rangle]) \\
\quad [OR7()[TagA()]] \\
\quad | OR8()[TagB]]
\end{array}$$

If  $TagB$  needs to use  $TagA$ 's status, it can inquire:

$$TagB \triangleq TagB()[stateA \langle TagB \rangle]$$

and will receive an answer from the network:

$$\begin{array}{l}
Net \quad (!stateA(x) \triangleright \nu h [def^x h() \triangleright \text{"operating" in } h \langle \rangle]) \\
\quad [OR7()[TagA()]] | OR8()[TagB()[stateA \langle TagB \rangle]] \\
\rightarrow Net \quad (!stateA(x) \triangleright \nu h [def^x h() \triangleright \text{"operating" in } h \langle \rangle]) \\
\quad [OR7()[TagA()]] | OR8()[TagB()[\nu h [def^{TagB} h() \triangleright \text{"operating" in } h \langle \rangle]]]] \\
\rightarrow Net \quad (!stateA(x) \triangleright \nu h [def^x h() \triangleright \text{"operating" in } h \langle \rangle]) \\
\quad [OR7()[TagA()]] | OR8()[(\nu h)TagB(h) \triangleright \text{"operating"}][h \langle \rangle]]
\end{array}$$

### 3.3.4 Scenario: mobile code

Keep the two tags in the operating rooms of the hospital covered by the infrastructure:

$$Net \triangleq Net()[OR7()[TagA] | OR8()[TagB]]$$

$TagA$  needs to send a process  $P$  to  $TagB$ . Since they are in remote locations, the code is routed by the network. Note that the context added by the code to the network is not permanent (the only significant difference to the previous scenario).  $TagA$  sends the code out:

$$TagA \triangleq TagA()[def^{Net} codeAtoB() \triangleright \nu h [def^{TagB} h() \triangleright P \text{ in } h \langle \rangle] \text{ in } 0]$$

and the network becomes 'aware' of it:

$$\begin{array}{l}
Net \quad () \\
\quad [OR7()[TagA()][def^{Net} codeAtoB() \triangleright \nu h [def^{TagB} h() \triangleright P \text{ in } h \langle \rangle] \text{ in } 0]] \\
\quad | OR8()[TagB]] \\
\rightarrow Net \quad (codeAtoB() \triangleright \nu h [def^{TagB} h() \triangleright P \text{ in } h \langle \rangle]) \\
\quad [OR7()[TagA()]] | OR8()[TagB]]
\end{array}$$

If  $TagB$  agrees to download  $TagA$ 's code, it inquires:

$$TagB \triangleq TagB()[codeAtoB \langle \rangle]$$

and will receive an answer from the network:

$$\begin{array}{l}
Net \quad (codeAtoB() \triangleright \nu h [def^{TagB} h() \triangleright P \text{ in } h \langle \rangle]) \\
\quad [OR7()[TagA()]] | OR8()[TagB()[codeAtoB \langle \rangle]] \\
\rightarrow Net \quad () \\
\quad [OR7()[TagA()]] | OR8()[TagB()[\nu h [def^{TagB} h() \triangleright P \text{ in } h \langle \rangle]]]] \\
\rightarrow Net \quad () \\
\quad [OR7()[TagA()]] | OR8()[(\nu h)TagB(h) \triangleright P][h \langle \rangle]] \\
\rightarrow Net \quad () \\
\quad [OR7()[TagA()]] | OR8()[TagB()[P]]
\end{array}$$

The downloaded piece of code  $P$  will preserve in  $TagB$  any bound variables that were bound in  $TagA$ , but will solve macro calls with free names according to the new context.

### 3.4 Encoding asynchronous $\pi$ -calculus

Following Zimmer's approach in [30], we encode the monadic (communicating only one name in an action), asynchronous (disallowing outputs with continuations), replicated input (of all the processes, only the input process is replicated)  $\pi$ -calculus, formalized in Table 7. It is known that this restricted asynchronous  $\pi$  is as expressive as  $\pi$  (from Boudol [4] and Honda [16]). As in Zimmer's case, a  $\pi$ -calculus system is a  $world()$  ambient in our case, only one ambient is needed for the encoding, and the topology of our calculus is not used.

---

$\llbracket 0 \rrbracket$	$\triangleq$	$0$
$\llbracket P R \rrbracket$	$\triangleq$	$\llbracket P \rrbracket \llbracket R \rrbracket$
$\llbracket (\nu z)P \rrbracket$	$\triangleq$	$(\nu z)\llbracket P \rrbracket$
$\llbracket f \langle v \rangle \rrbracket$	$\triangleq$	$f \langle v \rangle$
$\llbracket f(x).P \rrbracket$	$\triangleq$	$def^{world} f(x) \triangleright \llbracket P \rrbracket in 0$
$\llbracket !f(x).P \rrbracket$	$\triangleq$	$def^{world} !f(x) \triangleright \llbracket P \rrbracket in 0$

---

Table 7: The encoding of  $\pi$ -calculus

In order to study the extent to which this encoding is adequate, we first proceed towards defining (weak) barbed congruence in our calculus following the definitions in  $\pi$ -calculus [23]. Essentially, we assimilate the  $\pi$  input action with the CAC definition  $f(\tilde{x}) \triangleright P$  in  $D$  from  $a(D)[P]$ , the  $\pi$  output action with the CAC macro call  $f \langle \tilde{v} \rangle$ , and we add to the potential internal actions  $\tau$  the CAC definition  $def^{world} f(\tilde{x}) \triangleright P in 0$  and its banged variant. Given this assimilation, behavioural equivalence in CAC is identical to that in  $\pi$ .

**Definition (Observability predicates).** For each macro name  $\mu$ , the *observability predicate*  $\downarrow_\mu$  is defined by:

1.  $P \downarrow_\mu$  if  $P$  can be written in the form  $\mathbf{A}[a(\mu \langle \tilde{x} \rangle \triangleright Q, D)[R]]$ , where contexts  $\mathbf{A}$  are the active contexts from Table 5,  $a$  is any ambient,  $D$  is any list of definitions,  $\tilde{x}$  is any tuple of names,  $Q, R$  are any processes, and  $\mu$  is free in  $\mathbf{A}$ ;
2. also,  $P \downarrow_\mu$  if  $P$  can be written in the form  $\mathbf{A}[\mu \langle \tilde{v} \rangle]$ , where contexts  $\mathbf{A}$  are the active contexts,  $\tilde{v}$  is any tuple of names, and  $\mu$  is free in  $\mathbf{A}$ .

Using this, we define the weak observability predicates abstracting from internal reactions. In the following, we denote by a context  $\mathbf{C}$  the general context, which has the exact syntax of a process from Table 1, with one hole in the place of any internal process.

**Definition (Internal reaction).** An *internal reaction* is an intra-action given by either of:

1. the (*Reaction*) or (*!Reaction*) transitions in Table 6;

2. the (*Definition*) transition in Table 6, with the destination ambient  $a$  being *world*.

**Notation (Weak observability predicates).**  $\Downarrow_\mu$  is  $\Rightarrow\downarrow_\mu$ , where  $\Rightarrow$  is the reflexive, transitive closure of the internal transition.

We then define (weak) barbed bisimilarity and (weak) barbed congruence as follows.

**Definition (Barbed bisimilarity).** *Barbed bisimilarity* is the largest symmetric relation  $\approx$ , such that whenever  $P \approx Q$ ,

1.  $P \downarrow_\mu$  implies  $Q \downarrow_\mu$ ;
2.  $P \Rightarrow P'$  implies  $Q \Rightarrow \approx P'$ .

**Definition (Barbed congruence).** Processes  $P, Q$  are barbed congruent,  $P \cong^c Q$ , if  $\mathbf{C}[P] \approx \mathbf{C}[Q]$  for any context  $\mathbf{C}$ .

Given these definitions, we claim (the proof being work-in-progress) that our  $\pi$  encoding in CAC differentiates between processes at least as much as  $\pi$ .

**Claim (Adequacy).** For any  $\pi$  processes  $P$  and  $Q$ , whenever  $\llbracket P \rrbracket \cong^c \llbracket Q \rrbracket$ , then  $P \cong^c Q$ .

### Related literature

A framework for context awareness is called *direct* if it deals with context awareness explicitly through languages and theories, instead of employing middlewares for hiding the details of context from the application. A brief survey of few direct models of context awareness which stand out – with a focus on process calculi – is given in the following.

Birkedal's [3] proposes a complex model of context awareness able to model both the usual reconfigurations of the context, and queries upon context; noting that context queries cannot be naturally modelled with one bigraphical reactive system (BRS), it proposes a solution (called a Plato-graphical model) which comprises three BRSs: the context  $\mathbf{C}$ , its observed image or proxy  $\mathbf{P}$  and the computational agents  $\mathbf{A}$ , such that its expressivity suits well sophisticated real-world context-aware systems. For a full investigation upon logical reasoning with Plato-graphical models, a notion of equivalence of BRSs (yet unknown) is needed, and not just an equivalence of bigraphs of a single BRS.

Braione's [5] builds *contextual reactive systems (CRS)* upon *reactive systems, RS*. The only difference between a CRS and a RS is the presence of a function which captures an association between elementary rules and their allowed reaction contexts, so that a CRS can express inhibitor and enabler factors for interaction. With regard to logical reasoning with CRSs, with [5] being a work-in-progress, an investigation of desirable operational congruences is under way.

Roman's [22] builds a language-based model of context-aware systems, an interesting feature of which is the fact that agents have as context the *exposed* (not *private*) variables of other agents. [22] also has a limited associated proof logic, with program properties being expressed using a small set of predicate relations whose validity can be derived directly from the program text, indirectly through translation of program text into formal constructs, or from other properties through the application of inference rules.

## 4 Conclusions and Future Work

We have presented work covering two different aspects of ubiquitous computing. Section 2 consists of a sensor network service discovery protocol tailored for a special application framework, in which the workings of the networking layer (routing and service discovery, embedded into one protocol) can be optimized by allowing knowledge of user context even at the networking layer. Section 3 describes a context-awareness process calculus building upon mobile ambients and introducing a modelling of computing context, distributed through the network and expressed by macros; key features include the expressing of contextual commands and context-triggered actions, and a mechanism for extending the visibility of context in the network through explicit context provision and discovery. The practical experience given by the actual building of the networking system in section 2 played an important part in grounding the formal work from section 3 into practice.

### Future work

Having developed a context-awareness calculus expressive enough to encode a set of scenarios from practical systems, we now look into system verification techniques (essentially, mechanisms to formalize and prove properties of systems) using this tool.

Hennessy [15] submits to study two inherent features of context-aware systems:

- the *behaviour* of an entity is the observable outcome, over time, of the entity's interaction with the changing environment; it is tightly linked with the knowledge the entity has of the (public capabilities of the) context;
- the *properties* of an entity are the view upon this entity from the context; it consists, at every moment in time, of the set of (public) capabilities that the entity makes available to the context; thus, the behaviour of an entity depends on the dynamics of properties visible in the context.

Given this, the study of *contextual behaviour* reasons about the equivalence in behaviour of two processes  $P$  and  $Q$  running in any contexts which publish to  $P$  and  $Q$  a set of properties  $\mathcal{A}$  and also have knowledge about  $P$  and  $Q$ 's set of properties  $\mathcal{B}$ :  $\mathcal{AB} \models P \approx Q$ . Following this approach for behavioural study, we find interesting properties of systems to include:

- system *performance* and *coverage*; for example, given a certain topology of the network and certain positions for deploying resources in the network, together with the resources' policy of context provision, it can be asked if a mobile entity is always able to get hold of a resource, regardless of its current position;
- system *security*; access restrictions can be thought of, just like general data communication, to also be a feature of context; after expressing firewall rules formally, it can be asked if such a rule will have the expected outcome given all the possible properties of processes involved.

We intend to retrieve our study cases from real-life applications, and test their expressiveness against such practical scenarios. We also consider, given the dual contribution in this report, a way of putting an experimental face to the formal tools by using the context-awareness paradigm in section 3 to develop actual system prototypes.

### **Acknowledgements**

This work is supported in part by a KIRK telecom and ISIS Katrinebjerg project entitled "Fokus på fremtiden: arkitektur, applikationer og grænseflader til trådløs telefoni". The author would like to thank Jakob Bardram for the chance to participate in the Activity-Based Computing project and Mogens Nielsen for his great ideas and never-failing support.



## References

- [1] Jakob E. Bardram. Activity-Based Computing: Support for Mobility and Collaboration in Ubiquitous Computing. *Personal and Ubiquitous Computing*, 9(5):312–322, July 2005.
- [2] 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.
- [3] L. Birkedal, S. Debois, E. Elsborg, T. Hildebrandt, and H. Niss. Bigraphical models of context-aware systems, 2005.
- [4] Gerard Boudol. Asynchrony and the pi-calculus. Technical Report RR-1702.
- [5] Pietro Braione. Operational congruences for contextual reactive systems. Technical Report Technical report 2004.33, DEI, Politecnico di Milano.
- [6] Doina Bucur and Jakob E. Bardram. Resource Discovery in Activity-Based Sensor Networks. In *Proceedings of the First International Conference on Pervasive Computing Technologies for Healthcare*, 2006.
- [7] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures: First International Conference, FOSSACS '98*. Springer-Verlag, Berlin Germany, 1998.
- [8] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin. GSD: A Novel Group-based Service Discovery Protocol for MANETS. *Mobile and Wireless Communication Networks*, 2002.
- [9] Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Dartmouth Computer Science Technical Report TR2000-381, 2000.
- [10] Request for Comments 2608. Service Location Protocol Version 2. <http://www.openslp.org/doc/rfc/rfc2608.txt>.
- [11] UPnP Forum. PnP Device Architecture. <http://www.upnp.org/>.
- [12] David Gay, Philip Levis, and Robert von Behren. The nesC Language: A Holistic Approach to Networked Embedded Systems. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI)*, 2003.
- [13] Zygmunt J. Haas, Marc R. Pearlman, and Prince Samar. The Zone Routing Protocol (ZRP) for Ad hoc Networks. *IETF MANET Internet Draft*, 2002.
- [14] A. Helmy, S. Garg, N. Nahata, and P. Pamu. CARD: A Contact-based Architecture for Resource Discovery in Wireless Ad Hoc Networks. *Springer Mobile Networks and Applications*, 10:99–113, 2005.
- [15] Matthew Hennessy. Context-awareness: Models and analysis. Talk given at the second UK-UbiNet Workshop: Security, trust, privacy and theory for ubiquitous computing, Cambridge UK, May 2004.
- [16] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. *Lecture Notes in Computer Science*, 512:133–??, 1991.

- [17] Omer Sinan Kaya. NesCT: A language translator. <http://nesct.sourceforge.net/>.
- [18] R. Koodli and C. E. Perkins. Service Discovery in On-Demand Ad hoc Networks. *IETF MANET Internet Draft*, 2002.
- [19] Sun Microsystems. Jini Technology Architectural Overview. <http://www.sun.com/software/jini/whitepapers/architecture.html>.
- [20] Robin Milner. A Scientific Horizon for Computing. Lecture at IFIP World Computer Congress, 2004.
- [21] Robin Milner. Turing, Computing and Communication. Essay and lecture at King's College, Cambridge, to celebrate the 60th anniversary of the publication of Turing's paper On computable numbers (1937), 1997.
- [22] Gruia-Catalin Roman, Christine Julien, and Jamie Payton. A formal treatment of context-awareness. In *Proceedings of the 7th International Conference on Fundamental Approaches to Software Engineering (FASE 2004)*, pages 12–36. Lecture Notes in Computer Science 2984, Springer, 2004.
- [23] Davide Sangiorgi and David Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
- [24] 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.
- [25] The Open TinyOS Community. OMNeT++: Discrete Event Simulation System. <http://www.omnetpp.org/>.
- [26] The Open TinyOS Community. TinyOS. <http://www.tinyos.net/>.
- [27] Christopher N. Ververidis and George C. Polyzos. Extended ZRP: a Routing Layer Based Service Discovery Protocol for Mobile Ad Hoc Networks. In *MobiQuitous*, pages 65–72. IEEE Computer Society, 2005.
- [28] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 1991.
- [29] Mark Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 1993.
- [30] Pascal Zimmer. A calculus for context-awareness. Technical Report BRICS Report Series RS-05-27.