

**A Survey of Works and an Identification of  
Challenges within the field of Context-Awareness  
Supported Application Mobility**

Dan Johansson  
Luleå University of Technology

2009-03-18

## **Abstract**

The number of mobile users grows constantly; an important demand is for IT to be accessible always and everywhere. Users have gone from being stationary to mobile, accessing and using their electronic services everywhere, even while moving. But not only users can be mobile; when migrating an application from one device to another, one has achieved *application mobility*. As migration is often carried out in heterogeneous environments, applications have to be able to receive and manage input from the user and the surroundings. Taking interfaces and gathered information about the context (defined as any information that can be used to characterize the situation of an entity) with them while migrating, the notion of context-awareness supported application mobility has become a reality, and this would indeed enhance the field of ubiquitous computing, where the vision is often hidden, yet always accessible applications.

The purpose of this survey is twofold: to compile previous work on context-awareness supported application mobility and to identify challenges connected to the area. The challenges that are identified concern how to identify and keep track of applications that move between devices; how to best design the model for context-awareness; how to attain and communicate a good enough context quality; how to achieve seamlessness and minimize downtime while executing a migration; and finally how to handle heterogeneity within and between devices, in shape of differences in hardware and software.

# 1 Background

The number of mobile users grows constantly; the International Telecommunication Union (ITU, 2008) expects the number of global mobile phone subscribers to reach 4.5 billions in 2012. Mobile users communicate within what Castells (1996/2001) calls a space of flows, transcending the traditional space of places. In this space of flows, physical boundaries dissolve and both information and technology roams through time and space; an important demand is for IT to be accessible always and everywhere. The users have gone from being stationary to nomadic, being able to access and use the same service on many locations, and then truly mobile, accessing and using the same service everywhere, even while moving.

There are four traditional types of mobility (ITU, 2002): *terminal mobility*, allowing a device to change location and still be able to communicate; *personal mobility* (or user mobility), when a user can keep his or her user identity irrespective of terminal or network; *session mobility* (or continuous user mobility), letting the user change location or device and still be able to keep media streams active; and lastly *service mobility*, making a particular service accessible by the user, regardless of terminal or network.

To be able to reach mobility as an IT user, one needs at least four different components: a mobile device; a wireless network, supporting mobility; an application providing an interface for communication in some sort; and a service. Mobile devices come in many shapes, the most common being the mobile phone. Examples of other mobile devices are smartphones, personal digital assistants (PDA:s), handheld game consoles and other similar IT enhanced items. Laptops form a category in between, being portable but less suitable for mobile use compared to handheld devices. Wireless networks can typically mean WiFi (IEEE 802.11 standard), Wimax (IEEE 802.16 standard), or wireless cellular networks, but can also be implemented at short range through Bluetooth, IR communication or RFID. Applications and services will be discussed later on.

An area closely linked to mobile computing is context-awareness. Some authors go as far as naming it the “*next computing paradigm in which infrastructure and services are seamlessly available anywhere, anytime, and in any format*” (Anagnostopoulos, Tsounis & Hadjiefthymiades, 2007, p. 445). Barnard et al (2007) show how context has

an immense impact upon performance in mobile computing systems. Indeed would context-awareness be a key component in realizing Mark Weisers (1993, 1994) frequently cited visions about ubiquitous computing, with invisible, yet always accessible applications.

Applications can be used from afar, rendering some mobility. This can for example be realized through virtualization technology (David et al, 2007). But mobility can also be extended to the application; when migrating the application from one device to another, one has achieved *application mobility*. Taking interfaces and gathered information about the context with them, the notion of context-awareness supported application mobility has become a reality.

The purpose of this survey is twofold: to compile previous work on context-awareness supported application mobility and to identify challenges connected to the area.

To make it possible to discuss context-awareness supported application mobility, it is important to first define and shortly discuss the notion of context, context-awareness and application mobility respectively. Section 2 of this survey will be devoted to presentation and comparison of projects within context-awareness supported application mobility, while clusters of major challenges in the area are identified and put forth in section 3. Within section 4, I conclude by discussing the use of context-awareness supported application mobility.

## **1.1 Application mobility**

Applications are IT artifacts (as defined by Löwgren & Stolterman, 1998), manmade, which may or may not be abstract, and which have information technology as the basic element in their fundamental structures and functionalities. More specifically, applications are software, designed to support the users intended tasks or improve his or her work.

Applications can be moved from one device to another, sometimes while running, and without having to restart. As early as 1995, Bharat and Cardelli presented migratory applications as:

*“a new genre of user interface applications that can migrate from one machine to another, taking their user interface and application contexts with them, and continue from where they left off. Such applications are not tied to one user or one machine, and can roam freely over the network, rendering service to a community of users, gathering human input and interacting with people.”* (p. 133)

Bharat and Cardelli (1997) also define some semantics that can be used to describe certain aspects of the application mobility area. Among other things, they introduce three kinds of state transmissions, the first being *network transmission*. This is when only some parts of an application is actually copied and transmitted to another device, while other parts are replaced by references to the original objects. In Obliq, the object-oriented language resembling Java used by Bharat and Cardelli, the objects are never copied during network transmissions, as they have state. Instead, a pointer is transmitted in its place. A more profound transmission is the *network copy*, when a complete copy of the application is taken and transferred to the new device. When, for instance, moving user interfaces, relying on windows, threads and other site-dependent resources, a snapshot of the application is assembled, and then completely copied to the new device. The third type of migration is called *agent migration*. An agent is *“a computation that may hop from site to site over the network”* (p. 135), parameterized with a suitcase and a briefing. A *suitcase* is the ‘bag’ of data an agent carries with it when moving; it might contain tasks to perform, where to perform them or results of earlier tasks. The *briefing* is the data an agent receives when hopping to a new device. This is made possible through an *agent server*, a program that can receive code, execute it, and also provide it with a local briefing and *hop instructions*, telling the agent when and how to move from one device to another. If the suitcase holds the entire application state, application mobility is achieved.

In this survey, Koponen, Gurtov and Nikander’s (2005) definition of application mobility is used; a definition building on an earlier one by Milojičić et al (2000), when describing application mobility as *“the act of moving a process or an application between hosts during its execution.”* (p. 1). This should not be confused with *partial migration* (2004), when controlling an application on one platform from another – the actual process or application must be migrated. Closely-related, when copying an

application to a new site without destroying the original application, it is called *cloning* (Bharat & Cardelli, 1997). A ‘weak’ type of application mobility is code mobility, defined by Fuggetta, Picco and Vigna (1998) as migrating code from one device to another, but not the states of the application. A strongly mobile application should be able to migrate both its code and its execution state, along with variables and all important data (Cabri, Leonardi & Quitadamo, 2006). Thus, code mobility cannot be regarded as ‘true’ application mobility, when compared to the definition given by Koponen et al (2005). Limited application mobility often relies on a certain amount of *checkpointing*, “a limited form of process migration; an application is periodically freezed for a short period of time and the operating system (or the application itself) stores application state to a persistent storage.” (ibid, p. 1), which is the case with systems like Porch (Ramkumar & Strumpfen, 1997). Stronger application mobility demands a more intricate and advanced bidirectional interface between the source and the target (Koponen et al, 2005). The borders are, as we will see below, rather undefined.

Application migration usually follows a given set of procedures: first the application state is captured and represented using metadata (this is often called *serialization*); secondly, a transmission is carried out, when the serialized state is moved to another device and deserialized; and the finally, the application is resumed (Milojčić et al, 2000). Another popular naming of the three phases is suspension, migration and resumption (Zhou et al, 2007).

Bharat and Cardelli (1997) carry out a simple application migration, using Obliq and its migration command. First the destination’s agent server is contacted to check that a migration is possible. If the agent server complies, the user-interface’s state is checkpointed and Obliq objects are created. The user interface is then destroyed, and so are all links to local runtime. A suitcase is prepared and the hop instruction is then executed, creating a network copy. If the migration should fail, the interface can rebuild itself from the saved states. This is often called *to unpickle*, saving a serialized object to a buffer (*to pickle*), and then saving the buffer to a file which can be reused if needed. This makes the system persistent. Garbage collection sees to that applications can hop in and out of a device without causing memory losses.

Du, Sun and Chanchio (2003) present a middleware called HPCM that takes source code from an application running on a device as input, remembers it's current memory state and settings and migrates it to another device, using TCP/IP for communication. The move is initiated through a console/scheduler, starting a pre-compilation of the application's C code into so called annotated, migration capable C code. System states are stored into libraries, working together with a run-time process to carry out the migration. Even though the actual move can be carried out rather quickly, much pre-processing and calculation is needed before the migration can take place.

Koponen, Gurtov and Nikander (2005) make use of Host Identification Protocol, a way of separating the end-point identifier and locator roles of regular IP addresses. This is done through introducing a new host identity name, based on hashed public keys. HIP is added to the protocol stack between transport and network layers. A migration component keeps track of and executes the suspending of the application, conducting the actual transfer of the processes and their states, at the same time seeing to that the applications continue running on the new device. The proposed architecture will however rely on a certain amount of checkpointing, when processes are bundled into so called process domains before being transferred. These domains contain both virtual and static views for the packed processes.

Zhang and Pande (2006) make use of a compiler driven framework, pre-serializing application states during execution to as big an extent as possible. Some downtime still occurs when stopping and transmitting the application data, and also when deserializing on the target device. The platform used is .NET, providing XML serialization (allowing good readability for humans and also good compatibility between different platforms), binary serialization (converting a complete object to a binary stream) and SOAP serialization.

## **1.2 Context**

Compared to the other notions discussed in this survey, the notion of context is the hardest one to reach unity around. Traditionally within computer science, context has

often been treated as an equivalent to location, but other aspects have also been identified, such as network connectivity, noise level, communication costs, bandwidth and more. This is what Brynskov et al (2003) call *physical context*. The three most important aspects of context, as put forth by Schilit, Adams & Want (1994), are the questions of where you are, who you are with and what resources that are nearby; all very closely bound to matters of location.

Dey and Abowd (1999, p. 3-4) state that context rather should be seen as “*any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*” An important part of this definition is that the information has to be relevant to the interaction; otherwise it is not to be regarded as context. Chen and Kotz (2000, p. 3) acknowledge and add to this definition in stating that: “*Context is the set of environmental states and settings that either determines an application’s behavior or in which an application event occurs and is interesting to the user.*” They thereby also differ between active and passive context, the former actively influencing the behavior of an application while the latter might be relevant, but not essential in any way to the application. Brynskov et al (2003) capture this expanded meaning in the notions of *digital context*, the relationship of objects such as system models services, relative time and space and so on, and *conceptual context*, containing user intentions and understanding, completing the trinity of context aspects along with physical context, as discussed above. Other taxonomies can of course also be found, depending on which aspects one wish to emphasize and compare.

Dourish (2004) makes an excellent point when displaying the two extremist views of context: the technical notion and the notion drawn from social science. The technical notion, used by many programmers and designers, describes context as information that can be known, that is delineable and stable and that it is separable from the actual activity. If one instead sees context as an interactional problem, contextuality is a relational property; in other words, information may only be regarded as context if it is relevant to the particular activity. As a result of this, context is not invariable, but rather decided dynamically and bound to each particular occasion of activity. Context “*arises from the activity*” (ibid, p. 22) and is actively created. Designer conceptions regarding

what to be treated as context might therefore differ from the meaning that arises in the actual use of the IT artefact.

Another taxonomy that can be used is to distinguish between primary and secondary context; *primary context* being the context related to the actual situation, and *secondary context* being context data related to the user or the application, not relevant in the specific case (Dey & Abowd, 1999). When a person for example wishes to locate and use a printer as an output device, the user location, printer location, printer quality, printer queues etc could be regarded as primary context, while the user's phone number and email address might be considered secondary context – related to the user, but not relevant in the actual finding and using printer-case.

There are two major ways of using context: 1) encoding context along with information to be used as a retrieval cue or 2) use context dynamically to tailor the behavior of the system or respond to patterns of use (Dourish, 2004). In a mobile and dynamic setting, I would claim the latter to be the most important, as the system needs to adapt to the environments and the situations it is confronted with.

In a computer system, context can be modeled in many ways. The traditional view has rested on a definition of context as something fixed and stable, a set of attributes which does not change. Chen and Kotz (2000) list key-value pairs; tagged encoding (nowadays often XML based, such as the markup scheme models derived from SGML); object oriented models; and logic-based models, the last one relying on a set of rules that express context data as facts through an entity-relationship data model. Furthermore, one can add graphical models, for example models based on Unified Modeling Language (OMG, 2009) and ontology-based models, which uses concepts and relations between these concepts to represent context (Schmol & Baumgarten, 2008a), defining a hierarchy of entities, for example based on what the entities can do for the user. In short, ontologies can be defined as a set of terms and how they relate to each other within a certain domain. Ontologies can have multiple parent objects; for example can a video device be a subclass of both audio presentation and visual presentation (Ranganathan, Shankar & Campbell, 2005). Anderson, Hansen and Bouvin (2006) shape context in the form of queries. This renders dynamical updates possible to what the current context should be,

for every given time and situation. Du and Wang (2008) see context as points in multi-dimensional context spaces, where the dimensions might relate to user, location, speed or other things, depending on their relevance.

### 1.3 Context-awareness

Dey (2001) writes that “*When humans talk with humans, they are able to use implicit situational information, or context, to increase the conversational bandwidth [...] By improving the computer’s access to context, we increase the richness of communication in human-computer interaction*” (p. 4). Improving the computer’s access to context is a first step of making a system context-aware.

According to Dey and Abowd (1999, p. 6) a system is context-aware if “*it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task*”. Schilit, Adams & Want (1994) were among the first to discuss context-awareness within the area of computing, defining context awareness through describing four categories of context aware applications. *Proximate selection* helps the user to choose between several available options, using information about the location and selection rules to create the awareness. For example, the printers best suitable for a certain job could be emphasized by the application and presented to the user. Quality of print jobs, the printer’s distance from the user, the cost and so on could be used to calculate which printers to recommend. Secondly, *automatic contextual reconfiguration* is when altering connections between components in the context aware system and/or adding and removing components. The third category, *contextual information and commands*, is when context awareness is used to provide the user or the system with certain information or views, depending on the actual context, for example pointing out information about work appointments at work during working hours, but hiding it when at home or on vacation. Adaptive user interfaces can be sorted under this category; Mitrovic and Mena (2002), Bandelloni and Paternò (2004), Schmidt and Hauck (2007) and van Tonder and Wesson (2008), to name a few, demonstrate and discuss such interfaces and methods. Ranganathan, Shankar and Campbell (2005) apply the term application polymorphism to applications that change their structure to adapt to different

environments. Finally, *context-triggered actions* are simple IF-THEN rules that dynamically see to that different procedures are started, stopped or altered, depending on if certain conditions are met (Schilit, Adams & Want, 1994). Dey (2001) prefers a more generalized approach to what categories of features that a context-aware system can support, naming “*presentation of information and services to a user, automatic execution of a service for a user and tagging of context to information to support later retrieval*” (p. 5).

Much can be won by not only relying on the user to explicitly decide the context, but to make the IT artifact aware of the use situation and its meaning. Häkkinen and Mäntyjärvi (2006), for instance, acknowledge the device’s role by defining context-awareness as when: “*the device is, at least to some extent (sic!), aware of its usage situation*” (p.1). Therefore, to achieve context awareness, the system needs to be able to receive and manage input from the user and the environment, and also reason and react upon this input. Input can be gathered through sensors, defined by Loke (2006) as software and/or hardware built for that specific reason. Sensor input is not necessarily equivalent to context, as it takes the form of raw data, which might or might not be relevant to the user or the usage situation. Gellersen, Schmidt and Beigl (2002) argue that one often needs multiple sensors, detecting different types of context data, to capture the situational context.

A number of sensors common in handheld mobile devices are listed by Hickley et al (2000), namely touch sensors, tilt sensors and proximity sensors. Touch sensors can for example be used to detect if a user is holding a device, if she is pressing a certain button or (in a non mobile setting) if a chair is empty. Tilt sensors are used to measure the angle of which a device is held, but can also be used to receive data on how fast the angle is switched, and in what directions, through an accelerometer. Shaking the device will of course give rise to a certain chaotic input, which can be caught and interpreted as well as more subtle and deliberate tilts. Proximity sensors can measure the distance between the device and/or the user and other places or artifacts of interest. One such technique is RFID, Radio-frequency identification, allowing transponders (or tags) to be attached or incorporated in different objects, giving them specific id:s, allowing wireless automatic identification and data capture (Ahson & Ilyas, 2008). As the RFID tags use incoming

radio waves to power their responses, they need no batteries, although there are examples of battery powered tags as well. Reading distances vary from ten centimeters up to six meters (Moh et al, 2008). A very good, as well as amusing, example of RFID use is presented by Coulton, Rashid and Edwards (2008), demonstrating a live RFID based version of Pacman, with real people taking the roles as Pacman and the ghosts, hunting tags and each other with their mobile phones in university corridors. Location on a more global scale can be identified by techniques such as Global positioning system (GPS), a very common way to connect a mobile device (such as a modern mobile phone) to its current location, even while moving. Of course, there are also many other different types of sensors, ranging from optical sensors to rain sensors. Pretty much anything that can measure and quantify something from the environment and provide the context-aware system with this data can be counted as a sensor. Sensors show a great range in complexity, from being simple binary on-off mechanisms to sensors that can capture a whole meaning of a situation, for example if a meeting is taking place in a room or not (Kummerfeld et al, 2003).

However, Anderson, Hansen and Bouvin (2006) underline that the challenge lies not so much in acquiring data, but to determine which collection of data, that at a given time, best represents the current context. Reasoning therefore becomes crucial when trying to gain context-awareness. Except from sensing, a context-aware system has to be able to think and act (Loke, 2006). Albeit being a powerful calculator, the computer's ability to think and reason is limited compared to the human brain in many ways. Thinking in a context-aware system is often achieved by taking sensor data and letting the system use a certain set of techniques to make decisions. Among these techniques are physical mathematical models, feature-based inference techniques, and cognitive-based models. Physical mathematical models can for example be based on Kalman filtering (Bishop & Welch, 2001), a model which has become increasingly important when it comes to sensing the real world through computer systems. With this method, mathematical equations are continuously used on noisy sensor measurements to predict and correct the data, making stochastic estimations from incremental results.

Another useful method that stands out is the hidden Markov model (Rabiner, 1989). This statistical method can be used to determine hidden parameters from data collected

by sensors; based on observations, estimations can be made about conditions not directly observable. If we, for example, know that Alice is keener to use the internet during rainy days and days when she is free from work, we can use sensor data about Alice's internet usage in combination with the hidden Markov model to draw conclusions about non observable parameters, such as her working schedule and the local weather conditions in Alice's village.

Cognitive-based models often include fuzzy logic (Mendel, 1995), which can handle both numerical data and linguistic knowledge. This is particularly useful when comparing pieces of heterogeneous context data, or when reasoning is approximate rather than exact, as in calculating degrees of possibilities or truths. When dealing with both objective and subjective knowledge within the context model, fuzzy logic is a highly suitable method.

Many works have been conducted in the field of context-awareness in mobile settings, and many different architectures have been proposed. An architecture (or rather a basic framework for an architecture) that's often referred to is Dey's (2001) context toolkit, supporting acquisition of and access to context, storage of context related data, distribution, and independent execution from applications. Dey identifies three abstractions that a context-aware middleware should contain, namely *widgets* (acquiring raw context data, as single pieces of information), *interpreters* (putting raw data together, discovering its meaning) and *aggregators* (compiling context connected to an entity, for example a specific person or an application). A small system can therefore have a very simple architecture. An example of such a design is presented by Wickramasuriya, Mehrotra and Venkatasubramanian (2008), supporting a system of RFID-enabled privacy-preserving video surveillance. The system consists of a sensing module, processing data from all the sensors. This module communicates with a data management module, holding the XML-based policy engine, keeping track of users and preferences. Auxiliary services (such as methods of blurring the incoming video to preserve privacy) can be added to the latter module, and at the end of the line-shaped architecture we find an output module, handling logging, reporting and, in this particular case, video rendering. Another simple framework is presented by Ahn (2008), focusing on a single context parameter, namely location, and consisting of sensors, gathering data to a mobile

object database through a context manager, connected to the user via a query handling subsystem. Attached is also an inference engine, making context-aware decisions through responding to if-else rules.

Du and Wang (2008) present a layer model, consisting of an application layer, a context conversion layer and a device layer. The application layer holds specifications of application context, behavior and bindings. The application context is represented by a context space, holding all the possible contexts that the application uses. The context space is defined as “*the Cartesian product of multiple context dimensions. [...] A coordinate of dimension  $d$  takes the format of  $d[i]$  or  $d[v]$ , where  $i$  is an integer index and  $v$  is the actual context value*” (p. 216). The behavior is specified through an application class and all its methods. The binding specifications link contexts to behaviors. The second layer of the model, the context conversion layer, is responsible for interpreting raw data, provided by the low device layer. The device layer contains sensor components, local or remote, independent of any particular context-aware application. In the same article, Du and Wang (ibid) also present an implementation framework, building on the three layer model. The framework contains components that together fulfill the different tasks connected to each layer, as discussed above.

As supported by the context toolkit (Dey, 2001), context-awareness is often added to a system through a middleware. Mikalsen et al (2006) have developed a model for a generic mobility and adaptation enabling middleware (MADAM) consisting of a *context manager*, which monitors changes in user requirements and situational context; an *adaptation manager*, modifying system behavior; and a *configurator*, reconfiguring the application according to the adaptation manager’s decisions. Focus lies on the context manager, containing a repository keeping track of data needed by sensors, reasoners and the historical context storage module. The context manager provides a standardized way to represent context, standardized interfaces, storage and retrieval functions and consistency (seeing to that no context elements or components are duplicated within the system).

CASP is a middleware approach for a context-aware platform, developed by Devaraju, Hoh and Hartley (2007). It has five components, namely context sensing,

context modeling, context association, context storage and retrieval. The messaging within the system is XML-based, and both client-side and server-side APIs facilitates communication with the system.

Another middleware approach is suggested by da Rocha, Endler and de Siqueira (2008), building on *context domains*, each comprising of a set of context types, a design of the context storage, rules for managing clients within the domain and information about what relationship the context domain should have towards its sub-domains. The context domain is run by a CMN (Context Management Node), enabling the functions above. The researchers have designed a prototype in Java with IP-based protocols, and are currently in the final stage of testing.

Schmol and Baumgarten (2008a & 2008b) have tried to summarize the common characteristics of context-aware computing, creating a general architecture for context-aware middleware. Their work has resulted in a layer model (deriving from the layers presented by Christopoulou, Goumopoulos and Kameas (2005), consisting of lexical, syntactical, reasoning, planning and interaction levels). Sensors capture raw data at the lexical level, being refined at the syntactical level by the *context capturing interface* into context information. The chosen context form is the ontology, as described in section 1.2, but this can be replaced if the system hardware capabilities are found inadequate. A *context repository* contributes with persistent storage at the reasoning level, while an *inference engine*, working at the planning level, alters the context into inference rules. Between the last two layers, a context API provides access to the context for the *context application*. Some cross-layer communication is embedded within the architecture, as the context API has a direct line of communication with the context repository, being able to read the current context and at the same time make updates directly from the user. Apart from being altered by the user or the environment, the context reasoned can also be derived by the inference engine alone (Schmol & Baumgarten, 2008b).

## 2 Context-awareness supported application mobility

Deriving from the definitions of context, context-awareness, applications and mobility respectively, context-awareness supported application mobility could be described as:

*“Using context, in the act of moving a process or an application between hosts during its execution, to provide relevant information and/or services, where relevancy depends on the user’s task.”*

David et al (2007) use the synonym *adaptive application mobility*, with approximately the same meaning.

Context-awareness supported application mobility is a rather young research area. Ranganathan, Shankar and Campbell (2005) state that works in application mobility until recent years had *“little or no support for adaption based on context information. Further they did not have the concept of substituting application components by other semantically similar components. Also, application structure was not altered during adaptation.”* (p. 111). Among earlier works worthy of note is Nakajima et al (1999), proposing a system for migratory continuous media applications, where applications are moved between systems depending on user location. Building on a three pillar architecture, being an environment server (keeping track of context), a continuous media toolkit (through which applications are built) and a migration manager (saving application states and migrating them), the authors create applications that are environment-aware and can migrate. The notion of context only stretches as far as location. We also find Phan et al (2001), who in their system handoff application sessions across heterogeneous platforms, with basic changes in GUIs. Sousa and Garland (2002) show a system called Aura, allowing limited migration bordering on the nomadic rather than the truly mobile, migrating tasks between Windows and Linux platforms. MobiDesk (by Baratto et al, 2004) moves a private working environment between work stations, but relies on a constant network connection as the applications have sessions and application logic placed on proxy servers. Grimm et al (2004), present an architecture for discovery and migration called one.world, however lacking adaptation capabilities.

Bandelloni and Paternò (2004) migrates a web application keeping its runtime state, while at the same time adapting its interface to the device its being moved to. Using a

migration server to manage context information and supporting migration requests, even light-weight devices, short on processing capabilities, can be used in the system. A device that wants to access the migration service just downloads a migration client from the server; the intelligence of the system is located on the server side. When requesting a migration, the client contacts the server, which in turn loads the migration client on the target device. The runtime context is collected at the source and sent to the server along with the source URL. The migration server maps the incoming application so it will fit the target, before finally moving the tailored application.

Siu et al (2004) show how to achieve context-aware application migration through the use of a software infrastructure platform called Sparkle, completely implemented in Java. Sparkle use applications composed of *facets* – functional units located on network servers – that are not connected to certain data or user interfaces; execution state and interfaces are instead stored in a *container*, thus separated from the facets. When an application needs certain functionality, it makes a request whereupon the most suitable facet is chosen to provide that functionality. A *facet manager* gathers usage statistics, which can then be used by a *mobility manager*, to pre-load facets (thus reducing migration time). The mobility manager tries to decide when an application migration should be carried out (and where to), using information from a *context state manager*. If a migration is called for, a *data state manager* capture the application state (the data, execution state and interfaces stored in the container). Most of the system and context state data is stored as XML. The whole process can be summarized in the form of state capture, state processing, state transmission, another state processing at the destination device, and finally a state restoring phase. Tests were carried out with a universal browser and two small computer games as the applications being migrated. The system used in the test consisted of a desktop PC (2.26GHz) and a notebook PC (1133MHz), with migration latencies of around 4000 ms when transferring 2 – 2,5kB.

Ranganathan, Shankar and Campbell (2005) build a context-awareness supported application mobility framework, using the term *application polymorphism* to describe applications that “*can change their structure in order to adapt to different environments and recover from failures*” (p. 109). Their applications have three components, handling input, output and logic respectively. This means that the application can

divide its components to different devices, for example using a widescreen TV as display, a keyboard for input and a mobile phone for control. The context-aware adaptation takes place when a migration is executed, or when failure arises in one of the running components. Three different types of adaptations can be made: a change in types of components (the authors give an example of switching Acrobat Reader for PowerPoint or vice versa), a change of devices (for example switching the widescreen TV display in the example above to a speaker) but also a change in the actual number of components used – maybe several displays can be used, or a display in combination with a speaker. The adaptation process relies heavily on what the authors call semantic similarity, which is determined through using ontologies based on functionality and behavior (see section 1.2). No matter the adaptations, the application keeps its semantics, which are based on the users' originally intended tasks. When an adaptation is triggered, a semantic check is carried through, after which the system associates the application with a device, finally instantiating and executing the application. Typical context used in the framework can be location (of users and devices), information about which application (if any) are running on the different devices, the user's current activity and so on. The authors have built a prototype framework on top of the Gaia system, a middleware platform for managing physical and digital resources in active spaces by Román et al (2002). An *ontology server* stores all the ontologies in Gaia, and other entities communicate with this server to create and compare concepts. A special algorithm is used to measure similarity between entities, and the results are then used for adaptation. An *application adaptation service* supports the actual adaptation, which can be automatic (through context-awareness supported decisions) but also manually initiated. The application is then suspended by a *migration service*, and the state of the application is stored to a file and then sent to the migration service at the destination space. The destination application adaptation service creates a customized adaptation scheme suitable for the destination space and resends it. Finally the saved application state is obtained, the adaptations made and the migration carried through. A *fault manager* recovers application usage from any failures (such as device failures or network trouble), being alerted by a *presence service* and using checkpoints created periodically in the active space. All components also send so called heartbeats now and then to the

presence service, notifying it that they are still running. In the prototype system built, applications such as slideshows, music players and browsers are migrated between rooms and devices at a university lab. Adaptation phase clocks in at less than 1 ms on a system driven by a Pentium 1.7GHz processor with 1GB RAM memory, but no measurement data is provided regarding device discovery phase and response time from different context infrastructures. The time it takes to restart a failed or migrated component ranges between 1 and 7.5 seconds. (Ranganathan, Shankar & Campbell, 2005)

Hwang, Park and Chung (2006) demonstrate a system for context-aware desktop migration, defined as “*moving a user’s working environment from one computer to another, which includes editing files, playing A/V files, working application, window size, window position, tool bar, wallpaper, and so on*” (p. 1587). The system relies on dynamic linking of application specific libraries. When a user leaves her working place, this situation is caught by a RFID reader. A migration agent is notified, capturing the states of her desktop, and then sending it to a migration server where it is stored along with application-specific library URLs. When the user arrives at her destination, a RFID reader attached to the new computer downloads the stored desktop from the migration server and recreates it. The obvious drawback with this design is of course that it does not take seamlessness into account. Satoh (2005) proposes a system where application mobility is more prominent; he uses an architecture consisting of several computers, forming a *federation*. Within the federation, an application can be dynamically distributed and deployed even while the application is executed. The application can be split onto several devices for input and output, for example a screen showing the GUI, a mobile phone providing input and a computer (without keyboard) for processing. All components consist of collections of Java objects, each with their own identifier and also an identifier of which federation they belong to. Each component can attach to other components through *hooks* – migration policies that enable one component to follow another in case of a migration being executed. There are two types of hooks, attach hooks and follow hooks. When a component moves, an attach hook tells the hooked component to migrate to the same destination if it meets the requirements of the component. A follow hook signals the hooked component to migrate to the destination, or another, better host, that is nearby, should a migration be carried out. The follow hooks are only available when the system can

make use of location sensors and/or location information, knowing where to find suitable devices. Each migration host must have a runtime environment (Java virtual machine) and at least one TCP connection with all hosts close by. The migration process follows a number of steps: first the component registers its policies with the destination host, to find out if it can meet these requirements. A message is then sent back, telling the component if it can migrate to the chosen destination, but also what other devices are close by, maybe along with recommendations, should one of these be suitable as a destination as well. Finally, the destination host instructs the component to migrate to the recommended destination, be it the original destination contacted or another one. The system was tested on a federation consisting of five component hosts (1.2GHz PCs) connected through Fast Ethernet. Focus resided on monitoring migration times of hooked components after an initial component migration had been carried out. Times varied between 40–60 ms. No measurement results were provided for the whole migration process.

Another system, proposed by Yu et al (2006), allows application mobility supported by mobile agents. The applications are context-aware, can adapt to the environment according to context and can follow the mobile user. The authors use a middleware called MDAgent on wireless handheld devices, which has four main components: managers for agent, context, rules and configuration respectively. The *agent manager* keeps record of the agent life cycles and takes snapshots of applications. The *context manager* has interpreters, filters, reasoning, and prediction capabilities; ontologies are used to control context. The *rule manager* contains context inference rules, while the *configuration manager* handles configuring the application to fit with different context. Applications are then designed with two levels in mind: a meta-level, including XML-based interface descriptions etc for adaptation, and a base-level, containing context-free functions (such as stop() and play()-functions, taking a media player as an example). A prototype was created, and a notepad-application and a media player application were sent between a 1.4GHz PC and a 1.7GHz RAM Laptop, with total migration times of between 500 and 4000 ms. Using a middleware, the system proposed by Yu et al becomes more loosely coupled compared to for instance the framework designed by Ranganathan, Shankar and Campbell above.

Zhou et al (2007) present a middleware supporting agent-based application mobility in pervasive environments. Their architecture consists of four layers: A sensor layer,

collecting the raw data; the context layer, classifying and storing this data; an agent layer, consisting of two agents (an autonomous and a mobile ditto) working with decision-making and migration of the application respectively; and finally the application layer, containing the application potentially being transferred. Keeping track of the applications is possible due to a resource registry and an application registry, where the devices and applications are listed along with descriptions of their properties. The chosen context representation form is the ontology, and the markup language used is OWL. OWL is a standard format of the Semantic Web (W3C, 2004b), used by applications to process information instead of just presenting it to humans. OWL adds logic to the mere syntax offered by XML. To demonstrate their architecture, Zhou et al (2007) exhibits an implementation of a prototype, its design resting upon the ideas put forth. Six demo applications (spanning from a media player to a follow me instant messenger) were migrated between stationary computers connected with 10MBps Ethernet, thus not mobile devices. The experiments show that suspension and migration times are rather stable, while resumption times increase remarkably according to size of the files migrated. Total migration time for files between 2MB and 7.5MB varies between 1000 ms and 1200 ms.

Schmidt and Hauck (2007) use a middleware called SAMProc (*self-adaptive mobile processes*) to dynamically migrate applications, allowing them to adapt their interfaces, states and implementations while running. Their proposed and partly implemented architecture is built around web services (as described by W3C, 2004a) and *web facets* (“*a particular configuration of state, functionality and implementation*”), with a *factory service*, dynamically loading code on demand, a *factory service finder*, using UDDI-like repository functionality along with a *state store service*, and also a dedicated *context service* collecting data about the current state of the runtime environment. Changes can for example trigger the application to move from one device to another. When initiating a move of a mobile web service, a manager service first stores the current state using the state store service, and then it finds a suitable factory, creating the web service at a new location using the stored states. As a last step, the original web service is removed (Schmidt, Kapitza & Hauck, 2007). A description language for SAMProcs is in development, as is an implementation of the context service (Schmidt & Hauck, 2007).

	<b>Main architecture</b>	<b>Key components</b>	<b>Registration/ Identification</b>	<b>Context representation</b>
Bandelloni & Paternò (2004)	Server-based	Migration clients and server	XML-based descriptions	XML
Siu et al (2004)	Facets	Facet, mobility, context state and data state managers	XML-based descriptions	XML
Ranganathan et al (2005)	Middleware	Adaptation, migration and presence services	Globally unique identifiers (CORBA IOR)	Ontologies
Satoh (2005)	Federation	Java objects with hooks	Host-based	Text data
Hwang et al (2006)	Server-based	Migration agent; linking of application specific libraries	Global ID, shared library	Database entries
Yu et al (2006)	Middleware	Agent, context, rule and config managers	XML-based descriptions	XML
Zhou et al (2007)	Middleware	Sensor layer, context layer, agent layer, application layer	Resource and application registry	Ontologies/OWL
Schmidt & Hauck (2007)	Middleware	Web services/ web facets	UDDI-like repository	XML-based and enhanced BPEL

*Table 1: Context-awareness supported application mobility systems overview*

As shown, over the last ten years context-awareness supported application mobility systems have gone from continuous media applications (sometimes nomadic in its mobility nature), residing heavily upon location as context notion, over systems with greater focus on interface adaptation, to more general and robust frameworks, in regards to functionality and interoperability. Ontologies as context representation are common in the more recent systems, as is a design comprising of modules each responsible for different tasks within the context-awareness supported application mobility framework, sometimes built into a middleware that can provide services such as migration planning and execution, as well as context reasoning and storage. (See also Table 1: Context-awareness supported application mobility systems overview, above.)

### 3 Identified challenges

Bu et al (2006) state that many context aware applications have been designed over the last decade, but few of them have actually been deployed in real life. There are indeed many challenges which must be overcome when trying to create context-awareness supported application mobility. In my survey, I have identified five areas that stand out, namely application identification, design of the context-awareness model, context quality, and handling seamlessness and heterogeneity respectively.

#### 3.1 Application identification

Security is an important matter when dealing with digital data. When defining security, the notion of Triple-A is often used, referring to authentication, authorization, and accounting.

Methods for authorization are needed when deciding if a user is allowed to access an application or the services it makes available. How to support privacy in context-aware systems and applications is clearly a public research issue (Du & Wang, 2008). Bharat and Cardelli (1997) raise issues of *privacy*, how to keep ownership of the applications and lists of available programs (as a user, you might not want to show everyone which applications you use and when), and *secrecy*, keeping parts of the applications and application data hidden for some users, while allowing others to access the whole application. Reponen, Huuskonen and Mihalic (2008) also point out the hardships of controlling the results of applications creating secondary context (as defined in section 1.2 in this survey), producing mobile content that might be accessible long after the actual application has stopped running. A good example is recorded video, which can be gathered and used in the primary context of a user or application, but then stored and thus becoming secondary context, accessible to other applications or users.

If a context-aware system is to be used by several or even many users, one must also consider how to handle accounting issues. If many users are to share limited resources, control has to be built into the system, deciding who can use a resource, when, how much and how often.

But before even thinking of solving these problems, one has to consider how to authenticate the devices, the users and, of course, the applications. There are several methods and standards for this. If the application is migrated via TCP, TCP headers can be used to map certain types of applications to registered port numbers. Many applications do however use dynamic port negotiation, and (naturally) they are also TCP dependent (Bernaille, Teixeira & Salamatian, 2006). Port numbers as such are therefore an inadequate solution. Identification of mobile applications is instead often carried out on the application layer. With SIP (IETF, 2002), an application layer protocol, one can establish real-time sessions and handle session migration. Applications can be connected to devices or users through non-IP identifiers, such as user@realm. SIP does however lack continuous seamless mobility support, firewalls may cause troubles in certain extent, and privacy protection cannot be guaranteed, as SIP messages might be caught and read (Oberle, Wahl & Sitek, 2007). Without means of keeping track of the moving applications, one would end up with disorder and have a hard time managing the mobile system.

When dealing with service mobility, service discovery protocols (SDP:s) are used to locate facilities in a network (Mian, Baldoni & Beraldi, 2009). Service discovery is made possible through service descriptions, abstractions of the available services' characteristics. These descriptions are often written in XML or in XML based languages, such as DARPA or OWL. Service descriptions can be stored and distributed in a number of ways: in the nodes' local cache, on every node in the network or on certain chosen nodes (directory nodes). For maximum mobility support, it is best to store descriptions on every node, but when it comes to large networks, a directory-based registration is more suitable. Similar approaches are applicable when identifying mobile applications. Among projects that use different kinds of global identifications are Ranganathan et al (2005), Hwang et al (2006) and Schmidt and Hauck (2007).

Nakajima et al (1999) see application identification as one of three key challenges when trying to achieve context-awareness supported application mobility. Making applications recognize the devices in the active space, as well as recognizing other active spaces, users, other persons and other applications are important challenges and deserve to be pointed out.

Future work will contain proposals for security solutions in the realm of context-awareness supported application mobility. However, I will not delve more deeply into this area in my survey.

### 3.2 Designing the context-awareness model

Du and Wang (2008) have looked into context-aware application programming for mobile devices, and they point out a central issue in making the application context aware. Context *acquisition*, *representation*, *interpretation* and *abstraction* are main areas that have to be regarded when designing the context-awareness model. Acquisition is often narrowed down to a question of what that data can be collected from the available sensors, making it a bottom-up approach, limiting the designers of the applications. Jacob et al (2006) show a more advanced example, using a request driven bio-inspired context gathering method, using terms such as pheromone density to weight the importance of context and in deciding how to spread it. A facet of acquisition is *aggregation* (Dey, 2001) collecting contextual data from different and varied sources. Representation and abstraction is much about how to present context data, within the system and to the user. These areas are together with interpretation (the transformation of raw data into basic data for decision-making) closely connected to the area of context quality, which is discussed in section 3.3.

Dey (2001) also recognizes acquisition and interpretation, but adds *storage* and *distribution*. The question of whether the architecture should be centralized to a context server, or if context information is to be distributed throughout several or all devices and applications in the system, is an important one. When designing the context-awareness model. Pretty much the same challenges are illustrated by Anagnostopoulos, Tsounis and Hadjiefthymiades (2007). They list a set of design challenges that developers of context-aware applications must confront. Except from context acquisition, representation, interpretation, abstraction, aggregation, storage and distribution, already mentioned above, the authors add context consistency and query. *Context consistency* deals with dynamically changing the distributed context models. Enabling *context query* gives the user an opportunity to question the context-aware system in a transparent way.

Where to store and how to access context seems to be a big challenge. Many different suggestions have been put forth, all with different pros and cons. One question is whether to put everything on the devices that will host the migration applications, or if server involvement is desirable. Adding a server, or even distributing context data and rules throughout a network, might be necessary in many cases, leading to potential difficulties with computational limitations, network restrictions etc (Du & Wang, 2008).

Another contiguous question is whether the context-awareness should be shaped around a specific application or scenario, or if it rather should be added through a middleware. The latter would make it possible to use the context-awareness model for a wider range of applications and scenarios, but currently most middleware platforms are still what da Rocha, Endler and de Siqueira (2008) call *context-aware islands*, independent and isolated from each other.

da Rocha, Endler and de Siqueira (2008) identify twelve requirements that a middleware for mobile context-aware environments should meet. First, it must instantly update and distribute context information to the client (be it an application or a device), dynamically aggregate context and be able to discover and handle new services and changes in the context environment. Secondly, it must be able to add new context types, for example if a new kind of sensor is added to the system. Third, closely linked to the former requirements, it must be able to discover the new context types, for example when the client has moved into a new environment. Further, it must handle different scopes of context perception, knowing which clients that are allowed access to context information and which are not. To grant access, the middleware has to contain mechanisms and policies for accessing in an heterogeneous environment, and also be able to handle abstractions of usage, for example only allowing users close by to access, 'close by' being an abstraction configured in the context-aware middleware. The next requirement has to do with management of application dynamic loading, registering which services are available in a certain environment and downloading them when a user enters that particular environment. Another important aspect is to know who or what having the interest in the context, in other words: who decides what is relevant. This might be the user, the application or a combination of the two. The last four requirements have to do with independence: architectural (platform and hardware) independence, decoupling

between context management and inference mechanisms, easy deployment in heterogeneous environments and tools for adapting to the most appropriate context. One of the writers' conclusions is that, currently, no middleware for context-awareness meet all these requirements.

Challenges in the design of the context-awareness model can also be seen from the user's perspective. Häkkinen and Mäntyjärvi (2006) list ten designer guidelines which give hints about the difficulties that be. They stress the importance of considering uncertainties in decision-making, whether and in that case when the user should be alerted and maybe asked to confirm a reaction from the context-aware system, or if the system should be able to autonomously decide what to do and when. Personalization is a keyword, as is visibility of the system status (also stressed by Dourish, 2004), making it easier for the user to understand the underlying rationale of the context-aware system. However, it is also important not to overflow the user with information. Häkkinen and Mäntyjärvi (2006) continue by emphasizing the value of securing the user's privacy, and also the user's control of the mobile device and access to the context (maybe letting the user edit context attributes or in a direct way shape the context model). Moreover, the system has to minimize (and preferably prevent) any interruptions in the services running through the mobile applications, even while moving – remembering the mobility is important. This might however collide with the proposed design criterion of giving control to the user. Finally, an important guideline is to always have the utility value of the information in mind; is the context really relevant? Else, it should not even be considered context, following the definitions presented in section 1.2.

### **3.3 Context quality**

Achieving quality in the context is a major challenge (Bu et al, 2006). Different sources of input might produce different kinds of data values, which will lead to inconsistency. The rate of which input is gathered is also crucial, along with a sense of when a certain input has been collected. Anagnostopoulos, Tsounis and Hadjiefthymiades (2007) stress resolution, accuracy, repeatability, frequency and staleness of context as quality indicators.

Mowafi and Zhang (2007) point out incommensurability between the common sense used by humans and reasoning done by many context-aware computer systems. Building a system that reflects human abilities as instinct and intuition seems to be impossible, yet a good system should have an integration of implicit means and also take input from the user; a system relying only of sensor data is not enough, the authors claim. To achieve good quality of context, the system must also be able to capture contextual information added (and abstracted) by the users themselves.

Another challenge, connected with both the context-awareness model and context quality is *context matching* (Anderson, Hansen & Bouvin, 2006). When a certain context is defined, a user or an application might return to that context several times. There must be a way to match parameters – physical, digital and conceptual – to the context. Several options are available, for example requiring the input to correspond exactly to the context stored in the context model, be it a repository or other means of set context representations. Sometimes it might be more appropriate to allow the input to differ a bit from the exact context, and still be counted as a certain set context. The actual matching can be carried out by calculating overlap between two sets of pre-processed or raw data input. Another way is to create hierarchies of context and calculating the depth  $n$  of the child context compared to its ancestor's level in the context hierarchy; a third way to define thresholds or ranges, and then comparing input to these.

Lastly, if a context-aware system is to be used by several or even many users, sharing contexts, the designer is faced with the challenge of modelling a large amount of users' personal requirements, and figure out how the system will handle these in a way that satisfies all users (Du & Wang, 2008).

### **3.4 Seamlessness**

All downtime in application mobility must be minimized, or preferably eliminated, to allow a good usage of the system and its services (Zhang & Pande, 2006). As migration is often associated with big overheads, this is very hard to achieve. The authors present pre-serialization as a mean of tackling the problem, putting the system developer up against

questions such as which parts of the application state should be migrated, which objects should be pre-serialized, how to solve this technically, and how to minimize downtime. But even though the downtime goes down with pre-serialization, this might not be enough. The topic of seamlessness must also be seen from a user perspective I believe, as seamlessness might be experienced different by different users in different context. A user might for example be more tolerant with downtime in a video stream, as she still has to move her eyes from one device to another, but when listening to music, an interruption in a song might be experienced differently and with less patience. This is also what two ITU-T Study Groups (ITU, 2007) state in their definition of Quality of Experience, taking into account “*the complete end-to-end system effects (client, terminal, network, services infrastructure, etc)*” and the fact that “*overall acceptability may be influenced by user expectations and context*”. I have found no thorough studies on subjective user experience of seamlessness in application migration.

Apparently, there are difficulties connected to the qualitative dimensions of seamlessness. The same goes however for quantitative measurements. As Zhou et al (2007) show, the time consumption of suspension and resumption are easy to calculate as these two parts of the migration process occur on the original platform and the destination platform respectively. The middle part, the actual migration, is however carried out between different platforms, which might have clocks that are not synchronized. Zhou et al tries to work around the problem through using round trip migration time cost, letting the application move back and forth between two platforms in a full cycle, thus eliminating asynchronization errors. Time differences between the different hosts cancel each other out, as shown in the formula:

$$T_2@H2 - T_1@H1 + T_4@H1 - T_3@H2 = T_2@H2 - T_1@H2 + T_4@H1 - T_3@H1$$

where  $T_i@H_j$  is the time value (T) at the moment of i, at host platform or device (H), with id j. Figure 1 gives a graphical overview of how this works.

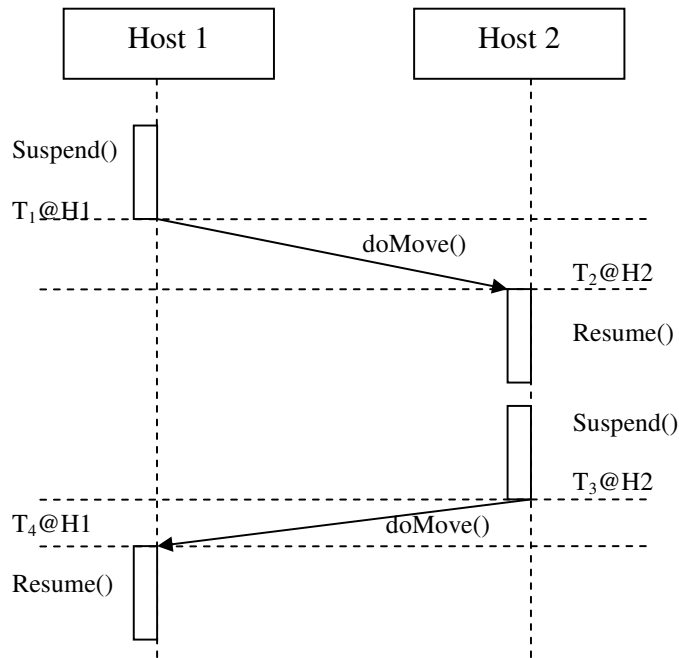


Fig. 1: Time Cost Illustration (Zhou et al, 2007)

### 3.5 Heterogeneity

When moving an application within or between systems with devices that differ in hardware or software configurations, migration is said to be heterogeneous (Geoffray, Thomas & Folliot, 2006). For such an application mobility system to function, it has to be designed for diverse environments, with special importance regarding user interfaces. The system state must always be interpreted with regards to available resources (such as network capabilities) and local preferences. Mobile devices are often associated with weaker performance capabilities than desktop computers; screen size, processing power, battery consumption etc is often limited when it comes to mobile devices.

Bharat and Cardelli (1997) address application adaptation by using an abstract representation of the user interface (namely a FormsVBT template, being a simple language for mediating user interface layout and interpreting events) during the migration process, which then can be translated by components at the migration destination. A

disadvantage with this method, as I see it, is that all communication between devices, including decisions about adaptation, must be done beforehand to secure a possible migration. Seamlessness might be harder to reach with this approach. To avoid this, a solution would be to have pre-installed device specific applications, and only migrate information and data state (David et al, 2007), but then the system would no longer be heterogeneous from a software perspective. If some of the devices should be public terminals or added to the system by other users from outside, this method is obviously insufficient.

Mitrovic and Mena (2002) instead use mobile agents to adapt the applications to heterogeneous environments. An agent is typically goal oriented, autonomous, communicative, adaptive, persistent, and reactive and can also stop its own execution. Through using an agent, one can add intelligence to the component mediating the application preferences, allowing it to make autonomous decisions concerning how to adapt the application to the new environment. Another big advantage is that the information required to adapt might be gathered during the agent's travel through the network. To carry out the adaptations and render user interfaces, the system makes use of jXUL, interpreting XUL descriptions and translation it into Java Swing interfaces. jXUL does however only render to Java Swing. Other similar programming language specific solutions are presented by Bandelloni and Paternò (2004) and David et al (2007) who list Java, Microsoft Intermediate Language and LLVA as enablers of migration in heterogeneous environments. David et al (2007) also decouple GUI adaptation from other adaptations, and do pre- and post-migration processing to minimize downtime. They use a Java-based system called JADE, which consequently means that all applications have to implement the Java interface needed for JADE.

To cope with heterogeneity and also take seamlessness ambitions into account, it would be desirable to have devices communicate with each other as autonomous as possible and also in a standardized way. Schmidt and Hauck (2007) use standardized protocols to achieve interoperability between different environments; web services allow XML-based application-to-application communication, independent of vendors, platforms and programming languages. A context service collects data about the runtime environment and takes autonomous decisions about possible migrations. A slightly more

advanced approach is presented by van Tonder and Wesson (2008), who split adaptation into three parts: information adaptation, visualisation adaptation and interface adaptation, modifying each part of the mobile application separately. However, their system relies on an existing knowledge base, already containing user preferences.

The most promising application adaptation methods can maybe be found within those relying on ontologies. Ranganathan, Shankar and Campbell (2005) demonstrate a system where every entity has an OWL file connected to it, describing its properties. The OWL files contain information about which tasks a component can perform, which (classes of) devices it can host and what data-formats it can read. Much understanding is therefore already embedded within the OWL language, making translation and communication more accurate. It also allows the creation and interpretation of hierarchies, where child entities inherit the properties of their parent entities. The system contains an existing knowledge base, but interworking with the OWL files knit to the individual entities. Clearly, standardized protocols form an important part of functioning context-awareness supported application mobility systems. The ongoing works in projects concerning development of the semantic web, OWL and other areas should play a big role in this.

#### **4 The use of context-awareness supported application mobility**

Through application-level mobility, supported by context-awareness, the user can interact with the environment in a potentially natural and comfortable way (Zhou et al, 2007). Bharat and Cardelli (1997) make use of four different metaphors, describing what application mobility can be used for. The first metaphor is when the application follows the user across several different physical locations, becoming *ubiquitous*. It can also take the role as *electronic secretary*, going from person to person, serving the users in turn. Another metaphor is the *interactive agent*, when the migratory application is working with others on the user's behalf, carrying out the user's agenda. It can also take the form of an *interactive message*, being sent from user to user, who in turn can interact with the application and edit the message, before passing it on.

This can of course be vastly enhanced by adding context-awareness. David et al (2007) describes a number of scenarios, where the gains of context-awareness supported application mobility are highlighted. For example, when using your PC via a remote desktop on your laptop, you need two computers running, a network connection for both devices (even though the applications as such might not require access to the internet) and two applications. If the application instead could move from your PC to your (or the system's) chosen device, you would not be dependent on other computers running or constant network connections. If the application also could adjust itself to the new device, regarding interface, battery consumption etc, it would be even better. All this can be realized through context-awareness supported application mobility. Moreover, a device might be better suited to cope with displaying certain kinds of data compared to a remote desktop – picture yourself watching a video stream on your plasma screen at home instead of via a remote desktop on your laptop. If you then, to continue the scenario, receive a phone call listed as important (derived from your personal context), the movie could be automatically paused and the call let through. These are only some of the scenarios which can be made possible if overcoming the challenges connected with context-awareness supported application mobility.

The late Mark Weiser's (1993, 1994) ideas about the everyday computer systems of the future, with ubiquitous but at the same time always accessible applications, are about to be realized. In that, context-awareness supported application mobility will play an important, not to say crucial, role.

## **5 Summary**

The purpose of this survey was twofold: to compile previous work on context-awareness supported application mobility and to identify challenges connected to the area. The challenges that was identified concern how to identify and keep track of applications that move between devices; how to best design the model for context-awareness; how to attain and communicate a good enough context quality; how to achieve seamlessness and minimize downtime while executing a migration; and finally how to handle heterogeneity within and between devices, in shape of differences in hardware and software.

## Acknowledgement

This survey has been compiled as part of the Mobile and Open Service Access project (MOSA, 2008), which is supported by EU structural funds (Objective 2).

## References

- Ahn, Y. 2008. Design of a Mobile Object Data Management Framework for Location-Enhanced Applications. In *Proceedings of the 2008 international Conference on Convergence and Hybrid information Technology - Volume 00* (August 28 - 29, 2008). ICHIT. IEEE Computer Society, Washington, DC, 270-273.
- Ahson, S. & Ilyas, M. (eds.) (2008). *RFID handbook: applications, technology, security, and privacy*. Boca Raton: CRC Press.
- Anagnostopoulos, C. B., Tsounis, A. & Hadjiefthymiades, S. (2007). Context awareness in mobile computing environments. In *Wireless Personal Communications*, 42(3):445–464, 2007.
- Anderson, K. M., Hansen, F. A., & Bouvin, N. O. (2006). Templates and queries in contextual hypermedia. In *Proceedings of the Seventeenth Conference on Hypertext and Hypermedia* (Odense, Denmark, August 22 - 25, 2006). HYPERTEXT '06. ACM, New York, NY, 99-110.
- Bandelloni, R. & Paternò, F. (2004). Flexible interface migration. In *Proceedings of the 9th international Conference on intelligent User interfaces* (Funchal, Madeira, Portugal, January 13 - 16, 2004). IUI '04. ACM, New York, NY, 148-155.
- Baratto, R. A., Potter, S., Su, G. & Nieh, J. (2004). MobiDesk: mobile virtual desktop computing. In *Proceedings of the 10th Annual international Conference on Mobile Computing and Networking* (Philadelphia, PA, USA, September 26 - October 01, 2004). MobiCom '04. ACM, New York, NY, 1-15.
- Barnard, L., Yi, J. S., Jacko, J. A. & Sears, A. (2007). Capturing the effects of context on human performance in mobile computing systems. In *Personal Ubiquitous Comput.* 11, 2 (Jan. 2007), 81-96.
- Bernaille, L., Teixeira, R. & Salamatian, K. (2006). Early application identification. In *Proceedings of the 2006 ACM CoNEXT Conference* (Lisboa, Portugal, December 04 - 07, 2006). CoNEXT '06. ACM, New York, NY, 1–12.
- Bharat, K. A. & Cardelli, L. (1995). Migratory applications. In *Proceedings of the 8th Annual ACM Symposium on User interface and Software Technology* (Pittsburgh, Pennsylvania, United States, November 15 - 17, 1995). UIST '95. ACM, New York, NY, 132-142.
- Bharat, K. A. & Cardelli, L. (1997). Migratory applications. In *Mobile object systems: towards the programmable Internet*. Second International Workshop, MOS '96, Linz, Austria, July 8-9, 1996:

- selected presentations and invited papers; Jan Vitek & Christian Tschudin (eds.). Berlin: Springer, 131-148.
- Bishop, G. & Welch, G. (2001). *An introduction to the Kalman filter*. SIGGRAPH, Course 8, 2001.
- Brynskov, M., Kristensen, J. F., Thomsen, B. & Thomsen, L. L. (2003). *What is context?* Technical report, Department of Computer Science, University of Aarhus, 2003. URL: <http://www.daimi.au.dk/~brynskov/publications/what-is-context-brynskov-et-al-2003.pdf>.
- Bu, Y., Gu, T., Tao, X., Li, J., Chen, S., & Lu, J. (2006). Managing Quality of Context in Pervasive Computing. In *Proceedings of the Sixth international Conference on Quality Software* (October 27 - 28, 2006). QSIC. IEEE Computer Society, Washington, DC, 193-200.
- Cabri, G., Leonardi, L. & Quitadamo, R. (2006). Enabling Java mobile computing on the IBM Jikes research virtual machine. In *Proceedings of the 4th international Symposium on Principles and Practice of Programming in Java* (Mannheim, Germany, August 30 - September 01, 2006). PPPJ '06, vol. 178. ACM, New York, NY, 62-71.
- Castells, M. (1996/2001). *Informationsåldern: ekonomi, samhälle och kultur. Bd 1, Nätverkssamhällets framväxt*. 2., rev. och utvidgade uppl. Göteborg: Daidalos.
- Chen, G. & Kotz, D. (2000) *A Survey of Context-Aware Mobile Computing Research*. Technical Report. UMI Order Number: TR2000-381., Dartmouth College.
- Christopoulou, E., Goumopoulos, C. & Kameas, A. (2005). An ontology-based context management and reasoning process for UbiComp applications. In *Proceedings of the 2005 Joint Conference on Smart Objects and Ambient intelligence: innovative Context-Aware Services: Usages and Technologies* (Grenoble, France, October 12 - 14, 2005). sOc-EUSAI '05, vol. 121. ACM, New York, NY, 265-270.
- Coulton, P., Rashid, O. & Edwards, R. (2008). RFID and NFC on Mobile Phones. In *RFID handbook: applications, technology, security, and privacy*. Ahson, S. & Ilyas, M. (eds.). Boca Raton: CRC Press.
- David, F. M., Donkervoet, B., Carlyle, J. C., Chan, E. M. & Campbell, R. H. (2007). Supporting Adaptive Application Mobility. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*. Berlin: Springer, 896-905.
- Devaraju, A., Hoh, S., & Hartley, M. (2007). A context gathering framework for context-aware mobile solutions. In *Proceedings of the 4th international Conference on Mobile Technology, Applications, and Systems and the 1st international Symposium on Computer Human interaction in Mobile Technology* (Singapore, September 10 - 12, 2007). Mobility '07. ACM, New York, NY, 39-46.
- Du, C., Sun, X.-H., & Chanchio, K. (2003). HPCM: A Pre-Compiler Aided Middleware for the Mobility of Legacy Code. In *Proc. IEEE Cluster Computing Conf.*, Dec. 2003.
- Du, W. & Wang, L. (2008). Context-aware application programming for mobile devices. In *Proceedings of the 2008 C<sup>3</sup>S<sup>2</sup>E Conference* (Montreal, Quebec, Canada, May 12 - 13, 2008). C3S2E '08, vol. 290. ACM, New York, NY, 215-227.

- Dey, A. K. & Abowd, G. D. (1999). *Towards a better understanding of context and context-awareness*. GVU Technical Report GIT-GVU-99-22, College of Computing, Georgia Institute of Technology, 1999.
- Dey, A. K. (2001). Understanding and Using Context. In *Personal Ubiquitous Comput.* 5, 1 (Jan. 2001), 4-7.
- Dourish, P. (2004). What we talk about when we talk about context. *Personal Ubiquitous Comput.* 8, 1 (Feb. 2004), 19-30.
- Fuggetta, A., Picco, G. P. & Vigna, G. (1998). Understanding Code Mobility. In *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 342–361, 1998.
- Gellersen, H. W., Schmidt, A. & Beigl, M. (2002). Multi-sensor context-awareness in mobile devices and smart artifacts. *Mob. Netw. Appl.* 7, 5 (Oct. 2002), 341-351.
- Geoffroy, N., Thomas, G. & Folliot, B. (2006). Live and Heterogeneous Migration of Execution Environments. In *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*. Berlin, Springer, 1254-1263.
- Grimm, R., Davis, J., Lemar, E., Macbeth, A., Swanson, S., Anderson, T., Bershad, B., Borriello, G., Gribble, S. & Wetherall, D. (2004). System support for pervasive applications. *ACM Trans. Comput. Syst.* 22, 4 (Nov. 2004), 421-486.
- Hinckley, K., Pierce, J., Sinclair, M., and Horvitz, E. 2000. Sensing techniques for mobile interaction. In *Proceedings of the 13th Annual ACM Symposium on User interface Software and Technology* (San Diego, California, United States, November 06 - 08, 2000). UIST '00. ACM, New York, NY, 91-100.
- Hwang, T., Park, H. & Chung, J. W. (2006). Desktop Migration System Based on Dynamic Linking of Application Specific Libraries. In *Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference*. Volume 3, 20-22 Feb. 2006, 1586 – 1588.
- Häkkinen, J. & Mäntyjärvi, J. (2006). Developing design guidelines for context-aware mobile applications. In *Proceedings of the 3rd international Conference on Mobile Technology, Applications & Systems* (Bangkok, Thailand, October 25 - 27, 2006). *Mobility '06*, vol. 270. ACM, New York, NY, 24.
- IETF. (2002). *SIP: Session Initiation Protocol*. URL: <http://www.ietf.org/rfc/rfc3261.txt>
- ITU. (2002). *ITU-T Recommendation H.510, Mobility and Collaboration procedures*. Mobility for H-Series multimedia systems and services. URL: [http://www.itu.int/rec/dologin\\_pub.asp?lang=e&id=T-REC-H.510-200203-I!!PDF-E&type=items](http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-H.510-200203-I!!PDF-E&type=items)
- ITU. (2007). *Definition of Quality of Experience (QoE)*, International Telecommunication Union, Liaison Statement, Ref.: TD 109rev2 (PLEN/12), Jan. 2007. URL: <http://www.itu.int/md/T05-FG.IPTV-IL-0050/en>

- ITU. (2008). *International Telecommunication Union*. URL: <http://www.itu.int/net/home/index.aspx> (read 2009-02-09)
- Jacob, C., Linner, D., Steglich, S. & Radosch, I. (2006). Bio-inspired context gathering in loosely coupled computing environments. In *Proceedings of the 1st international Conference on Bio inspired Models of Network, information and Computing Systems* (Cavalese, Italy, December 11 - 13, 2006). BIONETICS '06, vol. 275. ACM, New York, NY, 15.
- Koponen, T., Gurtov, A. & Nikander, P. (2005). *Application Mobility with HIP*. URL: <http://infracorp.hiit.fi/papers/appmob.pdf>. Also in Proc. of ICT'05, May 2005.
- Kummerfeld, B., Quigley, A., Johnson, C. & Hexel, R. (2003). "Merino: Towards an intelligent environment architecture for multi-granularity context description", In *Proc. User Modeling for Ubiquitous Computing*, Johnstown, USA, 2003.
- Loke, S. (2007). *Context-aware pervasive systems: architectures for a new breed of applications*. Boca Raton, FL: Auerbach Publications.
- Löwgren, J. & Stolterman, E. (1998). *Design av informationsteknik: materialet utan egenskaper*. Lund: Studentlitteratur.
- Mendel, J. M. (1995). Fuzzy logic systems for engineering: A tutorial. In *Proc. IEEE*, vol. 83, Mar. 1995, 345–377.
- Mian, A. N., Baldoni, R. & Beraldi, R. (2009). A Survey of Service Discovery Protocols in Multihop Mobile Ad Hoc Networks. In *Pervasive Computing, IEEE*, vol.8, no.1, Jan.-March 2009, 66–74.
- Mikalsen, M., Paspallis, N., Floch, J., Stav, E., Papadopoulos, G. A., and Chimaris, A. 2006. Distributed context management in a mobility and adaptation enabling middleware (MADAM). In *Proceedings of the 2006 ACM Symposium on Applied Computing* (Dijon, France, April 23 - 27, 2006). SAC '06. ACM, New York, NY, 733-734.
- Milojičić, D. S., Douglis, F., Paindaveine, Y., Wheeler, R. & Zhou, S. (2000). Process migration. *ACM Comput. Surv.* 32, 3 (Sep. 2000), 241-299.
- Mitrovic, N. & Mena, E. (2002). Adaptive User Interface for Mobile Devices. In *Proceedings of the 9th international Workshop on interactive Systems. Design, Specification, and Verification* (June 12 - 14, 2002). P. Forbrig, Q. Limbourg, B. Urban, and J. Vanderdonckt, Eds. Lecture Notes. In Computer Science, vol. 2545. Springer-Verlag, London, 29-43.
- Moh, M., Ho, L., Walker, Z., & Moh, T-S. (2008). A Prototype on RFID and Sensor Networks for Elder Health Care. In *RFID handbook: applications, technology, security, and privacy*. Ahson, S. & Ilyas, M. (eds.). Boca Raton: CRC Press.
- A User-centered Approach to Context-awareness in Mobile Computing
- MOSA. (2008). *Mobile and Open Service Access*. URL: <http://www.mosaproject.org/> (read 2009-03-16)

- Mowafi, Y. & Zhang, D. (2007). A User-centered Approach to Context-awareness in Mobile Computing. In *Fourth Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services, 2007*. MobiQuitous 2007. 6-10 Aug. 2007, 1-3.
- Nakajima, T., Kone, M. T. & Aizu, H. (1999). System Support for Migratory Continuous Media. In *Proceedings of the 11 Ipps/Spdp'99 Workshops Held in Conjunction with the 13th international Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing* (April 12 - 16, 1999). J. D. Rolim et al, Eds. Lecture Notes In Computer Science, vol. 1586. Springer-Verlag, London, 430-441.
- Oberle, K., Wahl, S. & Sitek, A. (2007). Enhanced Methods for SIP based Session Mobility in a Converged Network. In *Mobile and Wireless Communications Summit, 2007. 16th IST*, 1-5 July 2007, Budapest, 1-5.
- OMG. (2009). *UML*. URL: <http://www.uml.org/> (read 2009-02-03)
- Pan, J., Huang, Z., Mao, G., & Dong, J. (2006). A Context Awareness Architecture for Mobile Learning Based on Fuzzy Petri Nets. In *Proceedings of the 16th international Conference on Artificial Reality and Telexistence--Workshops* (November 29 - December 01, 2006). ICAT. IEEE Computer Society, Washington, DC, 552-557.
- Phan, T., Guy, R., Gu, J. & Bagrodia, R. (2001). A New TWIST on Mobile Computing: Two-Way Interactive Session Transfer. In *Proceedings of the Second IEEE Workshop on internet Applications (Wiapp '01)* (July 23 - 24, 2001). WIAPP. IEEE Computer Society, Washington, DC, 2.
- Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *Proc. IEEE*, 77(2), 257-286.
- Ramkumar, B. & Strumpfen, V. (1997). "Portable Checkpointing for Heterogeneous Architectures", in *Proceedings of the 27<sup>th</sup> International Symposium on Fault-Tolerant Computing – Digest of Papers*, Seattle, WA, June 1997.
- Ranganathan, A., Shankar, C. & Campbell, R. (2005). Application polymorphism for autonomic ubiquitous computing. *Multiagent Grid Syst.* 1, 2 (Apr. 2005), 109-129.
- Reponen, E., Huuskonen, P. & Mihalic, K. (2008). Primary and secondary context in mobile video communication. *Personal Ubiquitous Comput.* 12, 4 (Apr. 2008), 281-288.
- da Rocha, R. C., Endler, M. & de Siqueira, T. S. (2008). Middleware for ubiquitous context-awareness. In *Proceedings of the 6th international Workshop on Middleware For Pervasive and Ad-Hoc Computing* (Leuven, Belgium, December 01 - 05, 2008). MPAC '08. ACM, New York, NY, 43-48.
- Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H., & Nahrstedt, K. (2002). A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing* 1, 4 (Oct. 2002), 74-83.
- Satoh, I. (2004). Self-deployment for Distributed Applications. In *Scientific engineering of distributed Java applications: third international workshop* (N. Guelfi Ed.). Berlin, Springer, 48-57.

- Schilit, B., Adams, N. & Want, R. (1994). Context-Aware Computing Applications. In *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications - Volume 00* (December 08 - 09, 1994). WMCSA. IEEE Computer Society, Washington, DC, 85-90.
- Schmidt, H. & Hauck, F. J. (2007). SAMProc: middleware for self-adaptive mobile processes in heterogeneous ubiquitous environments. In *Proceedings of the 4th on Middleware Doctoral Symposium* (Newport Beach, California, November 26 - 30, 2007). MDS '07. ACM, New York, NY, 1-6.
- Schmidt, H., Kapitza, R. & Hauck, F. J. (2007). Mobile-process-based ubiquitous computing platform: a blueprint. In *Proceedings of the 1st Workshop on Middleware-Application interaction: in Conjunction with Euro-Sys 2007* (Lisbon, Portugal, March 20 - 20, 2007). MAI '07, vol. 224. ACM, New York, NY, 25-30.
- Schmohl, R. & Baumgarten, U. (2008a). Context-aware computing: a survey preparing a generalized approach. In *IMECS 2008: Proceedings of the International MultiConference of Engineers and Computer Scientists 2008*. International Association of Engineers, 2008.
- Schmohl, R. & Baumgarten, U. (2008b). A Generalized Context-aware Architecture in Heterogeneous Mobile Computing Environments. In *Proceedings of the 2008 the Fourth international Conference on Wireless and Mobile Communications - Volume 00* (July 27 - August 01, 2008). ICWMC. IEEE Computer Society, Washington, DC, 118-124.
- Siu, P. P. L., Belaramani, N., Wang C-L. & Lau, F. C. M. (2004). Context-Aware State Management for Ubiquitous Applications. In *Embedded and Ubiquitous Computing*. Yang, L. T., Guo, M, Gao, G. R, Jha, N. K. (Eds.). International Conference on Embedded and Ubiquitous Computing, Aizu, Japan, 26-28 August 2004. 776-785.
- Sousa, J. P. & Garlan, D. 2002. Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments. In *Proceedings of the IFIP 17th World Computer Congress - Tc2 Stream / 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance* (August 25 - 30, 2002). J. Bosch, W. M. Gentleman, C. Hofmeister, and J. Kuusela, Eds. IFIP Conference Proceedings, vol. 224. Kluwer B.V., Deventer, The Netherlands, 29-43.
- van Tonder, B. & Wesson, J. (2008). Using adaptive interfaces to improve mobile map-based visualisation. In *Proceedings of the 2008 Annual Research Conference of the South African institute of Computer Scientists and information Technologists on IT Research in Developing Countries: Riding the Wave of Technology* (Wilderness, South Africa, October 06 - 08, 2008). SAICSIT '08, vol. 338. ACM, New York, NY, 257-266.
- W3C. (2004a). *Web Services Architecture*. URL: <http://www.w3.org/TR/ws-arch/> (read 2009-01-21).

- W3C. (2004b). *Web-Ontology (WebOnt) Working Group*. URL: <http://www.w3.org/2001/sw/WebOnt/> (read 2009-02-20).
- Weiser, M. (1993). Hot Topics: Ubiquitous Computing. In *IEEE Computer*, October 1993, 71-72.
- Weiser, M. 1994. The world is not a desktop. *Interactions* 1, 1 (Jan. 1994), 7-8.
- Wickramasuriya, J., Mehrotra, S. & Venkatasubramanian, N. (2008). RFID-Enabled Privacy-Preserving Video Surveillance: A Case Study. In *RFID handbook: applications, technology, security, and privacy*. Ahson, S. & Ilyas, M. (eds.). Boca Raton: CRC Press.
- Yu, P., Cao, J., Wen, W. & Lu, J. (2006). Mobile Agent Enabled Application Mobility for Pervasive Computing. In *Ubiquitous Intelligence and Computing 2006, LNCS 4159*. J. Ma et al (Eds.). Springer, Berlin. 648-657.
- Zhang, K. & Pande, S. (2006). Minimizing downtime in seamless migrations of mobile applications. In *Proceedings of the 2006 ACM SIGPLAN/SIGBED Conference on Language, Compilers, and Tool Support For Embedded Systems* (Ottawa, Ontario, Canada, June 14 - 16, 2006). LCTES '06. ACM, New York, NY, 12-21.
- Zhou, Y., Cao, J., Raychoudhury, V., Siebert, J. & Lu, J. (2007). A Middleware Support for Agent-Based Application Mobility in Pervasive Environments. In *Proceedings of the 27th international Conference on Distributed Computing Systems Workshops* (June 22 - 29, 2007). ICDCSW. IEEE Computer Society, Washington, DC, 9.

