Electronic Communications of the EASST
Volume 28 (2010)

Proceedings of the Third International DisCoTec Workshop on Context-Aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS 2010)

Testing self-adaptive applications with simulation of context events

Konstantinos Kakousis, Nearchos Paspallis, George A. Papadopoulos, Pedro Antonio Ruiz

12 pages

Guest Editors: Sonia Ben Mokhtar, Romain Rouvoy, Michael Wagner
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Testing self-adaptive applications with simulation of context events

Konstantinos Kakousis[1], Nearchos Paspallis[1], George A. Papadopoulos[1], Pedro Antonio Ruiz[2]

Department of Computer Science, University of Cyprus[1]
Integrasys S.A., Esquillo, Spain[2]

Abstract: Modern trends in mobile computing have raised the expectations of users in terms of such features such as context-awareness and self-adaptiveness. With such capabilities, applications can autonomously sense their context and automate a number of tasks, effectively reducing the attention required by the end users. This paper presents a custom simulation engine, designed to support the testing of applications developed using the MUSIC platform. The simulation tool consists of a platform-independent server module, deployed along with the application, and a client module which is responsible for interpreting and executing the simulation script. The use of the tool is demonstrated in the scope of the SatMotion application, which is designed to assist satellite antenna installers with specialized functionality.

Keywords: Context-awareness, self-adaptation, simulation, context plug-ins, OSGi

1 Introduction

With recent advances in hardware and software for mobile devices, the users have grown to expect more from their smart-phones. In particular, with the widespread use of sensors and the pervasiveness of network connectivity, many applications featuring advanced context-awareness and self-adaptiveness are now possible. Nevertheless, the design, implementation and testing of such applications remains a work-in-progress, posing significant challenges to their developers.

This paper presents a custom *context simulation tool*, designed to support the testing of applications developed using the MUSIC platform. The latter provides modeling tools for design-time support, along with a middleware architecture used in run-time. The context simulation tool is used to verify the operation of the designed applications and consists of a platform-independent server module, deployed along with the application, and a client module which is responsible for interpreting and executing the simulation script. The use of the tool is demonstrated in the scope of the SatMotion application, which is designed to assist satellite antenna installers with specialized functionality.

The rest of this paper is organized as follows: Chapter 2 presents the foundations of the MUSIC platform, which is the one targeted by the proposed simulation tool. Next, chapter 3 presents the functionality and design of the context simulation tool, which is then demonstrated in the context of testing a real application—the SatMotion—in section 4. The paper then discusses related work in section 5 and closes with the conclusions in section 6.

2 The MUSIC platform

As part of a more general framework [RBD+09], the MUSIC context management middleware provides developers with both a methodology for designing context-aware applications, as well as a platform for deploying them [Pas09]. To enable context-aware, self-adaptive applications, the developers specify their applications in terms of *components* and *configuration plans*. The former are normal self-contained and reusable software entities, commonly termed as software components [Szy97]. The latter are customized artifacts used to define the composition possibilities for the application [FHS+06]. In this case, the composition variations are defined either in terms of configurable parameters, or in terms of interchangeable components [MSKC04].

While the composition plans are sufficient to define the adaptation domain of an application (i.e., the set of possible states it can be adapted to), additional adaptation reasoning mechanisms are needed for autonomously selecting the optimal adaptation as needed. In the case of the MUSIC platform, these mechanisms are based on the use of *utility functions* [AEP+07, PEHP09, KRW+09]. These provide a means for selecting an adaptation by assigning a *utility value* to each possible configuration, in a way that corresponds to their overall fitness to the context. In this way, it is possible for the middleware to continuously evaluate the context in runtime, and pick a configuration that optimizes utility, as measured by the utility functions [FHS+06].

An important component of the MUSIC middleware is the *context manager*, which is of particular interest to this paper. In MUSIC, context is dealt with as an individual aspect. In this regard, context clients—including the adaptation reasoning module that evaluates the utility functions—register with the context middleware for particular context types. In the used context model [RWK+08a, RWK+08b], the context types are disambiguated via two concepts: the *scope* and the *entity*. The former models the targeted context property (e.g., location, temperature, memory use, etc) and the latter defines the entity which is described by the property (e.g., my car, the room I am in, the device I am holding, etc). While the context needs of clients are explicitly identified, so does the context provided in the middleware. For the latter, pluggable mechanism is used which allows individual context plug-ins to be dynamically installed and deployed to provide the necessary context types (e.g., a location plug-in which uses an underlying GPS sensor, a temperature sensor plug-in that uses a web-service, etc) [Pas09].

The context middleware, as well as the whole MUSIC middleware, are built on top of the OSGi framework. The functionality of the MUSIC middleware architecture is illustrated in the SatMotion application, presented throughout the paper.

2.1 The SatMotion application

SatMotion, developed by Integrasys, is an application that runs on PDAs and assists installers during the process of alignment of satellite antennas that provide broadband access through bidirectional satellite communications. SatMotion allows the control and command of a measurement instrument, normally a spectrum analyzer, from a remote wireless handheld terminal in order to obtain trace information representing electromagnetic signals acquired by the instrument. Signal traces are retrieved from the measurement instrument by a software server module, then transmitted wirelessly and finally displayed in real-time on the screen of the remote terminal. The SatMotion runtime high-level architecture is illustrated in figure 1.

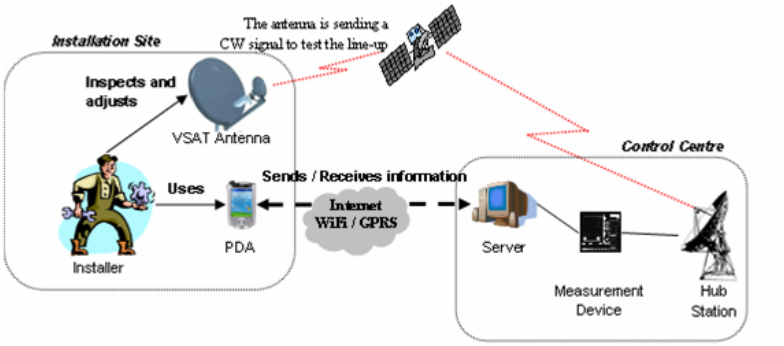

Figure 1: Using SatMotion to line-up a VSAT antenna Installation

SatMotion is the main application of a software suite called SatCom. Besides SatMotion, SatCom contains two complementary applications that ease the installer in his daily tasks: First, the *PlaInstallation* application is in charge of controlling the connection with the company server, as well as for authenticating and retrieving the task schedule. The task schedule contains route information, installation sites and specific tasks to do. Second, the *Adap2Nav* application provides navigation facilities to guide the user to the installation site, and provide different interfaces, such as Graphical navigation mode, Text mode or Voice mode.

2.2 Variability in the SatMotion application

The SatMotion trial is a mobile application whose usability can be affected by several factors, and therefore, can benefit from MUSIC. In order to maximize the usability, the application exhibits context-aware behavior that is manifested through the self-reconfiguration of the application. The selection of the optimal variant depends on the contextual information, as described in the potential adaptations below:

- The PlaInstallation application connects to a remote service provider in order to download the installation site/sites. The application uses the network interface available at the moment (Wi-Fi/GPRS). Depending on the economic cost of the attached network, the middleware decides whether to download just the necessary or all of the information. Therefore, the amount of downloaded data depends on the currently used network.
- When the installer accepts the tasks in her or his PDA, the platform automatically shows the optimal route on PDA (Adapt2Nav application). If a Car GPS is detected, the platform selects it as the optimal interface. Depending on the device type (i.e., PC, PDA or Car GPS) different versions of the user interface are presented.
- When the installer is aligning the antenna, several applications modes are automatically offered, depending on the context conditions. The network QoS is the most influential context parameter during the alignment process:
  - TwoWay communication: when there are plenty of network resources. The application receives the signal information from the server. Also, it sends commands to the

server to change the setup of the spectrum analyser and consequently the setup of the application.

  - OneWay: When the network does not provide enough throughput for a usable TwoWay mode, a unidirectional communication mode is selected. This application mode does not allow neither control nor command of the Spectrum analyzer.
  - Text mode: It only receives relevant data (Signal Noise Ratio and Cross Polarization value) when the network QoS is extremely low.
  - Adjust the trace resolution: The resolution of the received signal can also be adjusted to the quality of the network.
  - If the terminal provides signal traces through the satellite network, this service is selected as the optimal one.

- If several installers are using the server at the same time to align antennas, the SatMotion Server may not be able to serve all of them. The installer has to wait until a free slot is available. The installation processes with the lowest priority is interrupted until the resources become available. The priority of installation is given by the user profile and client profile.

The main contextual information relevant to the adaptation of the SatMotion application are:

- Application status: A sensor that monitors specific information of the application, such as the installation sites, alignment status, user profiles, etc.
- User preferred mode: The application provides an interface for the users to set their preferences.
- Network throughput: It is simulated with a sensor plugin.
- Network availability: WiFi, 3G, or None Available.
- Device memory: The available memory of the device for applications.
- Location: The GPS coordinates of the installer.

3 Context Simulation Tool

This section presents the mechanism developed for simulating context events within the MUSIC middleware. The implementation of this mechanism is provided as an independent application—not part of the core middleware—allowing the developers to use it only as needed.

This mechanism is directly interfaced with the context middleware. The middleware provides interfaces (in the form of OSGi services) that enable external components to:

- First, access information concerning the available and required context types (and optionally receive asynchronous notification upon their update);

- Second, use the *simulation mode* which allows for the interception of all context data while blocking a subset of it as needed; and

- Third, create and publish arbitrary context events containing the simulated context values.

The *simulation mode* is an option, that when set can intercept the flow of all context events from the providers (i.e., the context plug-ins) so that they do not reach the corresponding context clients. With this option, it is possible to have complete control of the middleware, so that context events (both real and simulated) are forwarded to the middleware and the context clients only as intended by the testers.

The context simulation tool is organized in a client-server architecture. The two sides coordinate via invocations of commands, as formalized by a predefined script (discussed in the next subsection).

The server side consists of the *scenario engine* which interprets and executes the commands, as defined in the script. The client side typically consists of a graphical frontend, that the tester can use to load, edit, and execute simulation scripts.

3.1 Scenario Engine Dialect

The scenario engine consists of a simple parser that recognizes a set of predefined middleware commands, and an execution engine that executes the commands on a local (or remote) MUSIC host. The main purpose of the scenario engine is to enable application and middleware developers to introspect the context middleware reaction to specific context events. The dialect supported by the scenario engine includes the following commands:

- **logtime** - Logs the current date and time. The logtime command has no arguments and returns the current date and time in milliseconds precision in the format "*yyyyy-MM-dd HH:mm:ss.SSS*".

- **sleep** int duration - Halts the execution of the script for a predefined number of milliseconds. The sleep command accepts a single argument of the integer type.

- **memory** - Lists the available and the total memory. The memory command has no arguments and returns the amount of free and total memory of the *Java Virtual Machine* (JVM) that the MUSIC node is deployed on (in Bytes).

- **contextEvent** String #entity String #scope String #representation Object[] values - Executes a simulated context event. This command generates a new context event as defined by its four arguments. The first argument specifies the entity of the context event to be generated, the second one its scope and the third argument its representation. The last argument is a list of values of any type, enclosed in brackets and separated by commas. An example is illustrated below.

  Example 1:

  ```
  contextEvent #...Entity.User|myself #...Scope.Location
  #...Representation.Location [#Latitude=30.1, #Longitude=31.2]
  ```

- **invocation** String bundleName String serviceName String methodName Object[] arguments - Invokes arbitrary services published by the installed OSGi bundles. The invocation method accepts four arguments and can be used for invoking just any method, of any service, provided by any resolved bundle. The first argument specifies the bundle's symbolic (full) name, the second argument the name of the selected provided service, the third argument the name of one of the desired method and finally the last argument defines the list of arguments that the defined method requires. The arguments must be enclosed in brackets, separated by commas. An example is illustrated below.

  Example 2:

  ```
  invocation org.istmusic.mw.gui.context.viewer TestInvocation
  testMethod [false,some_string,5.555,5]
  ```

The above list contains the commands that are currently supported by the scenario engine. However, this list can be easily extended to support additional commands, if needed.

As it was mentioned already, the scenario engine is accessed by a frontend graphical application. In the current implementation, this frontend is simply a subsystem of a wider *visualization tool*, used for monitoring and controlling many aspects of the MUSIC middleware. The simulation script can be loaded from an external text file, or it can be directly typed in a text area inside the frontend tool. After the simulation script has been loaded it can be executed. The output is also displayed in a text area of the frontend tool, and is also written to the middleware log. If needed the contents of the log area can be saved in a separate text file for further study and use.

In the *simulation mode* the context events are redirected to the visualization tool. The latter intercepts the events and does not let them back to the system for normal execution. Thus, only simulated events created in the simulator are delivered to the listeners. The interception mode is especially useful for testing certain aspects (i.e., context-aware or self-adaptive behavior) of the applications in isolation of the rest of the context events monitored.

3.2 Distributed simulation

An important feature of the MUSIC middleware is that it supports distributed applications which involve adaptable components from multiple nodes. In practice, applications might include components or services deployed in distributed nodes (which must also run an instance of the MUSIC middleware). In effect, all the nodes that contribute to an application form a so-called *adaptation domain* [AHPE07, AEP+07]. For this purpose, the simulation engine—and the corresponding script—were enhanced to also facilitate such distributed environments.

As argued already, the scenario engine is organized in two complementary components: the scenario engine client and the scenario engine server. The client can handle more than one server. The client sends commands (as parsed from the script) to the server for execution. The server executes the commands and returns a response which is received by the client. Communication is initiated by the client side only.

The support for distribution implies some changes in the script as well. For the two commands that is reasonable to expect distributed execution (i.e., *contextEvent* and *invocation*), the `[nodeX]` prefix is introduced. In principle, when a prefix is not indicated, then the command is

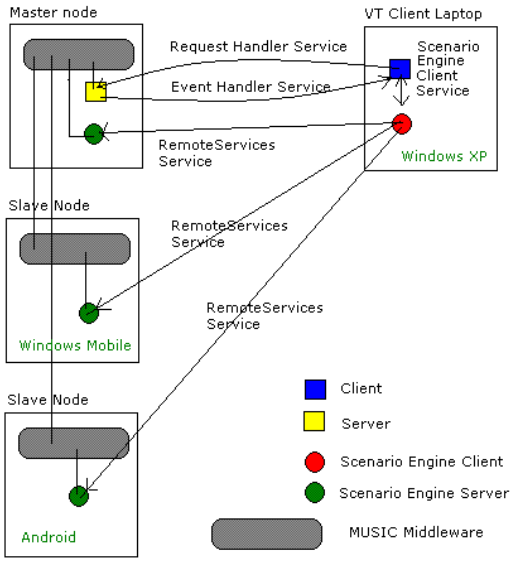

Figure 2: Overview of communications in distributed simulation

always executed locally. When the `[nodeX]` prefix is used (where `X` can be a integer), then the command is delegated to the corresponding node.

At this point, the only missing piece is which nodes do the `[nodeX]` prefixes correspond to. This is assigned dynamically right before the execution of the script, as shown below.

The first step is the discovery of the available MUSIC nodes and, consequently, the available instances of the scenario engine server. For this purpose, a service discovery agent is used to dynamically *discover* the MUSIC nodes. At a lower level, this is done using the *Service Location Protocol* (SLP). For example, in figure 2, the Master node and the two Slave nodes (i.e., the Windows Mobile and the Android devices) discover and are aware of each other.

The next step is the association of each logical name (i.e., indicated by `[nodeX]`) with actual nodes. It is possible during the launch of the script to check whether there are references to distributed nodes (e.g., to `[node1]` and to `[node2]`) and ask the user to manually associate the actual nodes with the corresponding logical names (e.g., the *Windows Mobile* and the *Android* devices respectively in the example in figure 2). This approach has the important advantage that the scripts remain agnostic to specific nodes (as indicated by their host names or IPs), and thus can be reused in different situations with various devices deploying the MUSIC middleware.

4 Demonstrating the Context Simulation Tool in SatMotion

This section presents a scenario, encoded using the proposed scripting language, which is used to evaluate the context-aware and self-adaptive behavior of the SatMotion application which was presented in subsections 2.1 and 2.2. In principle, the script dictates specific context changes which are expected to trigger certain adaptations by the middleware. These adaptations can be

eventually confirmed by examining the log messages.

The initial situation is that the PlaInstallation application is launched and the 3G connection is available. The script is as follows (comments appear as text after the "%" character):

1. `logtime % logs time`
2. `invocation satcom.plaInstall Authenticate authenticateClient ["login", "password"] % invocation of authentication method`
3. `sleep 10000 % sleeps 10 seconds`
4. `logtime`
5. `contextEvent #...Entity.Device|this #...Scope.WiFi.Signal #...Representation.Network.Signal [#Strength=0.95] % change % 3G to WiFi (by making a strong WiFi signal available)`
6. `memory % logs the available memory`
7. `sleep 10000 % sleeps another 10 seconds`
8. `invocation satcom.plaInstall TaskManager accept % invocation of method that signals tasks acceptance - once, % the tasks are downloaded and accepted, the navigation % application is automatically selected by the middleware`
9. `contextEvent #...Entity.Device|this #...Scope.Location #...Representation [#Longitude=35.1, #Latitude=33.4] % changes location`
10. `memory`

The script starts by printing out the time (step 1) and then proceeds by invoking the command `authenticateClient`, using as arguments the login and password strings (step 2). At the next step (3), the simulation *sleeps* for 10 seconds, and then it resumes with logging time again (step 4). The next step includes a context change that signifies transition from 3G to WiFi connection. This is modeled in terms of a context event that shows that the WiFi signal is now 95% strong (step 5). The script then proceeds with printing out details of the memory (step 6) and then sleeping again for another 10 seconds (step 7). Following that, the TaskManager is then explicitly asked to accept the assigned tasks through an appropriate invocation (step 8). At that point, a new context event is triggered, encoding that the location of the user has changed to a pair of designated longitude/latitude coordinates (step 9). The script ends with printing out details of the memory (step 10).

11. `logtime`
12. `[node1] invocation navigation NavigationService start`

13. `sleep 15000`
14. `logtime`
15. `[node1] invocation navigation NavigationService stop`

At this point, the distribution capabilities of the simulation engine are shown. At the next step (11), the time is logged and then the *NavigationService* is started on *node1* (step 12). This slave node offers the navigation service which allows the application to receive driving directions. By making this service available (simply by starting it), the client side of the application is adapted by binding to the service and start using it (the actual moment that this happens appears in the log of the adaptation engine). After a small delay of fifteen seconds (step 13) to allow the service to be started and discovered, and the adaptation engine to respond accordingly, the time is logged (step 14) and the navigation service is stopped (step 15). This eventually causes the service absence to be discovered again and the adaptation engine to respond accordingly (with all these event being recorded in the log).

5 Discussion and related work

Over the last decade much research work has been conducted on self-adaptive software. Several research groups and individuals have proposed middleware solutions for monitoring and reacting to context changes [SG02, Hen03, MPF+06, Pas09]. However, only a few works provide appropriate tools that aim to simplify the task of testing context-aware applications and frameworks. As already noted in [MSKC04], compositional adaptation is very powerful, but appropriate tools are needed to fully exploit its power. The MUSIC Studio aims to provide a suite of tools and methods that help MUSIC application developers creating applications based on the MUSIC middleware. The proposed simulation tool has been developed as part of this suite and allows to monitor and control context changes and visualize adaptation decisions and effects.

Similar to our work, Du and Wang [DW08] have proposed a programming model, an implementation framework and a development environment that facilitate the development of context-aware applications. The development environment provides a series of tools to simplify the development process. The environment is composed of three parts: program integrator, context change simulator and an interpreter library. The program integrator reduces the users effort on binding user-defined programs with the implementation framework, by automatically generating the binding code. The context change simulator provides testability for developed applications. The simulator can run in two different modes to support high-level application logic testing or implementation level testing. Finally, the extensible interpreter library reduces the effort on context conversion by providing a set of reusable and widely used context interpreters. When compared to our tool, Du and Wang's framework lack of support for simulating/testing system's behavior in the case of distributed adaptation.

In another work, Huebscher and McCan [HM04] have proposed a simulation framework for context-aware applications targeting developers who want to test their applications' context and autonomic logic prior to real-life deployment. Their framework runs on top of another simulation tool called TOSSIM [LLWC03], a simulator for TinyOS applications. The primary objective of

this work was to simulate the data produced from real sensors while in our case the focus is on simulating and monitoring the behavior of the applications and context middleware.

Finally, the Rainbow approach [GCH+04] provides a framework for monitoring and managing a target system throughout the adaptation cycle. Rainbow uses an external adaptation mechanism based on the utilization of architectural models at runtime. This enables system administrators to monitor a managed system and its runtime behavior and thus easier detect possible problems. Cheng *et al* [CGS06] have proposed a specialized adaptation language for communicating with the Rainbow framework. This language aims to improve adaptation decision making by capturing more effectively several adaptation factors that are considered relevant to an adaptation decision. A utility theory-based evaluation mechanism is then used for choosing the adaptation variant that maximizes user's utility.

6 Conclusions

This paper presented a context simulation engine developed in the scope of the MUSIC project to support developers of context-aware, self-adaptive applications into testing their applications. While many testing frameworks exist in terms of evaluating the functional properties of software, little support is provided in terms of testing extra-functional properties such as context-aware and self-adaptive behavior. The proposed framework achieves that in terms of reusable, parseable scripts which can be designed to evaluate the correctness and performance of custom applications. The proposed script format and the simulation framework are evaluated in terms of describing their applicability in the test of the SatMotion application. For the future we aim to enrich the scenario engine with more sophisticated commands and provide a graphical representation of the middleware performance during the simulation. Finally, the effectiveness of the tool will be evaluated in the scope of other MUSIC pilot applications as well.

Acknowledgements: The authors of this paper would like to thank their partners in the MUSIC-IST project and especially Ultan Carrol for his valuable input on the details of the communication aspects. We would also like to acknowledge the financial support given to this research by the European Union (6th Framework Programme, contract number 35166).

Bibliography

[AEP+07] M. Alia, V. S. W. Eide, N. Paspallis, F. Eliassen, S. O. Hallsteinsen, G. A. Papadopoulos. A utility-based adaptivity model for mobile applications. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*. Pp. 556–563. IEEE Computer Society Press, Niagara Falls, Ontario, Canada, May 2007.

[AHPE07] Alia, Hallsteinsen, Paspallis, Eliassen. Managing distributed adaptation of mobile applications. In *Proceedings of the 7th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'07)*. LNCS 4531, pp. 104–118. Springer Verlag, Paphos, Cyprus, 2007.

[CGS06] S.-W. Cheng, D. Garlan, B. Schmerl. Architecture-based self-adaptation in the presence of multiple objectives. In *SEAMS '06: Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*. Pp. 2–8. ACM, New York, NY, USA, 2006.
doi:http://doi.acm.org/10.1145/1137677.1137679

[DW08] W. Du, L. Wang. Context-aware application programming for mobile devices. In *Proceedings of the 2008 C3S2E conference*. Pp. 215–227. ACM, Montreal, Quebec, Canada, 2008.

[FHS+06] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, E. Gjorven. Using architecture models for runtime adaptability. *IEEE Software* 23(2):62–70, 2006.

[GCH+04] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, P. Steenkiste. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *Computer* 37:46–54, 2004.
doi:http://doi.ieeecomputersociety.org/10.1109/MC.2004.175

[Hen03] K. Henricksen. *A framework for context-aware pervasive computing applications*. PhD thesis, The University of Queensland, Sept. 2003.

[HM04] M. C. Huebscher, J. A. McCann. Simulation Model for Self-Adaptive Applications in Pervasive Computing. In *Proceedings of the Database and Expert Systems Applications, 15th International Workshop*. Pp. 694–698. IEEE Computer Society, 2004.

[KRW+09] M. U. Khan, R. Reichle, M. Wagner, K. Geihs, U. Scholz, C. Kakousis, G. A. Papadopoulos. An Adaptation Reasoning Approach for Large Scale Component-based Applications. Volume 19, p. 12. Electronic Communications of the EASST, Amsterdam, Netherlands, 2009.

[LLWC03] P. Levis, N. Lee, M. Welsh, D. Culler. TOSSIM: accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the 1st international conference on Embedded networked sensor systems*. Pp. 126–137. ACM, Los Angeles, California, USA, 2003.

[MPF+06] M. Mikalsen, N. Paspallis, J. Floch, E. Stav, G. A. Papadopoulos, A. Chimaris. Distributed Context Management in a Mobility and Adaptation Enabling Middleware (MADAM). In *Proceedings of the 21st Annual ACM Symposium of Applied Computing, Track of Dependable and Adaptive Systems (SAC'06)*. Pp. 733–734. ACM, Dijon, France, 2006.

[MSKC04] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, B. H. C. Cheng. Composing Adaptive Software. *IEEE Computer* 37(7):56–64, 2004.

[Pas09] N. Paspallis. *Middleware-based development of context-aware applications with reusable components*. PhD thesis, University of Cyprus, Sept. 2009.
http://member.acm.org/~nearchos/phd

[PEHP09] N. Paspallis, F. Eliassen, S. Hallsteinsen, G. A. Papadopoulos. Developing self-adaptive mobile applications and services with separation-of-concerns. In Nitto et al. (eds.), *At Your Service: Service-Oriented Computing from an EU Perspective*. Pp. 129–158. MIT Press, 2009.

[RBD+09] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, U. Scholz. MUSIC: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In *Software Engineering for Self-Adaptive Systems*. Pp. 164–182. 2009.

[RWK+08a] R. Reichle, M. Wagner, M. Khan, K. Geihs, J. Lorenzo, M. Valla, C. Fra, N. Paspallis, G. A. Papadopoulos. A Comprehensive Context Modeling Framework for Pervasive Computing Systems. In *Proceedings of the 8th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'08)*. LNCS 5053, pp. 281–295. Springer Verlag, Oslo, Norway, 2008.

[RWK+08b] R. Reichle, M. Wagner, M. U. Khan, K. Geihs, M. Valla, C. Fra, N. Paspallis, G. A. Papa. A Context Query Language for Pervasive Computing Environments. In *Proceedings of the 5th IEEE Workshop on Context Modeling and Reasoning (CoMoRea'08) in conjunction with the 6th IEEE International Conference on Pervasive Computing and Communication (PerCom'08)*. Pp. 434–440. IEEE Computer Society, Hong Kong, Mar. 2008.

[SG02] J. P. Sousa, D. Garlan. 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*. Pp. 29–43. Kluwer, B.V, Montreal, Quebec, Canada, 2002.

[Szy97] C. Szyperski. *Component software: beyond object-oriented programming*. Addison-Wesley Professional, Dec. 1997.