Managing Distributed Context Models Requires Adaptivity too

Christian Piechnick, Maria Piechnick, Sebastian Götz, Georg Püschel and Uwe Aßmann

Software Engineering Group, Technische Universität Dresden
Dresden, Germany
{christian.piechnick, maria.piechnick, sebastian.goetz1, georg.pueschel, uwe.assmann}@tu-dresden.de

Abstract. Mobile devices changed the way how software is developed fundamentally, because the changing context in which those devices are used, influences the requirements of apps running on those devices. Hence, mobile apps must provide means for self-adaptation. With upcoming technologies like wearables, people will carry a whole range of heterogeneous devices, sensing different aspects of their owner's context. Thus, self-adaptive systems have to collaborate, to combine their individual capabilities. To do so, they must incorporate strategies for the exchange of context information. The realization of such a distributed infrastructure requires to decide when, how and which parts of context models shall be exchanged, where they should be managed and who initiates the exchange. In this paper we show that for each aspect, different implementation strategies exist. Furthermore, we show that the properties of those strategies depended on (a) static and (b) dynamic requirements. Therefore, the management of distributed context models must be prepared for variability and adaptation. We present a prototypical implementation following the Smart Application Grids approach in the domain of Blended Interactive Spaces and highlight research questions w.r.t. self-adaptive distributed context models.

1 Introduction

Today, mobile devices are omnipresent and indispensable for both, personal and professional use. In the last years, the sales of stationary PCs constantly decreased, while the sales of mobile devices (e.g., smartphones, tablets) increased tremendously. As a consequence, mobile devices are present in almost every aspect of our daily life. Because those devices and the corresponding apps accompany people during their daily routines, they must be highly customizable. Further, people expect those apps to adapt their behavior to changing situations. Such a system is called a self-adaptive system (SAS). Typically, an SAS implements a variant of the MAPE-K feedback loop [12] to support adaptation in a structured way. In this process, one of the key models is the context model, capturing all potentially relevant information characterizing the situation of the application [4]. Especially in the application area of mobile apps, the knowledge of a single application is usually not sufficient to support more sophisticated decision making. The first reason is that mobile applications are rather small, specialized applications with a limited set of offered functionality. Thus, a single app has a limited scope w.r.t. the execution context. To expand this scope, multiple apps have to work together to combine their individual knowledge. Those apps, however, potentially were implemented by different developers and run in isolation. Consequently, generic approaches for the exchange of context information are required. Furthermore, even today, people carry a whole range of heterogeneous devices (e.g., smart watch, fitness trackers), sensing different aspects of their owner's context. In addition, hardware constantly gets cheaper and communicates on standard and open protocols. This paves the way for smart environments (smart home, smart office, etc.) consisting of different sensors and actuators, that can dynamically be integrated in the feedback loop of personal devices. Those developments are the enablers for ideas like *Blended Interaction Spaces* (BI Spaces), which are ubiquitous computing environments for computer supported collaboration [10]. Within those environments, heterogeneous devices work together seamlessly, to support collaborative workflows in physical spaces (e.g., rooms). BI Spaces must be highly adaptive w.r.t. the different users, tasks, requirements, devices, modalities, and interaction styles. Thus, applications in a BI Space must exchange contextual information. While the management of local context models is well investigated, one of the major challenges of todays self-adaptive systems is the management of *distributed context models*.

In the last decade, numerous solutions for the exchange of context information were realized, based on application-specific requirements. Investigating those solutions, we have identified 5 different aspects: **(1) What** information is accessible, **(2) When** context information should be exchanged, **(3) Who** initiates the exchange, **(4) Where** the information should be managed and **(5) How** retrieved information should be managed. We show that for answering those questions, different strategies exist having different benefits and drawbacks. We highlight that the corresponding requirements can change during runtime. For this reason, the context management itself should be self-adaptive. By now, to the best of our knowledge, no holistic approach for distributed context model management exists, which provides the required flexibility to self-adapt according to the five above questions. The intention of this work is to provide an initial categorization of different interaction strategies for the adaptive exchange of context information. In this paper, we show how an infrastructure for adaptive exchange of context information can be implemented with Smart Application Grids (SMAGs), an approach for the development and execution of self-adaptive systems, supporting Meta-Adaptation, from our previous work [16].

This paper is structured as follows, in Section 2 we present five different aspects w.r.t. the exchange of contextual information and the corresponding implementation strategies. In Section 3 we present a prototypical implementation of self-adaptive exchange infrastructure based on the SMAGs technology. Finally, in Section 4 we summarize our work and highlight important research topics.

2 Distributed Context Model Management

In scenarios where distributed SAS must work together to extend the individual knowledge bases (e.g., using a smart watch to improve the activity tracking of a smartphone), the system developers must provide means for the exchange of context information. Usually, a central feedback loop managing all self-adaptive applications is not feasible. Thus, the overall architecture consist of multiple collaborating feedback loops, executed on distributed, heterogeneous devices. One possible interaction is the exchange of monitored and potentially analyzed context information. In recent years, researchers have implemented different strategies to support context information exchange, based on their application-specific requirements. We have investigated 16 publications that support the ability to exchange context information [1–3, 5–9, 11, 13, 14, 17–21]. While analyzing the literature, we have identified the following application-specific requirements: *architectural flexibility, limited number of required connections, low data traffic, high expressiveness, small size of the individual models, high reasoning performance, fault-tolerance* and *the ability to handle privacy constraints.* The decision, which strategy should be used, depends on the individual static or dynamic requirements. To the best of our knowledge, the possible variation points as well as the corresponding implementation strategies for distributed context management have not been documented, yet. We have identified 5 aspects, the designer of a distributed context model management infrastructure has to cope with, regarding the exchange of contextual information. In some cases, the decision, which strategy should be used, can only be made at runtime. Thus, the context model management infrastructure must provide means for dynamic reconfiguration and must be subject to the system's feedback-loop, as well. If the same mechanisms are applied as for the adaptation of the business functionality, such an ability is called *Meta-Adaptation*. In this work, we assume that the distributed self-adaptive systems are able to find each other, can communicate and use a homogenous context model formalism and type system. We will use the term *Source* for the SAS sending context information to a *Sink*.

2.1 What Context Information Should be Exchanged?

This question focuses on the accessible content (see Figure 2.1). When all participating applications are developed by the same authorities, the access to the entire context model might be feasible. In more open scenarios, the access to private information must be restricted.

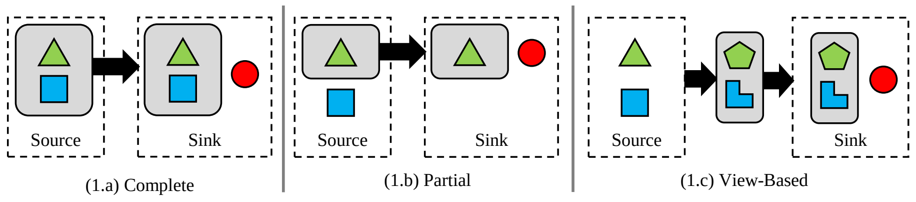

Fig. 1. What context information should be exchanged

Complete: The most common solution is to give any other application full access to the entire context model [1, 5–7, 9, 11, 21]. In this case, the context sink can decide which information may be relevant for its own decision making. On the other hand, however, for many applications this is not applicable, because of privacy issues. Furthermore, depending on the amount of context information and the update frequency, this approach leads to an increased data traffic. With an increasing number of collaboration partners, the amount of data might exceed the limit of the processing capabilities.

Partial In many cases, not all context information (a) should be accessible by [21] and (b) is relevant for a context-sink [5, 8, 13, 14, 17, 18, 20]. Therefore, the context-source might restrict access to certain information and/or the context-sink has the ability to exclude irrelevant information. This may improve privacy properties and reduces traffic as well as the overall amount of exchanged context information.

View-based Another possibility is to build views on a context model [2,3,5,17, 19]. This approach enables explicit privacy handling. Furthermore, the data traffic may be reduced, because views can describe abstractions of context information. The views can either be defined by the source or by the sink (e.g., queries with projection).

2.2 When Context Information Should be Exchanged?

This aspect focuses on the point of time and the circumstances, under which context information should be exchanged (see Figure 2.2). This is important to limit the amount of transferred data over time.

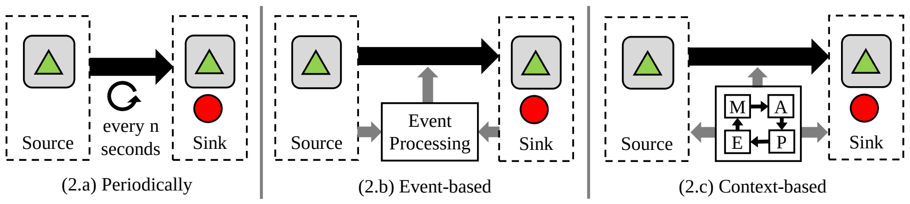

Fig. 2. When Context Information Should be Exchanged

Periodically In this strategy the information is either pulled by or pushed to the sink in certain time intervals [6, 7, 11]. The update frequency might be defined statically, depending on the category of information, or dynamically based on dynamic requirements. Especially for applications with highly frequent context updates, it is important to adjust the frequency accordingly, to prevent subsampling.

Event-based In contrast to periodic updates, this strategy triggers the exchange of contextual information based on events (e.g., explicit exchange request etc.), either fired by the source or the sink [3, 9, 11, 13, 14, 18–20]. In this approach the source and the sink have more influence on the exchange logic.

Context-based The exchange of context information can also be specified context-dependent [2,5]. In this strategy, a feedback loop decides based on the available context information (e.g., two devices are very close), when the exchange of contextual information should be initiated.

2.3 Who Initiates the Exchange of Context Information?

This aspect focuses on the participant, initiating the exchange (see Figure 2.3). For some applications, the source has the knowledge when the exchange should be initiated, in other scenarios the sink.

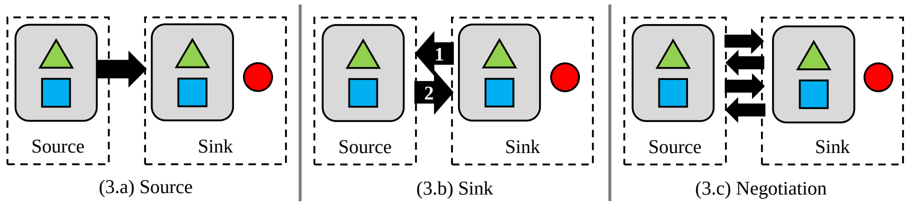

Fig. 3. Who Initiates the Exchange of Context Information

Source In this strategy the source proactively distributes context information to sinks [1, 3, 5, 9, 14, 18]. In contrast to sink-side initiation, the sink may not be able to specify which information is interesting in a certain situation. On the other hand, the source has full control over it's data and how it is distributed.

Sink Another variant is that sink pulls context-information from a source [5–7, 11,17,19,20]. For this approach, the source application has to offer a context query interface which enables the sink to retrieve information. Usually, this is combined with a periodic update mechanism.

Negotiation The sink- and source-based initiation can be combined in a blackboard-based architecture [2,13,17]. Sinks can access the context-model of a source via a query interface and retrieve data. The server might grant or deny access and might offer customized views. Sinks may be able to register for certain events (e.g., the temperature changed) and the source will notify the sinks when those events occur. This strategy is more complex and more expensive to implement. On the other hand, it offers more flexibility w.r.t. data traffic and privacy issues.

2.4 Where Should Context Information be Managed?

This question focuses on the location, where the context model data should be stored (see Figure 2.4. Especially for resource-limited devices it is necessary to outsource the context management and processing to more capable devices.

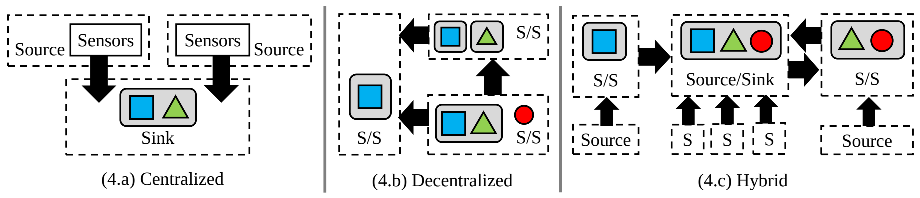

Fig. 4. Where Should Context Information be Managed

Centralized In this approach, a central system manages the context information of the participating self-adaptive systems [1, 3, 7, 11]. This decreases the number of required connections and data traffic. For devices with limited resources (e.g., CPU, main memory) this approach might be beneficial, because potentially large data sets and the corresponding reasoning mechanism can be outsourced. On the other hand, this approach introduces a single point of failure and makes privacy handling rather difficult.

Decentralized In contrast to the centralized approach, context models can be managed completely decentralized. Thereby, every application manages its own context model and can exchange context information with other applications [2, 5, 6, 9, 13, 14, 17, 19–21]. This approach makes privacy handling easier, since every application can decide whether or not certain information should be shared with a collaboration partner. Furthermore, this will decrease the size of the context models and increases the performance of the specific reasoning processes. On the other hand, this approach leads to a higher number of connections (i.e., combinatorial in the worst case) and increased data traffic.

Hybrid In situations where neither a fully centralized, nor a fully decentralized approach is feasible, the benefits of both architectures can be combined in a hybrid approach, with multiple central context managing applications [8, 18]. The individual source applications transfer their context information to a sink. The sink however might exchange information in a decentralized way. The concrete architecture can either be modeled statically, when all participating applications or application types are known, or dynamically when the appropriate architecture depends on dynamic information (e.g., the context). This may limit the number of connections and decreases data traffic, while still being able to handle privacy issues.

2.5 How Should the Distributed Context Information be Managed?

This question focuses on the strategies, how the context information should be stored (see Figure 2.5). Commonly, the received context information might be copied to the context model of the client. Sometimes, it is better to just store a reference to the actual data stored in the context model of the source.

Copy The most common strategy, to include the received contextual information in the client's context model, is by copying the received elements in the knowledge base [1, 5–7, 11, 13, 14, 18, 20, 21]. This strategy is easy to

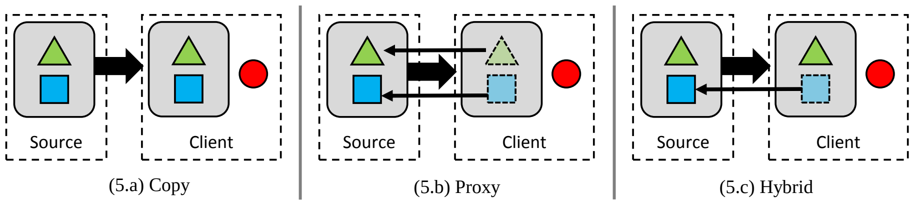

Fig. 5. How Should the Distributed Context Information be Managed

implement and is suitable for situations, where the number of reads of the copied elements (i.e., during reasoning) is similar to the number writes (i.e., reception of updated information).

Proxy When the number of writes of received context information exceeds the number of reads, it might be beneficial to only store a reference to the original information [2, 8]. This decreases the size of the stored model as well as the data traffic. On the other hand, the reasoning performance is decreased when the remotely managed information must be transferred on-demand.

Hybrid In situations where neither the concrete elements within the context model nor their characteristics (e.g., value range, update frequency, number of reads, degree of distribution) is known during design time, a hybrid approach can be applied [3, 17, 19]. The sink could start copying the values. When the number of updates exceeds the number of reads, the sink can switch to a proxy-based approach. This strategy introduces additional complexity, since additional monitoring and decision making algorithms are needed, on the other hand, this might decrease data traffic and the size of the managed model.

3 An Examplary Self-Adaptive Context Exchange Solution with SMAGs

To show the feasibility of runtime adaptation of distributed context model management we have implemented a small exemplary application with one adaptation, dynamically switching from a copy- to a proxy-based information management strategy (see Subsection 2.5). As one of multiple options, we have used the Smart Application Grids (SMAGs) approach to implement our solution, because it was developed within our group and supports Meta-Adapation natively. We expect meta-adaptation to be a feature of any middleware for self-adaptive systems. In SMAGs, self-adaptive systems are modeled and implemented as component-based systems and uses role-based programming for adaptation [15]. In contrast to many other platforms for self-adaptive systems, the adaptation mechanisms are developed in the same technological space as the self-adaptive systems itself. This enables Meta-Adaptation, by adapting the feedback loop itself at runtime [16]. Our example application originates from the domain of *Blended Interactive Spaces*, implementing the *"Bump-to-Give Gesture"*. In the scenario, a sender application running on `Device A` sends a file (e.g., an image) to a receiver application running on `Device B`, when those devices are *bumped* together. To implement such a scenario one must detect (1) when two devices are physically near (e.g., using the RSSI value of the Bluetooth sensor) and (2) that those devices were bumped together (e.g., using the accelerometer of the respective devices). To avoid misinterpretations, a plausibility check is performed by analyzing the proxemics as well as the motion interpretation of both devices.

In our first implementation, the adaptation logic was completely implemented on `Device A`, while `Device B` pushed all the context information to the first device. This solution was easy to implement, since all synchronization and interpretation artefacts could reside in one application. For testing, we used two *Samsung Galaxy S5* devices running Android 5.0. The accelerometer has an average sampling rate of about 90Hz, while the RSSI update frequency using Bluetooth Low Energy in *"Low Latency"* mode, has a sampling rate of 10Hz. Overall, this approach leads to an average bandwidth of about 20 kB/s, per collaborating device. Furthermore, about 100 nodes are added to the context model of `Device A` per second, per collaborating device. In more realistic scenarios, usually more than 2 devices are involved, leading to high data rates and model sizes. The average latency for data evaluation in case of an actual bump-to-give gesture was about 49ms on local Wi-Fi. For improving the situation we have implemented a role for monitoring the reads/writes of remotely obtained context values. When the number of writes exceeds the number of reads to a ratio of 2:1, instead of updating the actual values, a proxy is created, which redirects the read to a remote call to the actual data. Thus, updates must not be pushed or pulled constantly. This approach decreases the data rate to almost 0 kB/s, since the accelerometer data of `Device B` is only evaluated, when a bump-gesture was detected on `Device A`. On the other hand, however, when a bump occurs, the reasoning performance is decreased to 337 ms. Especially, for such visual reactions, the latency is perceivable.

This example is just an indication how self-adaptive distributed context management can be realized, not a full evaluation. In future, the possible realizations of the individual strategies must investigated exhaustively.

4 Conclusion and Future Work

In a world where people carry multiple devices and enter interactive intelligent spaces with a wide range of heterogeneous sensors and actuators, the distributed self-adaptive systems running on those devices must be able to dynamically exchange context information. In recent years many application-specific solutions were developed, meeting the requirements of the individual use-cases. In this work we have presented 5 different aspects that have to be considered w.r.t. exchange of contextual information. Furthermore, we have documented the different implementation strategies that were used in solutions or projects that support the management of distributed context information. This categorization should help developers to find a proper implementation strategy when designing and implementing solutions for distributed context models. Furthermore, it is also possible to change the concrete strategies at runtime. This requires to adapt the feedback loop and the context model management at runtime. Such an approach is called *Meta-Adaptation*. In addition, we have described a first prototypical implementation of a self-adaptive distributed context model management infrastructure using the Smart Application Grids approach in a Blended Interaction scenario. We have shown that the choice of a certain strategy might depend on dynamic context information, and thus, must be subject to adaptation as well. We think that this work is a starting point to an extensive investigation of variability in distributed context model management. We have only used a rather limited number of publications to develop a categorization. The presented categorization must either be confirmed or extended in a more extensive study. An exhaustive investigation w.r.t. realization possibilities must done. Furthermore, the non-functional properties of the different implementation strategies must be examined extensively. In addition, it should be investigated how the additional complexity influences the performance and the maintainability of the self-adaptive systems. Currently, there is no generic solution for coping with distributed context models that rely on different modeling formalisms and heterogeneous type systems. With the ability to adapt and extend the context model management of an application's feedback loop opens up the possibility to provide adapters for bridging technological spaces. This situation, however, is very likely to occur in open scenarios with heterogeneous devices and apps of different vendors and developers.

Acknowledgment

This work is supported by the German Federal Ministry of Education and Research (BMBF) within the project "SysPlace" (FKZ: 01—S14018E).

References

1. Broekstra, J., Kampman, A., Harmelen, F.v.: Sesame: A generic architecture for storing and querying rdf and rdf schema. In: Proceedings of the First International Semantic Web Conference on The Semantic Web. pp. 54–68. ISWC '02, Springer-Verlag, London, UK, UK (2002)
2. Bures, T., Gerostathopoulos, I., Hnetynka, P., Keznikl, J., Kit, M., Plasil, F.: Deeco: An ensemble-based component system. In: Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering. pp. 81–90. CBSE '13, ACM, New York, NY, USA (2013)
3. Cheng, S.W., Huang, A.C., Garlan, D., Schmerl, B., Steenkiste, P.: An architecture for coordinating multiple self-management systems. In: Proceedings of the 4th Working IEEE/IFIP Conference on Software Architectures (WICSA4). Oslo, Norway (11-14 June 2004)
4. Dey, A.K.: Understanding and using context. Personal Ubiquitous Comput. 5(1), 4–7 (Jan 2001)
5. Dey, A.K., Abowd, G.D., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Hum.-Comput. Interact. 16(2), 97–166 (Dec 2001)

6. Dobrican, R.A., Zampunierisl, D.: Moving towards a distributed network of proactive, self-adaptive and context-aware systems. In: In Proceedings of the Sixth International Conference on Adaptive and Self-Adaptive Systems and Applications 2014 (ADPATIVE 2014). pp. 22–26. ThinkMind, Venice, Italy (2014)
7. Götz, S., Wilke, C., Schmidt, M., Cech, S., Waltsgott, J., Fritzsche, R.: Theatre resource manager interface specification v. 1.0. Tech. Rep. TUD-FI10-08, ISSN 1430-211X, Technische Universitt Dresden (December 2010)
8. Hess, C., Romn, M., Campbell, R.: Building applications for ubiquitous computing environments. In: Pervasive Computing, Lecture Notes in Computer Science, vol. 2414. Springer Berlin (2002)
9. de la Iglesia, D.G.: A Formal Approach for Designing Distributed Self-Adaptive Systems. dissertation, Linaeus University (2014)
10. Jetter, H.C., Reiterer, H., Geyer, F.: Blended interaction: Understanding natural human-computer interaction in post-wimp interactive spaces. Personal and Ubiquitous Computing (DOI 10.1007/s00779-013-0725-4) (Oct 2013)
11. Kaila, L., Mikkonen, J., Vainio, A.M., Vanhala, J.: The ehomea practical smart home implementation. In: Proceedings of Pervasive'08. Sydney, AU (2008)
12. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer 36(1), 41–50 (Jan 2003)
13. Koster, A., Koch, F., Dignum, F., Sonenberg, L.: Augmenting bdi with relevance: Supporting agent-based, pervasive applications. In: Pervasive Mobile Interaction Devices (PERMID 2008), Workshop at Pervasive 2008. Sydney, Australia (2008)
14. Patel, P., Morin, B., Chaudhary, S.: A model-driven development framework for developing sense-compute-control applications. In: Proceedings of the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation. pp. 52–61. MoSEMInA 2014, ACM, New York, NY, USA (2014)
15. Piechnick, C., Richly, S., Götz, S., Wilke, C., Aßmann, U.: Using Role-Based Composition to Support Unanticipated, Dynamic Adaptation – Smart Application Grids. In: Proceedings of ADAPTIVE 2012, The Fourth International Conference on Adaptive and Self-adaptive Systems and Applications. pp. 93–102 (2012)
16. Piechnick, C., Richly, S., Kühn, T., Götz, S., Püschel, G., Assmann, U.: Contextpoint: An architecture for extrinsic meta-adaptation in intelligent environments. In: Proceedings of The Sixth International Conference on Adaptive and Self-Adaptive Systems and Applications. pp. 121–128. XPS Press (2014)
17. Quigley, A., Mcgrath, M., Nixon, P., Dishongh, T.: Home deployments for independent living. In: Proceedings of the Workshop on Pervasive Computing @ Home. Sydney, AU (2008)
18. Quix, C., Barnickel, J., Geisler, S., Hassani, M., Kim, S., Li, X., Lorenz, A., Quadflieg, T., Gries, T., Jarke, M., Leonhardt, S., Meyer, U., Seidl, T.: Healthnet: A system for mobile and wearable health information management. In: Proc. of the 3rd International Workshop on Information Management in Mobile Applications (IMMoA 2013). pp. 36–43. CEUR-WS.org, Riva del Garda, Trento, Italy (2013)
19. Reichle, R., Wagner, M., Khan, M., Geihs, K., Lorenzo, J., Valla, M., Fra, C., Paspallis, N., Papadopoulos, G.: A comprehensive context modeling framework for pervasive computing systems. In: Distributed Applications and Interoperable Systems, Lecture Notes in Computer Science, vol. 5053. Springer Berlin (2008)
20. Weiss, G., Becker, K., Kamphausen, B., Radermacher, A., Gerard, S.: Model-driven development of self-describing components for self-adaptive distributed embedded systems. In: Proceedings SEAA. pp. 477–484 (2011)
21. Zhang, G., Parashar, M.: Dynamic context-aware access control for grid applications. In: Stockinger, H. (ed.) GRID. pp. 101–108. IEEE Computer Society (2003)