Evaluation of Extensibility, Portability and Scalability in a Database-centric System Architecture for Smart Home Environments

Wagner O. de Morais[*] and Nicholas Wickström[†]

School of Information Technology
Halmstad University
Box 823, Halmstad 301 18, Sweden

March 2015

Abstract

Advances in database technology allow modern database systems to serve as a platform for the development, deployment and management of smart home environments and ambient assisted living systems. This work investigates non-functional issues of a database-centric system architecture for smart home environments when: (i) extending the system with new functionalities other than data storage, such as on-line reactive behaviors and advanced processing of longitudinal information, (ii) porting the whole system to different operating systems on distinct hardware platforms, and (iii) scaling the system by incrementally adding new instances of a given functionality. The outcome of the evaluation is demonstrated, and analyzed, for three test functionalities on three heterogeneous computing platforms. As a contribution, this work can help developers in identifying which architectural components in the database-centric system architecture that may become performance bottlenecks when extending, porting and scaling the system.

Index terms— database-centric architecture, smart environments, ambient assisted living, quality attributes

1 Introduction

The inherent and evolving diversity (e.g. users, needs and technologies) of smart environments still challenges the development of smart homes and ambient assisted living (AAL) systems. People have unique needs and preferences that change over

[*]Wagner.deMorais@hh.se
[†]Nicholas.Wickstrom@hh.se

time, and these can lead to *extensibility* and *scalability* issues when modifying systems. To address people's needs, heterogeneous technologies are employed. These technologies are provided by different vendors, and operate and communicate using different standards and protocols. *Integratability* and *interoperability* issues come as a result of these, as does data heterogeneity. Homes also differ and people are very concerned about technologies that can monitor or take control of several aspects of their lives. These raise *privacy* and *security* issues. Some individuals might rely on different technologies to cope with different kinds of impairments and limitations, and thus technology's *dependability* is imperative. Over time, technology can also assist and benefit other users, such as formal and informal carers and healthcare professionals. Figure 1 attempts to illustrate this evolving diversity.

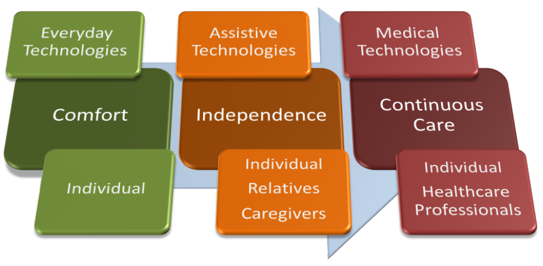

Figure 1: The model captures the evolving diversity of needs across the life-span. Over time, different technologies are employed to cater to the needs of primary users (the individual) and secondary users (relatives, caregivers and healthcare professionals). Such diversity is in the nature of smart homes and AAL.

Because of the aforementioned reasons, and because no universal selection and arrangement of devices or system configuration will fit every home settings, smart environments must be *extensible* to respond to evolving requirements with *scalable* solutions tailored according to individual needs and settings.

In general, one of the critical issues in the design and development of complex systems, such as smart environments, is their architecture [1]. Successful system architectures organize the elements that constitute the system not only to support functional properties, i.e. system behavior, but also non-functional critical properties (also known as quality attributes), that affect both system design and run-time behavior [2]. Moreover, an important principle in the development of system architectures for smart environments is to *build to change instead of building to last* [2].

Previously, in [3], a database-centric architecture for smart environments was proposed. According to the authors, features of modern database management systems (DBMS) allow a database system to serve as a platform for the development, deployment, and management of smart homes and AAL applications. The practical implication of the approach proposed was investigated with the integration and interoperation of heterogeneous technologies in a "smart bedroom" demonstrator.

Later in [4], active in-database processing using machine learning methods was employed not only to support "smart" reactive behavior of smart environments but also to accommodate privacy and security requirements.

This work concerns the evaluation of non-functional issues in a database-centric architecture when:

1. Extending the system with new features, *i.e.* extensibility;
2. Porting the architecture to different computing platforms, *i.e.* portability;
3. Increasing the load in the system by adding new running instances of a functionality, *i.e.* horizontal scalability.

As a contribution, this work provides a method to:

1. Understand how the system and implemented applications behave when ported across different operating systems on different computing hardware;
2. Identify which architectural components are mostly affected when extending, porting and scaling the system.

The remainder of this paper is structured as follows. Related work is presented in Section 2. An overview of the database-centric system architecture for smart environments is given in Section 3. Section 4 describes the methodology for evaluation. Section 5 concerns the experimental system set-up and data, and the results of the evaluation are presented in Section 6. Discussion and conclusions are covered in Section 7.

2 Related Work

Although the development of system architectures, middleware paradigms and platforms for smart homes and AAL systems has been the main topic of a number of research projects [5, 6], there is still no broadly adopted method for developing smart environments [7] or accepted metrics for evaluating them [8]. Moreover, given the evolving diversity of smart environments discussed Section 1, it is very unlikely that one single system architecture will perfectly fit the all requirements and environments [9]. Comparing and selecting a suitable architecture or platform for smart environments are therefore difficult.

Memon *et al.* recently identified that AAL research has not sufficiently focused on and addressed non-functional properties, such as interoperability, usability, reliability, security and privacy [6].

Design principles, such as economy of mechanisms, client simplicity and levels of indirection, have been employed to address portability, extensibility and robustness requirements in a middleware for interactive workspaces called iROS [10]. The authors concluded that a centralized design and implementation facilitated extensibility, portability, maintainability, scalability and robustness. Although the authors reported that the resulting scalability was acceptable for that class of smart environments, no information is provided as to whether the scalability tests included different operating systems or if extensibility created any scalability issues.

An evaluation of six well-known AAL platforms, such as universAAL [11], according to different quality attributes (reliability, security, maintainability, efficiency and safety) is reported in [12]. To evaluate the different platforms, the authors conducted a survey using semi-structured interviews. The number of interviewees was not mentioned. The authors observed that there are considerable differences among AAL platforms and that several of the above quality attributes are addressed only in part or not addressed at all.

As concerns AAL applications, the organizers of the “Evaluating AAL Systems through Competitive Benchmarking" (EvALL) competition selected five metrics split into two categories, termed hard and soft metrics [13]. Hard metrics, which can be objectively measured or quantified, include accuracy and availability. Soft metrics include installation complexity, user acceptance and integrability.

3 Database-centric System Architecture for Smart Environments - An Overview

Modern database systems can serve as a platform for smart home environments and AAL [3]. In the proposed database-centric architecture, the domain logic is contained in the DBMS, which becomes the most important architectural element in the system. Associated benefits of such an integrated and centralized approach are improved performance, security, data management and ease of implementation [3, 4].

The programming model of the above-mentioned architecture is a collection of independent software components, called resource adapters, that communicate with a central entity, *i.e.* the DBMS, referred to in the architecture as active database.

An overview of an integrated architecture of this kind is shown in Figure 2. The main components composing the architecture are described in the following subsections.

3.1 Active Database

In the active database, database triggers, user-defined functions (UDFs) and mechanisms for interprocess communication (IPC) are employed to detect and respond to events taking place in the environment. UDFs, IPC mechanisms and database views are used to create a database interface, which offers a clean interface with a set of methods for data access and manipulation (select, insert, update and delete), as well as to notify client applications connected to the active database. In addition to interoperability at the data level, an approach of this kind facilitates the portability of client application across distinct computing platforms because most of functional logic is implemented in the active database. The authors also take

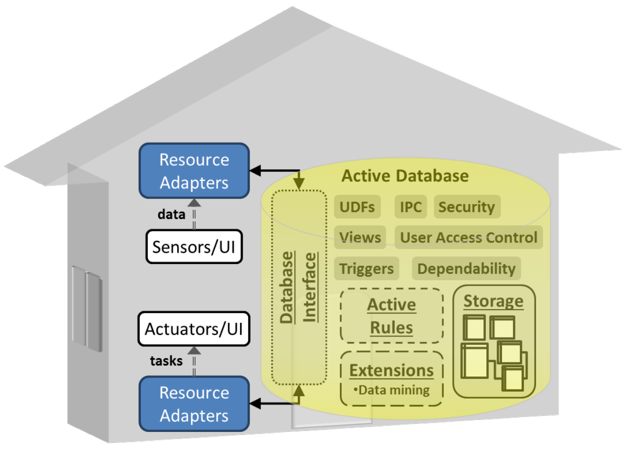

Figure 2: A database-centric architecture has been suggested as a platform for the development of smart environments [3]. Resource adapters and the active database are the two main components of the architecture.

advantage of UDFs to perform in-database processing, *i.e.* the DBMS is extended with the logic to process stored data. This logic may correspond to the semantics of an application or of a method for machine learning. Data security is an outcome of this, because no sensitive information is maintained outside the active database.

3.2 Resource Adapters

The concept of resource adapter was proposed by the authors to abstract and integrate heterogeneous technologies (*e.g.* sensors, actuators, user interfaces, software libraries) into the system [3]. The main functional aspects of a resource adapter are: i) stream data acquired by sensors or entered by the user to the active database or ii) control actuators or user interfaces as a response to commands received from the active database. Resource adapters do not interact directly with the tables in the active database but use methods provided by the database interface. Such an approach is aligned with the principles of “economy of mechanisms" (*i.e.* fewer mechanisms to port) and “client simplicity" (*i.e.* move the complexity to the server) [10], which are strategies for facilitating portability. Resource adapters are responsible for initiating the communication session with the active database and using a hybrid communication model. For streaming or querying data from the active database, resource adapters employ a client-server computing model and intermittent connections with the active database are more common. However, it has been observed that maintaining a continuous connection improves the throughput of resource adapters abstracting sensors that generate data at higher sampling rates (more than 10Hz). A publish-subscribe pattern enables resource adapters to receive notifications from the active database, such as control commands and actions to be executed, and this avoids database polling. A continuous connection with the active database is required to enable this messaging pattern. Each resource adapter has a universally unique identifier (UUID), that also identifies a unique channel used by the active database to publish notifications specific to a resource adapter. Further, the active database has its own UUID and corresponds to a global channel to which all resource adapters listen, so the active database can notify all resource adapters more efficiently.

4 Methodology

Figure 3 illustrates the process for evaluating scalability when extending a system with new functionalities. The complete scenario encompasses three related test functionalities exemplifying applications that are typically implemented in smart homes and AAL systems.

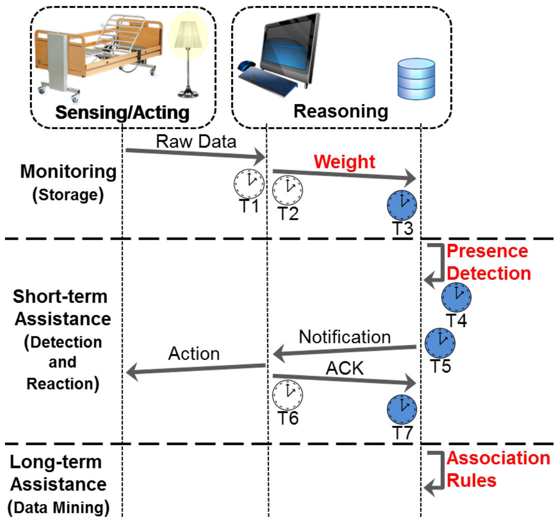

Figure 3: Weight monitoring, bed exit detection and common event transitions are the selected test functionalities for monitoring, short- and long-term assistance. The white clocks are timestamps attributed by resource adapters. Blue clocks are timestamps attributed by the active database.

Functional, physiological and safety monitoring and assistance are the main applications of smart homes in healthcare [14]. Storage of long-term health-related data is one of the most important features of these systems. Monitoring is therefore the first test functionality. As an extension of the first functionality, the other two test functionalities include short- and long-term forms of assistance. Short-term types of assistance include solutions that on-line detect and respond to events. Long-term types of assistance encompass solutions that require datasets collected over a longer period of time and employ advanced methods for data analysis. The rationale behind the proposed scenario is that:

- *Extensibility* implies that all three functionalities must be incorporated into the system;
- *Portability* encompasses the cross-platform capability of the underlying software architecture and the three functionalities, *i.e.* to run on different computing platforms; and
- *Scalability* requires an objective evaluation of the overall effect of loading the system with more instances of running functionalities, such as monitoring.

To objectively evaluate scalability, the interaction between different components must be measured for each test functionality on different computing platforms. Ideally, given the test functionality and the system configuration, by gradually adding new instances of the selected functionality, the workload in the system will increase and subsequently lead to increased delays in data streaming and processing as well as latencies in responding to events. These are issues that can compromise the reliability of a system. Therefore, to isolate the system components that are affected by the computational workload, the following measures are computed:

Pre-processing Delay ($T2-T1$): time to pre-process the raw sensor reading and make it available for storage.

Storage Delay ($T3-T2$): time required to transmit and store data.

Detection Delay ($T4-T3$): time to process an event of interest.

Detection Latency ($T4-T1$): total time to detect an event of interest.

Notification Delay ($T5-T4$): time to generate a notification for a detected event.

Notification Latency ($T6-T5$): time taken for the notification to reach its destination.

Detection/Notification Latency ($T6-T4$): time from the detection of an event to the reception of the notification for that event.

Reaction Latency ($T6-T1$): the total reaction time or system response time.

ACK Delay ($T7-T6$): time to acknowledge a notification.

Notification/ACK Latency ($T7-T5$): time from the generation of a notification until it is acknowledged.

As a result, the proposed approach can help in identifying which and how many functionalities are supported by a given computing platform configuration, and which architectural components are most affected when extending, porting and scaling the system.

5 Experimental System Setup and Data

Three distinct hardware platforms running different operating systems were selected for the evaluation. The configuration of each computer is presented in Table 1.

Table 1: The selected computing platforms.

	Dell Optiplex 7010	Dell Latitude E7240	Raspberry Pi B Rev 2
CPU	Intel(R) i7-3770	Intel(R) i5-4300U	ARM11
Speed	3.40GHz	1.90GHz	1000 MHz (overclocked)
Cores / Threads	4 / 8	2 / 4	1 / 1
RAM	16 GB	8 GB	512 MB
Storage	500 GB SSD	125 GB SSD	4 GB SDHC Class 10
OS	CentOS Linux 7	MS Windows 7	Raspbian wheezy GNU/Linux 7

In [3], a smart bedroom demonstrator was implemented according to the database-centric architecture. The demonstrator features a smart bed that provides weight monitoring, among other functionalities. Later, in [4], the authors propose an in-database method that on-line detects bed entrances and exits. Another in-database method reported by the authors employs association rules to detect common room transitions and anomalies during the night. Therefore, weight monitoring, bed exit detection and common event transitions were selected in the present work as the three test functionalities composing the experiment (Figure 3).

Using the smart bed, a dataset containing 75 seconds of weight data was collected and is used in this experiment to implement the three test functionalities described in Section 4. A subject participating in the data collection was instructed to lie down in bed 15 seconds after the measurement started and to leave the bed after 45 seconds. A resource adapter was created to simulate a weight measurement system installed in the smart bed. The resource adapter reads samples from the dataset file and streams the data to the active database at the same sampling rate of the real system, *i.e.* 80Hz. This constitutes the weight monitoring functionality. For the reactive short-term form of assistance, a threshold-crossing mechanism was used to detect bed entrances and exits. Association rules to detect common event transitions were used to implement a long-term type of assistance. Particular details of these two methods are found in [4]. An overview of the amount of computation of each functionality is given as follows:

Weight monitoring: generates 80 inserts in the active database per second.

Bed-exit detection: includes the workload of the weight monitoring and a trigger that fires every half second. This trigger the UDF to compute the mean value and the standard deviation of the last 80 inserted samples to detect bed exits and entrances. Events that are detected are inserted into a separate table. Each new event generates an IPC notification to resource adapters interested in that type of event.

Common event transitions: includes the workload of the weight monitoring and bed exit detection functionalities. A trigger fires at every bed entrance or exits and updates the tables for computing the association rules, such as transition matrix, support metric and confidence metric tables.

6 Evaluation Results

Each selected computing platform configuration was tested for each test functionality, and hosted both the simulated systems and the active database. For each test functionality (described in subsection 5 and illustrated in Figure 3), a corresponding in-database method was implemented in PostgreSQL [15] using PL/pgSQL. PostgreSQL is well known by its extensibility features, which enable new added functions and data types to perform as though they were native objects. Moreover, extensions are added and modified "on the fly", which is a necessary capability for smart environments.

Because PostgreSQL [15] is a cross-platform DBMS, portability was not an issue. PostgreSQL's binary installation packages are available for CentOS and MS Windows 7. For the Raspberry Pi, it was necessary to build and install PostgreSQL using its source code distribution. In each system set-up, PostgreSQL was configured for security communication and improved performance.

The resource adapter was implemented in Python, and as a result, the same code runs on the selected operating systems. The portability of the resource adapter and the test functionalities were facilitated by the domain logic residing in the active database and data manipulation being provided by the database interface.

To evaluate the scalability of each test functionality in the different computation platforms, the simulation started with the execution of one instance of the simulated system. After this first execution, two instances were executed at the same time. The execution of multiple simulated systems was gradually increased until the system workload affected the execution (resource starvation) of the components involved. A process manager was implemented to synchronize the execution of the simulated systems. The process was repeated 20 times in order to achieve more accurate results, and the average values of the measures are used.

A system will not scale if it is not able to maintain the average sampling rate of 80Hz (one sample every 0.125 seconds). This time difference will accumulate over time and will result in delays and latencies in different system modules. Figure 4 illustrates the effect of an increased system workload on the simulated sampling rate. The Raspberry Pi is able to host only one simulated weight monitoring system. The Raspberry Pi might be suitable for scenarios in which sensors generate data at very low sampling rate (less than 1Hz), for example, motion sensors. Windows can support up to four simulated systems executing simultaneously with the active database. CentOS supports up to eight systems.

Figure 5 illustrates the evaluation results for the monitoring test functionality. A commonly observed effect was an increase in storage delay, especially in the

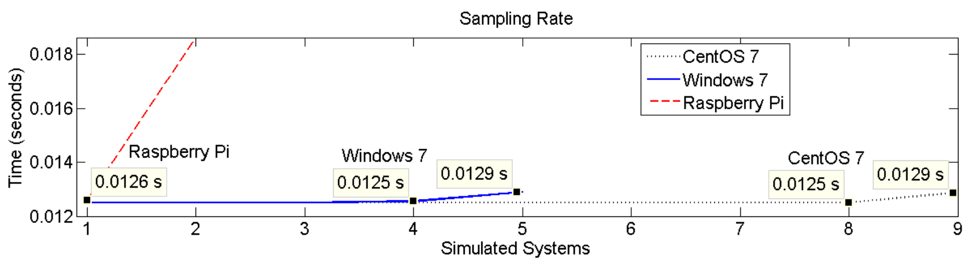

Figure 4: Data tips in the graph show the sampling rate for one or more simulated systems in different computing systems. The system should maintain a constant sampling rate. Hence, the graph helps in identifying the maximum number of simulated systems executing in parallel in each computing platform.

Raspberry Pi system. Compared with the high-end computer running CentOS, the storage delay in the Raspberry Pi is 16 times larger. It is twice as large in Windows.

An increased storage delay is also present in the Windows-based system and in the short-term reaction test functionality. In that functionality, the Windows-based system also presented an increased notification delay as more systems were executed concurrently, as illustrated in Figure 6. For the CentOS-based machine, the total reaction time plus notification acknowledgement is less than a millisecond. Hence, the overhead of monitoring and reacting through the active database and using the DBMS as interprocess communication mechanism is even less than that.

Figure 7 presents the evaluation results for all test functionalities for the computer with the highest computing capabilities. All measures are mostly constant for the different experiments except for the notification latency when the advanced data processing functionality is simulated. This means that the communication between the active database and the resource adapters can be affected by the workload created for this particular case.

The experimental results are summarized in Table 2. An overall observation is that architectural components abstracting hardware and software technologies are those that are most affected by an increased system computational workload.

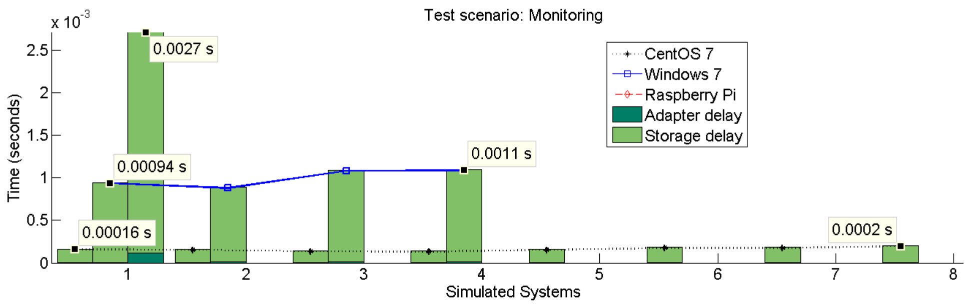

Figure 5: Lines in the graph indicate the total storage latency (pre-processing delay plus the storage delay).

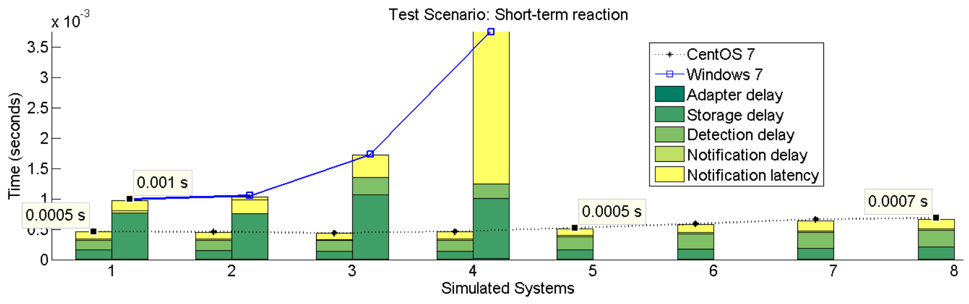

Figure 6: Effect of the reactive behavior for the computers running Windows 7 and CentOS. Lines indicate the total reaction storage latency (sum of all delays since the sensor reading until a response is provided). The computer running CentOS can maintain the same average performance for up to five simultaneous systems. Data tips indicate the total reaction time plus the notification acknowledgement.

Interestingly, the number of supported systems in a given computing platform, at least for the monitoring functionality, is equal to the number of logical cores in the CPU of the platform. The time resolution issues in Windows 7 [16] are worth mentioning, as they affect the measured values both from the resource adapters and the database system. This prevents accurate comparison of the architecture on different operating systems (Windows 7 and CentOS) on top of the same hardware platform. It also prevents generating accurate timestamps and timer functions. To cope with this issue, a busy-waiting method was implemented and is also used to simulate the real sampling rate. More experiments are needed to evaluate the proposed test functionalities and system configurations on a distributed scenario, *i.e.* the resource adapters and the active database running on separate machines. However, this type of approach could lead to even higher delays and latencies due to the network communication.

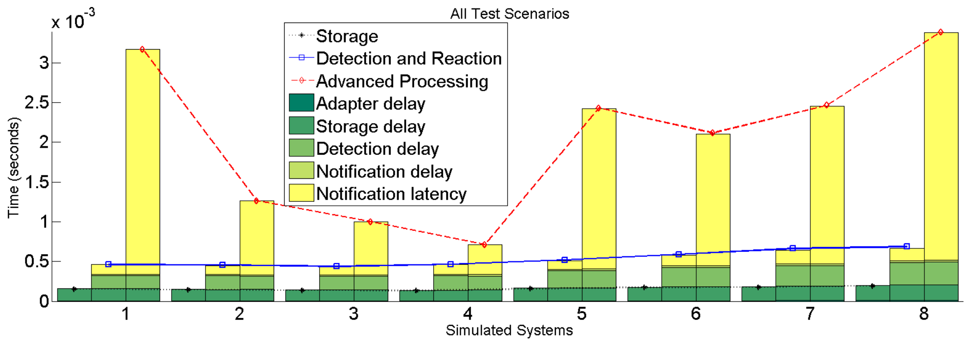

Figure 7: The evaluation of all test functionalities for the computer running CentOS. Lines indicate the response time or reaction latency (sum of all delays since the sensor reading until a response is provided). The notification latency is clearly affected by the data processing of the association rules.

Table 2: Summary of findings. Not supported means that the storage or the reaction latency was higher than the sampling rate.

	Dell Optiplex 7010	**Dell Latitude E7240**	**Raspberry Pi B Rev 2**
Extensibity	All functionalities	All functionalities	All functionalities
Portability	All functionalities	All functionalities	All functionalities
Scalability (Monitoring)	8 Simulated Systems	4 Simulated Systems	1 Simulated System
Scalability (Short-term)	8 Simulated Systems	4 Simulated Systems	Not supported
Scalability (Long-term)	8 Simulated Systems	Not supported	Not supported

7 Conclusion

Embracing extendable, portable and scalable system architectures that can respond to evolving needs is the most reasonable strategy to realize successful smart environments. However, the most important aspect is actually to provide evidence that a given system architecture supports these non-functional requirements.

This work investigated the extensibility, portability and scalability of a database-centric system architecture for smart environments. Different measures were employed to examine the interaction and behavior of different components in the system in order to identify and isolate system components that may become performance bottlenecks. Three test functionalities (data storage, on-line reactive behavior and advanced data processing) on three heterogeneous computing platforms were implemented and evaluated.

Results demonstrated the flexibility of the database-centric architecture towards being extended and ported across different operating systems and computer hardware. PostgreSQL plays an important role in this because of its extensible and cross-platform design. Results also revealed which functionalities and how many instances of a given functionality are supported in three different computing platforms. Resource starvation added delay in the input data streaming and processing, as well as latency in event processing and response. Components abstracting hardware and software technologies are the most affected when increasing the computational workload in the system. As a conclusion, this work can help developers in identifying which architectural components become performance bottlenecks when extending, porting and scaling a database-centric system architecture for smart home environments.

References

[1] Mourad Chabane Oussalah. *Software Architecture 1*. Wiley-ISTE, Hoboken, NJ, June 2014.

[2] Microsoft P. & P. Team. *Microsoft® Application Architecture Guide*. Microsoft Press, Redmond, washington, 2nd edition, October 2009.

[3] W. O. de Morais and N. Wickström. A “smart bedroom” as an active database system. In *Proceedings of the 2013 IEEE 9th International Conference on Intelligent Environments (IE)*, pages 250–253. IEEE Computer Society, 2013.

[4] W. O. de Morais, J. Lundström, and N. Wickström. Active in-database processing to support ambient assisted living systems. *Sensors*, 14(8):14765–14785, 2014.

[5] G. Fagerberg, A. Kung, R. Wichert, M-R. Tazari, et al. Platforms for AAL applications. In *Smart Sensing and Context*, pages 177–201. Springer, 2010.

[6] M. Memon, S. R. Wagner, C. F. Pedersen, F. H. A. Beevi, and F. O. Hansen. Ambient assisted living healthcare frameworks, platforms, standards, and quality attributes. *Sensors*, 14(3):4312–4341, 2014.

[7] J. C. Augusto, V. Callaghan, , D. Cook, A. Kameas, and I. Satoh. Intelligent environments: a manifesto. *Human-centric Computing and Information Sciences*, 3(1):1–18, 2013.

[8] A. Santos, H. Rodrigues, and R. José. Evaluating a crowdsourced system development model for ambient intelligence. In J. Bravo et al., editors, *Ubiquitous Computing and Ambient Intelligence*, pages 145–152. Springer, 2012.

[9] M. Becker. Software Architecture Trends and Promising Technology for Ambient Assisted Living Systems. In A. I. Karshmer et al., editors, *Dagstuhl Seminar Proceedings*, 2008.

[10] S. R. Ponnekanti, B. Johanson, E. Kiciman, and A. Fox. Portability, extensibility and robustness in iROS. In *Proceedings of the 2003 IEEE 1st International Conference on Pervasive Computing and Communications (PerCom 2003)*, pages 11–19. IEEE, 2003.

[11] M-R. Tazari, F. Furfari, A. F. Valero, S. Hanke, et al. The universAAL Reference Model for AAL. *Handbook on Ambient Assisted Living-Technology for Healthcare, Rehabilitation and Well-being. AISE Book Series*, 11:610–625, 2012.

[12] P. O. Antonino, D. Schneider, C. Hofmann, and E. Nakagawa. Evaluation of AAL Platforms According to Architecture-Based Quality Attributes. In D. V. Keyson et al., editors, *Ambient Intelligence*, volume 7040, pages 264–274. Springer Berlin Heidelberg, 2011.

[13] P. Barsocchi, S. Chessa, F. Furfari, and F. Potorti. Evaluating Ambient Assisted Living Solutions: The Localization Competition. *Pervasive Computing, IEEE*, 12(4):72–79, October 2013.

[14] G. Demiris and B. K. Hensel. Technologies for an aging society: a systematic review of "smart home" applications. *Yearbook of medical informatics*, pages 33–40, 2008.

[15] The PostgreSQL Global Development Group. *PostgreSQL 9.4.1 Documentation*. 2015. http://www.postgresql.org/docs/9.4/static.

[16] Microsoft® MSDN. Acquiring high-resolution time stamps, 2015. https://msdn.microsoft.com/en-us/library/windows/desktop/dn553408.aspx.