Simulation Modelling Practice and Theory 34 (2013) 208–220

Contents lists available at SciVerse ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Performance simulation of a context provisioning middleware based on empirical measurements

Eike Steffen Reetz [b], Michael Knappmeyer [a,b], Saad Liaquat Kiani [a], Ashiq Anjum [c], Nik Bessis [c,*], Ralf Tönjes [b]

[a] Faculty of Engineering and Technology, University of the West of England, Bristol, UK
[b] Faculty of Engineering and Computer Science, University of Applied Sciences Osnabrück, Osnabrück, Germany
[c] School of Computing and Mathematics, University of Derby, Derby, UK

ARTICLE INFO

Article history:
Received 21 December 2011
Received in revised form 7 March 2012
Accepted 10 March 2012
Available online 10 April 2012

Keywords:
Context
Pervasive
Middleware
Evaluation
Simulation

ABSTRACT

The evaluation of context middleware systems is a challenging endeavour. On the one hand, testbed investigations suffer from an unrealistic environment in terms of number of users, high implementation effort for changes and questionable portability of results. On the other hand simulation of middleware systems is complex due to the high abstraction of implementation. This article contributes to the understanding of a broker based context provisioning system based on black-box measurements of a testbed which are further utilised to increase the accuracy of a simulation model. Both simulations and measurements help in understanding the complex behaviour of a context provisioning middleware and enable the evaluation of complex distributed systems. The presented investigations identify significant parameters and corresponding models for the response delay of the key components of a context provisioning middleware and discuss their integration into an overall simulation model.

1. Introduction

Weiser's [1] vision of the proximate future depicts a world where interconnected smart entities are able to provide information on "anything, any time, anywhere". Since the inception of this concept nearly two decades ago, ubiquitous computing research has been dealing with the possibilities of the future; its progress has faced not only technological challenges but is also concerned with anticipation of future trends of human behaviour. Researchers have tackled this combined challenge by prototyping systems and applications in laboratory environments, even creating live-in laboratories with researchers as inhabitants of a futuristic abode. While our everyday environment has not transformed into such a world yet, continuing advances in communication technology, microelectronics and materials science indicate that the *Internet of Things* (IoT) is fast approaching realisation. Central to the theme of *IoT* is the processing and communication of information between smart objects. The information may relate to inhabitants of the environment, smart appliances or physical characteristics of the environment itself and is labelled as *context*. Due to the heterogeneous nature of digital and physical objects that may interact using *IoT* technologies, the vast scale they may encompass and resource management related issues, there is a need for the development of supportive context provisioning infrastructures so that the digital artefacts embedded in our smart environments can be fully utilised to support our seamless interaction with technology.

* Corresponding author. Tel.: +44 1332592108.
E-mail address: n.bessis@derby.ac.uk (N. Bessis).

1569-190X/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.simpat.2012.03.002

A number of such infrastructures, e.g. in the form of middlewares, have been proposed and many facets of context provisioning have been investigated during the last two decades. Different approaches in terms of communication paradigms, context modelling and representation, extendibility, entity diversity as well as processing and management scalability have been well addressed from the functional and qualitative point of view. Nevertheless, most of the proposed solutions have not proven their large-scale capabilities either by simulations or prototype deployment. Previous results ***in [2] have indicated that a simulation can aid in proving specific aspects of a context provisioning system (CPS) but is inadequate just at the functional level. In addition it is required to derive an appropriate model from testbed implementations and assessments. This model includes a functional model of the investigated context provisioning middleware and the related context, as well as a performance model of the underlying application server. Common problems in building a simulation model are caused by the need to build complex models of the real system, thus resulting in uncertainty, potential inconsistency and time consuming processes as well. Therefore many simulations have clear boundaries of the model validity and focus only on aspects of the system which can be abstracted more easily [3]. Moreover, prototypes provide only a very limited evidence with regard to scalability since creating a realistic environment is almost impossible in terms of heterogeneity, numbers of context sources and sinks, network traffic, etc. If assessed appropriately testbeds can reveal system-level behaviour as measurement results include hardware and software characteristics.

Our approach is to overcome the currently incomplete understanding of functionally described and evaluated context provisioning systems by improved simulation models based on black-box tests in a testbed. The measurements allow for identifying significant parameters and are utilised in order to build a realistic simulation model. The investigated model will guide us towards a deeper understanding of our proposed broker based CPS and will also assist in discovering and avoiding performance bottlenecks. Our context provisioning system has been implemented using Java Enterprise Edition (JavaEE) technologies and the performance of these technologies will also come under investigation in this article. The aim of our simulation model and the holistic evaluation is to evaluate a context provisioning middleware from different performance related angles and analyse the experimental results for elicitation of guidelines for further research and development in the domain of context provisioning in the *IoT*.

The rest of the article is structured as follows. Section 2 discusses related work in the field of the evaluation of context provisioning systems and relates to the general evaluation of JavaEE applications as well. Afterwards, Section 3 outlines our proposed broker-based context provisioning framework (entitled `C-ProMiSE`). The evaluation methodology is derived in Section 3.1. Section 4 presents prototype black-box assessments. The simulation models and results are discussed (Section 5) before Section 6 finally concludes the article.

2. Related work

Context provisioning systems (CPSs) aim at supporting heterogeneous context-aware applications/services systematically. In order to transparently facilitate access to context, they typically comprise the following set of functionalities: Sensor Data Acquisition, Context Modelling and Representation, Context Lookup and Discovery, Context Storage, Context Diffusion and Distribution, Context Processing and Reasoning [4]. A survey of how to evaluate such systems has been presented in our earlier work [4], in which a multidisciplinary assessment methodology is introduced and suitable performance metrics are suggested based on the analysis of surveyed systems. In detail, prototyping (e.g. [5–7], field trials including Experience Sampling Method (ESM) (e.g. [8–10]), context emulation (e.g. [11–13]), emulation of middleware components (e.g. [12,13]) and the emulation of actuation (e.g. [14,15]) are proposed.

However, context-aware applications/services are likely to follow location-based services and step out of the lab and into the real world in the proximate future. Due to the increasing market penetration of technologically advanced smartphones being the users' everyday companion and primary device the number of context sources and context sinks will grow tremendously. This is why scalability has evolved – together with security and privacy – as one of the key concerns. To investigate the scalability of a CPS not only qualitatively but quantitatively, the authors strongly recommend a system-level simulation based on Discrete Event Simulation (DES). Since only a very limited number of works (e.g. [2,16]) have applied DES in the domain of context provisioning, we aim at contributing to the corresponding understanding, model development and simulation parameter setting.

Multiplicity of different influence factors in programming platforms such as Java and C++ make it very difficult to estimate the overall performance correctly. The performance of JavaEE based Application Servers has been studied in several different ways [17]. The majority of related work focusses on a specific Application Server while other researchers compare different Application Servers with each other ([18]). Further efforts have been carried out in the category of optimisation by identifying influence factors such as the thread pool size or the number of deployed beans [19]. Comparisons of cluster performances are investigated, for instance in [20]. Software analysis includes examination of different Enterprise Java Beans (EJBs) systems, types of beans, local vs. remote invocations, transaction and security options (e.g. [21]).

Performance analysis requires either load and stress tests on a real testbed or a performance model. Since load test are often non-trivial to produce in a realistic manner, performance models have their advantages in terms of complexity and simplicity. Several researchers have tried to build analytical models of an EJB environment based on queuing network models [22,23], Petri networks [24], Markov Models [25] or workflow discovery [26]. Apart from analytical models little contribution exists in the area of simulation of EJB based servers. Nevertheless some work has been presented, e.g. by Mc Guinness et al. [27] who have evaluated a DES model for multi-server EJB servers based on a Hyperformix Workbench.

3. Evaluation of the C-ProMiSE middleware

The *Context Provisioning Middleware with Support for Evolving Awareness* (C-ProMiSE) [28,29] has been designed to provide coherent access to manifold context information and to transparently support various application domains. Due to the gradual extendibility and self-management capabilities, it supports device, sensor, network, and application heterogeneity.

For context management, the well known producer–consumer role model is applied in conjunction with the broker software design pattern. Therefore, the following component types are defined:

- *Context Consumer* (CxC): Being a context sink, the CxC can request context either on-demand (synchronously) or based on subscription (asynchronously) and utilise it to adapt or actuate accordingly. Each context-aware application/service is categorised as a CxC.
- *Context Provider* (CxP): A CxP provides a synchronous interface for context queries. Each CxP supplies a so called context scope (e.g. location) and must be invoked with the required input parameters (e.g. WiFi signal strengths).
- *Context Source* (CxS): A CxS is the asynchronous counterpart of a CxP. It does not provide an interface for external invocation but asynchronously pushes context to a broker or storage service.
- *Context Broker* (CxB): The CxB has been introduced for mediating between CxC, CxP and CxS. Most importantly, it provides a CxP registry and lookup service and a proxy query service. Although a CxC may directly interact with a CxP, usually it will query with the CxB as single point of contact and utilise its functionalities. The CxB collects all required context on behalf of the CxC and aggregates it autonomously.

Interaction between the components is based on RESTful [30] interfaces, i.e. all components use the Hypertext Transport Protocol (HTTP). For representing context as well as coordination messages, the XML based Context Meta Language (ContextML) [31] is applied. Fig. 1 illustrates a specific prototype testbed that has been implemented to evaluate the design concepts.

3.1. Evaluation methodology

The analysis of the C-ProMiSE prototype testbed particularly aims at identifying parameters which effect and characterise the context query response delays of the central CxB and associated CxP components. In the scope of this article, the *User Profile Context Provider* (UserProfileCxP) is selected as the representative CxP. Testbed measurements are utilised to identify the response time influence factors. The purpose is to isolate an abstraction model which (1) describes the testbed and component behaviour adequately and (2) can be integrated in an already established functional simulation model of the C-ProMiSE architecture (cp. [2]). The results of the simulation can be compared with the testbed measurements afterwards in order to assess its correctness. This fosters an improved understanding of the simulation as well as the implementation approach and can be further utilised for rapid and quantifiable system improvements. The next subsection explains the testbed measurement methodology and the target system parameters.

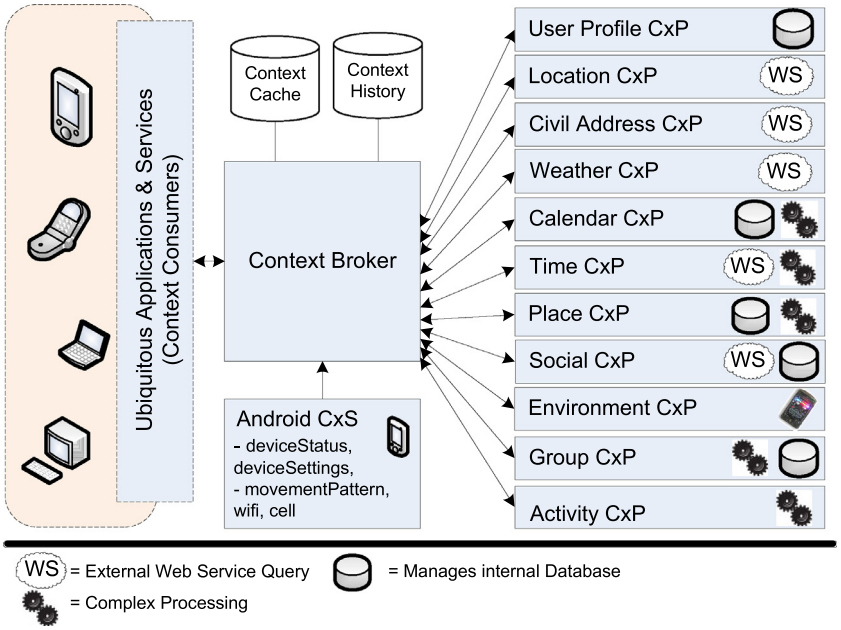

Fig. 1. C-ProMiSE prototype testbed.

The aim is to model the performance of a JavaEE environment in order to create a suitable simulation model including the C-ProMiSE functional logic on top of it. Therefore, black-box tests are applied. This allows measurement and simulation of the JavaEE environment without detailed knowledge and, more importantly, with a simplified model that can be divided into sub-models for further investigations. It is neither the goal to build a complex model nor to build a generally valid model. Hence, the measurement results and the models can not be easily transferred to other testbed and application environments. However, we derive and apply an evaluation methodology which is generally applicable and assess its suitability for a distributed context provisioning system.

As discussed in Section 3, the context request interfaces are based on a RESTful interface. Therefore, the *Apache Benchmark*[1] (AB) tool has been chosen to emulate context request. AB is capable of conducting performance tests for HTTP/HTTPS based requests and supplies information about the request-response delay. In addition several simultaneous requests can be emulated in order to identify the number of requests that can be served concurrently. Another tool that has been included in our evaluation is the GNU *wget*[2] package. The program is invoked within custom *Perl*[3] scripts we developed to create and delete database items based on HTML GET for evaluating the effect of database read/write access on the overall system performance.

4. Testbed measurements

Our investigations are focused on the CxB and the registered CxPs forms the functional backbone of the system; CxS and CxC processing is out of scope of this study. From the simulation modelling point of view, CxSs are only utilised to emulate and provide primitive context, CxCs are responsible for requesting context. The prototype testbed consists of two separate servers. One server hosts the CxB while the other one hosts the UserProfileCxP which has been selected as a typical CxP, and includes the databases access functionality. Both physical servers have the following hardware and software configuration:

- Intel Xeon CPU X3323 @2.5 GHz
- 4096 MB RAM
- Ubuntu 8.04.4 LTS Server
- JBoss 4.2.3.GA with 1024 MB Cache and max. 250 threads

The first step of our evaluation is to identify appropriate influence factors of our broker based framework. The initial parameter set is as follows:

- ContextML based context representation
- Database access time
- Load, i.e. number of concurrent requests

The context data is represented in the XML based ContextML [31] model, which comprises hierarchical compounds of simple, structured and arrayed context parameters. Accordingly, not only the size but also the complexity of the context document varies. The database access of the CxP and the CxB is also worth examining since both components use databases for managing persistent objects. The behaviour model of: (1) how many and, (2) at which delay concurrent requests can be served is our major investigation since it will play a key role when approximating the overall performance. The next subsection highlights the measured parameters of a typical CxP and a first abstraction model is introduced. Afterwards the CxB measurement results are presented.

4.1. Context Provider

The UserProfileCxP is able to add, change and delete profiles as well as to respond to profile context requests. It stores the user profiles in a MySQL database. The first measurement series identifies the influence of the profiles, i.e. the context, being requested. The measurement consists of 10,000 requests each and distinguishes between (1) querying the same context and (2) requesting different context instances. The resulting relative frequencies are shown in Fig. 2. The ordinate is limited to 1.5×10^{-3} for a better readability although the occurrence probability of the first interval between 4.875 and 6.625 ms is about 0.998. The results indicate that there is only statistical variance but no significant influence. Therefore, we clarified that the application server did not cache the requested context or the corresponding database values internally. This simplifies the design and generation of requests since there is no need to generate and query user profiles randomly; identical queries are sufficient.

[1] http://httpd.apache.org/docs/2.0/programs/ab.html.
[2] http://www.gnu.org/software/wget/.
[3] http://www.perl.org/.

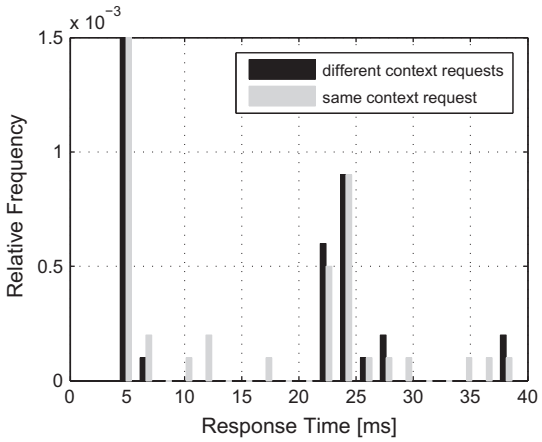

Fig. 2. Probability density function – prototype measured C × P response times. The graph examines the influence of requesting different or the same context.

Server based frameworks are optimised to handle several tasks and process queries at nearly the same time. Within C-ProMiSE, concurrent requests have to be processed in many situations. Multiple context requests may arrive at the CxP simultaneously. In addition to that, CxSs transmit context updates and CxPs register at the CxB with regular advertisement messages (see [28] for details). All these requests have to be processed at the same time. Fig. 3 illustrates the influence of concurrent requests on the CxP response time. The figure shows the relative frequency of the response times as a function of the number of concurrent requests. For reasons of clarity and readability the distribution envelope is plotted. Two effects can be observed: (1) an increasing number of concurrent requests results in a larger mean response time and (2) the deviation of the response time grows with an increasing number of concurrent requests. This is obviously expected since the server processing capabilities are shared autonomously amongst more threads if a larger number of concurrent requests occurs. The series of measurements with a low number of concurrent requests can be interpreted as an approximated normal distribution. Fig. 4 clarifies this interpretation with a selected series of measurements with 100 concurrent requests (100 *CR*). As illustrated, the utilisation of a single normal distribution is not adequate enough. Right next to the absolute maximum of the graph we identified a second local maximum whose amplitude increases with an increasing number of concurrent requests. Therefore, we decided to model the behaviour with two overlapping normal distributions which are separated, in this case (100 *CR*) at 154 ms. Eq. 1 describes the mathematical model with mean μ and standard deviation σ of the Gaussian distribution function N where x is a uniformly distributed random variable. The response time function $f(\mu_{i,n}, \sigma_{i,n})$ has been modelled as a function of the number of concurrent requests n. The comparison of simulation model and prototype measurement shows a satisfactory match (cp. Fig. 4).

$$f(\mu_{i,n}, \sigma_{i,n}) = \begin{cases} N(\mu_{1,n}, \sigma_{1,n}), x < \upsilon \\ N(\mu_{2,n}, \sigma_{2,n}), x \geqslant \upsilon \end{cases} \tag{1}$$

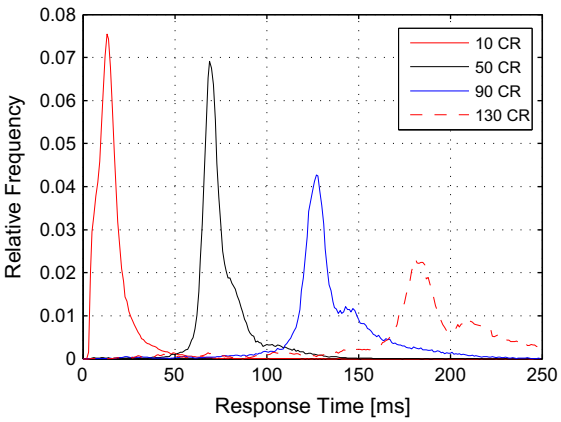

Fig. 3. Probability density function – measured C × P response times with different number of concurrent requests.

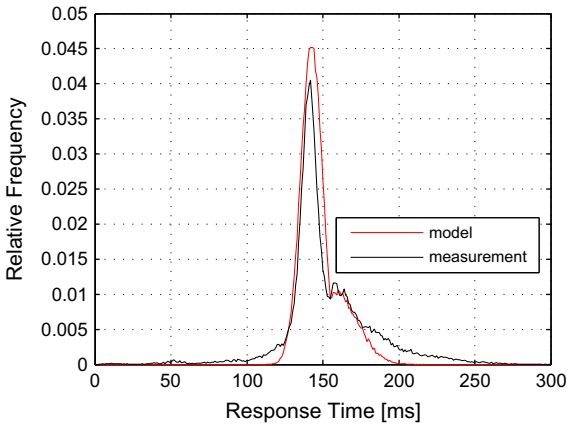

Fig. 4. Probability density function – measured and modelled C × P response time with 100 concurrent requests.

The results are valid for different context sizes (cp. Fig. 5). Two different measurements have been conducted; the first one reflects a minimal user profile (*min context size*) with a size of 1,388 Bytes. The second context instance contains a maximal user profile (*max context size*) with a size of 2461 Bytes. The minimal profile includes only basic profile data, i.e. forename and surname, email address and birthday and the maximal profile comprises additional profile information, including address, social network IDs, messenger IDs and self description. Both context instances are formatted in ContextML. The measurement results are almost equal between 10 and 160 concurrent requests. Afterwards the influence of side effects (e.g. the garbage collector and general performance limitations of the JavaEE environment) results in a higher variance but still indicates similar results between the *max context size* and the *min context size* profiles.

4.2. Context Broker

The measurements of the `CxB` focus on the proxy query mechanism, i.e. context requests are forwarded to the responsible `CxP`. The influences of an optional caching mechanism have been analysed in [2] and are out of scope of this article. Two different measurements are conducted with the *max context size* case applied in Fig. 5. One set of measurements is generated with an increasing number of concurrent requests (*ascending number of requests*) while the other one has been realised with a descending number of concurrent requests. The resulting mean response time is illustrated in Fig. 6. The measurements are aborted if the response time exceeds 30s which is often the case at the end of the measurement series. The measurements are stopped after 190 (*ascending number of requests*) and after 40 (*descending number of requests*) concurrent requests, respectively. The results of the *ascending number of requests* case reveal a smaller mean response time. This indicates that the permanent load has a high influence on the performance of the `CxB`. The *descending no. of requests* curve is selected for further simulations. This choice is made due to the assumption of the inter-arrival time of new requests (cp. [2]). In the simulation, the context requests are instantiated with an exponential distribution of the inter-arrival time according to a Poisson process. This results in likely occurrence of burst requests and seems to be modelled best with the *descending no. of requests*

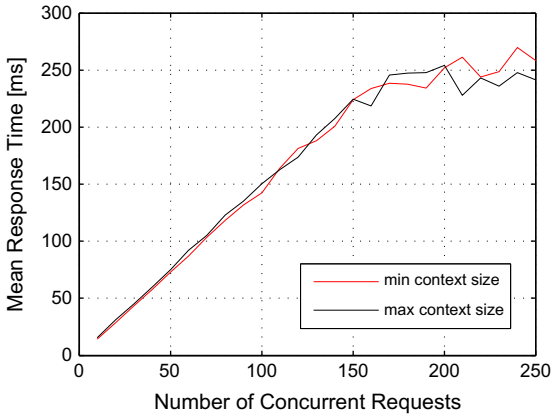

Fig. 5. Measured C × P mean response time for different context sizes.

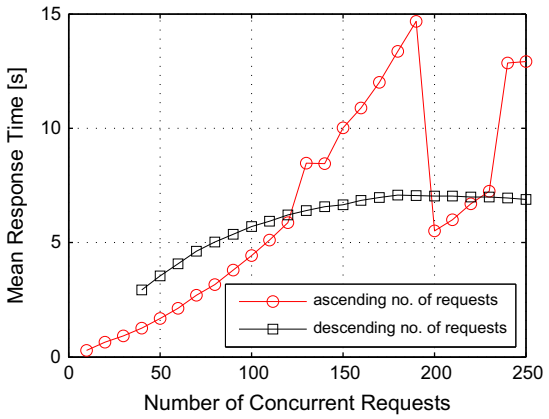

Fig. 6. Measured CxB mean response time with ascending and descending number of context requests.

run since there are high request rates at the beginning. However the model has some shortcomings: it might perform better than in reality if a large number of concurrent requests arrives and might be too pessimistic at low load situations. For a deeper analysis future measurements should also take CPU and RAM load into account.

5. Simulation model and results

5.1. Simulation environment

The simulation models have been implemented as OMNeT++ modules. OMNeT++ is a DES toolkit that offers an Eclipse based Integrated Development Environment (IDE). An OMNET++ simulation model generally consists of modules that can communicate via messages. Several components may be utilised to form a compound module. The modules are implemented in C++ while the simulation model structure (architectural design) is defined in an OMNeT++ specific language called NED. The concept of DES is realised based on messages that may be transmitted from one module to another or be self-messages triggering events in the future. This way, state transitions of a finite state automation can be represented. Fundamental events to start/stop the simulation as well as message arrival or module specific events are defined and triggered by individual modules. Different from classical communication engineering simulations which focus on OSI layer 1–5, the `C-ProMiSE` simulation model omits the detailed modelling of these layers. This is motivated due to the reason that the overall model is based on empirical models derived from prototype assessments. Therefore, the underlying protocol overhead is already included in the measured response time. Furthermore, the influence of the communication protocol stack is out of focus and assumed to be negligible for our purposes.

5.2. Overview of simulation model

The model overview of the C-ProMiSE simulation is shown in Fig. 7. According to the conceptual design the modules for the `CxS`, `CxC`, `CxP` and the `CxB` are instantiated. All modules interact via a common communication link with a channel

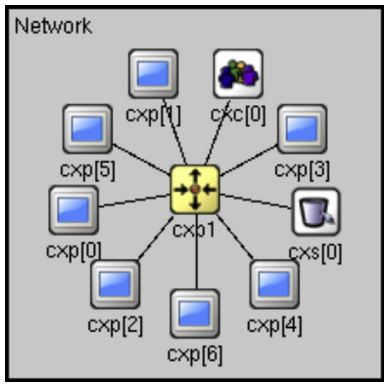

Fig. 7. Network simulation model.

whose delay can be defined. Logically the CxC, CxS and CxP are connected through the CxB module. For reasons of clarity the shown network does not reflect the whole simulation network; the simulated C-ProMiSE topology comprises 13 CxP and 7 CxS module instances.

The following subsections explain the simulation modules and the selected examples highlight how the models are derived from the testbed measurements. Table 1 summarises the most important simulation parameters.

5.3. Simulation modules

5.3.1. Context Consumer

The CxC module triggers the CxB module (that consecutively invokes the CxP modules) by sending context queries. Due to the assumed independence of requests from different CxCs the inter-arrival time of context requests is modelled to follow a Poisson process. Consequently the context requests occurrence time can be represented with an exponential distribution with a query rate $\lambda = \frac{1}{\mu_{\Delta t_Q}}$ where $\mu_{\Delta t_Q}$ refers to the mean time duration between two consecutive requests. Hence, all CxC are represented by one abstract CxC module. In addition to the inter-arrival rate, the content of the context request, i.e. the context query parameters *entityID* and *scopeID*, are highly relevant. By default the *entityID* follows a uniform distribution between zero and the maximum defined *entityID*. The *scopeID* is selected according to Zipf's law. This law can be interpreted as the discrete counterpart of the continuous Pareto distribution and takes into account that some context information (e.g position) might be far more requested than others. The CxC module is in charge of starting the simulation with the first context request and stopping the entire simulation after the configurable number of requests have been processed. The context responses from the CxB are stored in terms of delay and success/failure for statistical analysis.

5.3.2. Context Source

The CxS module generates context instances randomly and sends them to the CxB. One CxS module represents all simulated *entityID*s and produces all context instances of the specified scope consequently. Its settings (cp. Tables 1 and 2) are inspired from the prototype testbed. The CxS module has been included for the sake of completeness and is required for investigations with regard to the context caching mechanism of the CxB.

5.3.3. Context Provider

The overall simulation model comprises a configurable number of CxP modules, all being equipped with a communication gate. Their abstract model covers (1) registering with the CxB module by sending an advertisement message periodically and (2) responding to synchronous context requests originating from the CxB. Each CxP module instance is associated to one context scope. The most important parameters are outlined in Tables 1 and 3. The parameter set comprises the output context scope, a list of input context scopes it depends on and the expiration time of the context instance. Moreover, a failure rate is defined to take into account that a provider might not be able to supply context for all entities.

Upon reception of a message, a CxP module calculates the response delay and creates a corresponding event. Due to the observed influence of concurrent requests, the processing time has to be recalculated each time a new request arrives or a request leaves the request queue. This procedure is sketched in Fig. 8. Upon receipt of a message the CxP module distinguishes between messages sent by the CxB and by itself. If a context request for a specific entity is received the CxP will

Table 1
Selected simulation parameters.

Mod.	Nom.	Default value (s)	Description
CxC	N	10^5	Number of context requests
	$p(S)$	p_{zipf}	PDF for random selection of the queried *scopeID*
	$p(E)$	$p_{uni}(E;N)$	PDF for random selection of the *entityID*
	$p(\Delta t_Q)$	$p_{exp}(\lambda = 0.2/s)$	PDF for random calculation of the query inter-arrival time
CxS	$ID(s)$	–[a]	ID of the scope provided by the source
	$S(id)$	–[a]	Size of the ContextML document
	$\Delta T_{valid}(id)$	–[a]	Context instance validation duration
	$\Delta T_{update}(id)$	–[a]	Interval between two consecutive context updates
	$p_{fail}(id)$	–[a]	Probability of erroneous context instance replies
CxP	$ID(s)$	–[a]	ID of the scope provided by the provider
	$S(id)$	–[a]	Size of the ContextML document
	$\Delta T_{valid}(id)$	–[a]	Context instance validation duration
	$ID = \{id_i, id_{i+1}, \ldots\}$	–[a]	IDs of the scopes the CxP depends on
	$\Delta T_{advert}(id)$	120*s*	Duration between two (keep-alive) advertisements
	$f_{resp}(id, cc)$	–[a]	PDF used to calculate the response delay
	CC_{max}	250	The number of requests the provider is capable of processing simultaneously
	$p_{fail}(id)$	–[a]	The probability of erroneous replies, i.e. a NACK is returned instead of a context instance
CxB	$d_{pq}(cc)$	–[a]	Distribution determining the delay of forwarding proxy queries

[a] The default values are read from a configuration file and depend on the specific module instance. See Tables 2 and 3 for details.

Table 2
Default parameters of C × S simulation modules.

Scope name	Scope ID[a]	CML validity (s)	CML size (B)	Failure prob.	Update interval (s)
DeviceStatus	5	600	1056	10^{-4}	600
TasksInfo	8	300	2507	10^{-4}	300
DeviceSettings	9	600	1002	10^{-4}	600
Motion	16	120	950	10^{-2}	120
WiFi	17	300	1482	5×10^{-2}	300
Cell	19	600	787	10^{-3}	600
BT	20	300	789	2×10^{-2}	300

[a] The *ScopeID* is particularly relevant. Due to the application of Zipf's law, a lower ID increases the number of context queries for this specific scope.

Table 3
Default parameters of C × P simulation modules.

Scope name	Scope ID[a]	Input scope IDs	CML validity (s)	CML size (B)	Failure prob.
Position	1	14,17,19	300	748	10^{-2}
UserProfile	2		1800	1711	10^{-4}
CivilAddress	3	1	600	867	10^{-2}
Place	4	1	600	2995	10^{-2}
Time	6	3	60	1092	10^{-2}
Activity	7	2,4,6,16	120	982	10^{-2}
Weather	10	3	1800	1779	10^{-2}
Group	11	–[b]	300	654	5×10^{-2}
Environment	12	14	300	1032	10^{-3}
Social	13	2	600	2174	10^{-2}
Indoorposition	14	17	300	784	10^{-3}
Proximity	15	1	300	850	10^{-3}
Music	18	2	1800	3733	10^{-2}

[a] The Scope ID is highly relevant. Due to the application of Zipf's law, a lower ID increases the number of context queries for this specific scope.
[b] Instead of synchronous invocation with context parameters, this `CxP` acquires context as `CxC`.

either calculate the response time or send a Negative Acknowledgement (NACK) based on the defined failure rate $p_{fail}(id)$. In case of context availability, a new request is added to the query queue – together with a processing progress value and an estimated response time. In addition, the query queue is updated for each context request. Afterwards, the context request with the smallest estimated response time is selected in order to trigger the next self-message. This self-message enables the sending of the context response at the calculated simulation time and also ensures that the query queue entries will be updated at this point in time.

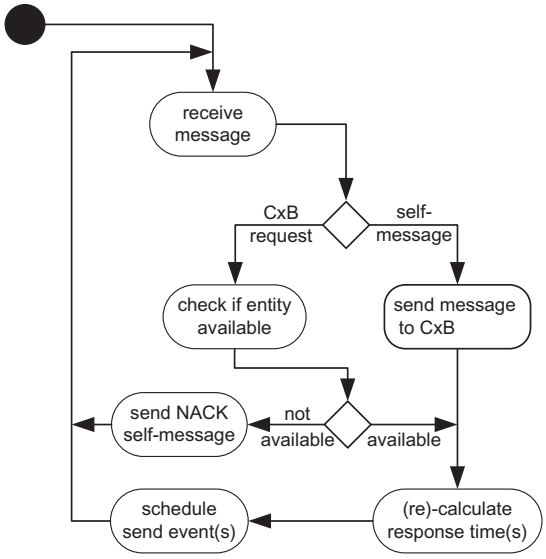

Fig. 8. Simulation model of the C × P context response.

As outlined in Section 4.1 the CxP response time is only strongly influenced by the number of concurrent requests. Therefore, the calculation of the estimated response delay is approximated with two overlapping normal distributions in order to generate Figs. 3 and 9. A linear regression is utilised to abstract the simulation model and ensure a continuous function between 1 and 250 concurrent requests.

5.3.4. Context Broker

The CxB response delay is modelled with two influence factors: (1) the number of concurrent requests and (2) the size of the requesting context (ContextML size). The first parameter is modelled with a normal distribution and the associated mean and standard deviation. Fig. 10 illustrates the standard deviation of the measurements and the approximation of the *descending no. of requests* curve with a third order polynomial. The size of the requesting context influences the delay of the response. In contrast to the CxP, the CxB needs to interpret and react to the content of the ContextML encoded context in order to fulfil the CxC request. The influence of the context size of the response delay is modelled with a linear equation.

The internal processing of the CxB module in terms of context request-response shares some similarities with the CxP module. In both modules the progress and the estimated response time of each context request is stored in a queue and each time a new element enters or leaves the queue the progress and the estimated response time for each context request is (re)-calculated. Unlike the CxP module the CxB stores previous context responses from the CxP and CxS and makes it possible to respond to context requests from the cache without invoking the corresponding CxP. This application flow is also modelled within the simulation but the cached context response time is not discussed in this article.

5.4. Simulation results

The system-level simulation is implemented based on *OMNeT++*. A functional and a performance model of CxC, CxS, CxB, and CxP have been designed. In addition, the CxC is extended to a request model which is similar to the AB tool in order to prove the correctness of the investigated simulation model. The simulation is conducted with an increasing concurrent request amount comparable to the measurement with 10,000 requests per run and is repeated 25 times. Fig. 11 illustrates the simulated response delay of the CxP as a function of the number of concurrent requests. The curve represents the envelope distribution of the response time. The characteristics clearly reveal a strong analogy to the measured values outlined in Fig. 3. Nevertheless the decreasing of the maximum at the occurrence probability is more likely to be exponential than linear. Influences either from the simulation environment or imperfect implementation can be the reasons for this circumstance.

Similar results can be observed from the simulation of the CxB. The simulated mean response times are shown in Fig. 12. The curve *simulation* is compared with the measurements taken in the prototype testbed. The simulated curves have a slightly larger mean response time. This is caused by the fact that the CxB has modified the ContextML frame slightly at the testbed measurement resulting in a larger size. This functional step is not modelled in our simulator.

Our proposed model has been shown that black-box testbed measurements can be utilised to build a performance model of the application server and the investigated context provisioning middleware. Nevertheless, the model of the CxP is based on the UserProfileCxP and even though many CxPs have similar architectures in terms of database interaction, modelling of context, communication interfaces, etc., it is not expected that the performance is equal. We believe that the simulation model of the UserProfileCxP can be easily adopted in order to model different CxPs and we are testing this hypothesis in our ongoing work. A large-scale evaluation of different types of CxPs (web based, database-centred, involving complex processing etc.) and CxB under different load conditions (e.g. concurrent requests) and features (e.g. cache enabled) is presented in [32]. This large-scale evaluation, which is carried out both as a system-level simulation and in a real-world middleware

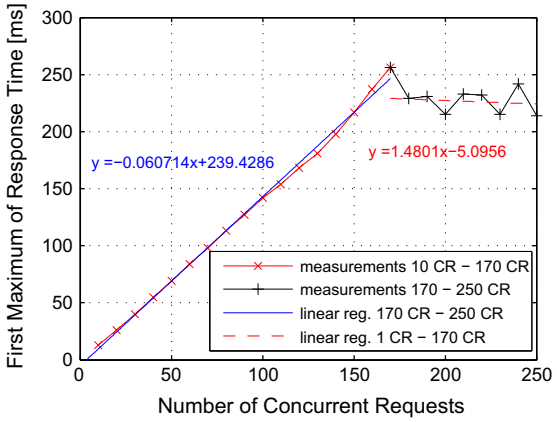

Fig. 9. Analytical model based on the empirical measurements of the first maxima of the C × P response times.

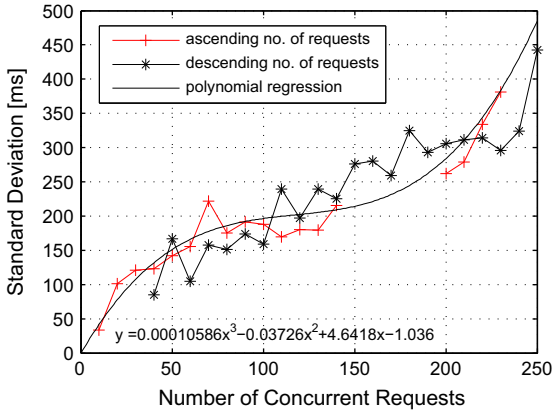

Fig. 10. Empirical model based on measurements of the standard deviation of the CxB.

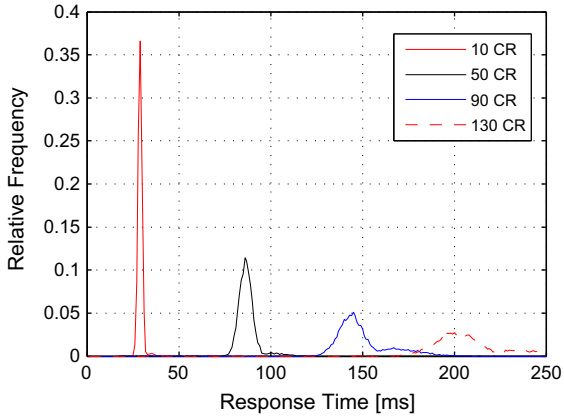

Fig. 11. Probability density function – simulated response times of the Profile-C × P.

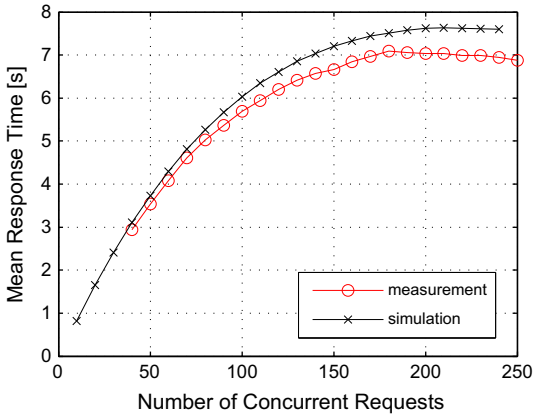

Fig. 12. Comparison between *simulated* and *measured* mean response time of the C × B.

deployment (cp. [32, Chapter 6]), reinforces the conclusions drawn from the work reported in this article i.e. the utility of black-box testing for abstracting testbed performance models and combining prototype assessment with discrete event simulation for estimating system-level performance.

6. Conclusion and outlook

This article has argued the need for quantitative evaluation of context provisioning systems since related work is often restricted to qualitative, i.e. functional, evaluation. Our experiments lend weight to the argument that prototype assessment and discrete event simulation can be combined in order to estimate the system-level performance of such a middleware or framework.

Specifically, measurements from a real testbed implementation have been used in order to design and build realistic simulation models. The measurements methodology is based on black-box tests and key influence parameters of the context query response performance of the Context Provider (CxP) and the Context Broker (CxB) have been identified. One key parameter has been identified for CxPs and two for the CxB. The resulting response time of the CxP is modelled with two overlapping normal distributions as a function of the concurrent requests. The mean and standard deviation of the response time of the CxP is simulated with polynomial functions up to third order. In addition to the influence of the concurrent requests, the CxB also relies on the size of the requested context size. The proposed abstraction is implemented and evaluated within the event based simulation environment OMNeT++. The results indicate a large match with the testbed measurements and prove that black-box tests can be utilised to abstract controllable and adequate models of testbed performance.

The introduced simulation and evaluation process contributes to a better understanding and validation of functional and architectural models in the area of context provisioning. However, our evaluation has only employed a limited number of interacting components, which provides a functionally complete system for analysis but we envision that there will be a considerably large number of such components in practical deployments of the CPS on top of the *IoT* technologies. Therefore, our primary efforts are currently directed towards expanding the scale of our simulation and prototype system, and analysing the effects of scale on our evaluation model and conclusions drawn from this study. Specifically, a realistic large-scale evaluation with exemplary context request and processing characteristics of our system is presented in [32].

Furthermore, there are emerging technologies and platforms that may assist in improved deployment-time performance and scalability of a complex software system, e.g. Cloud platforms [33]. A similar evaluation of the CPS in a Cloud deployment can be carried out to assess the suitability of Cloud-based context provisioning, which poses an interesting question regarding the compromise between the need for scalability/cost effectiveness (rendered by the Cloud platform [34]) and the overhead of additional middleware layers in the system.

Our evaluation has only considered an isolated deployment of CPS with a single CxB. The computing resources in the *IoT* environment are more than likely to fall under different administrative authorities, giving rise to issues of privacy, security, ownership association and the collaboration between different administrative domains. These issues can be coordinated by using a federation of CxBs, under different administrative domains, which apply administrative policies for resource sharing and coordinate communication amongst remote components across the domain boundaries. This arrangement not only induces administrative overhead in the component interaction, but also increases the scale at which the consumer–broker–provider interaction takes place. The evaluation of such a large-scale deployment of a context provisioning middleware is also a target of our future work.

References

[1] M. Weiser, The computer for the 21st century, in: Human–computer interaction: toward the year 2000, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995, pp. 93–940.

[2] S.L. Kiani, M. Knappmeyer, E.S. Reetz, N. Baker, R. Tönjes, Effect of caching in a broker based context provisioning system, in: Proceedings of the 5th European conference on Smart Sensing and Context, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 108–121.

[3] H.A. Henning, A., J. Tyack, Performance prototyping - generating and simulating a distributed IT-system from UML models, in: Proceedings of the 17th European Simulation Multiconference, ESM'03, Nottingham, UK, June 2003.

[4] M. Knappmeyer, S. Kiani, N. Baker, A. Ikram, R. Tönjes, Survey on evaluation of context provisioning middleware, in: Proceedings of the Second Workshop on Context-Systems Design, Evaluation and Optimisation, in conjunction with the 24th International Conference on Architecture of Computing Systems (ARCS), VDE VERLAG GmbH, 2011.

[5] B. Schilit, M. Theimer, Disseminating active map information to mobile hosts, IEEE Network 8 (1994) 22–32.

[6] R. Want, A. Hopper, V. Falc ao, J. Gibbons, The active badge location system, ACM Transactions on Information Systems 10 (1) (1992) 91–102.

[7] G.D. Abowd, C.G. Atkeson, J. Hong, S. Long, R. Kooper, M. Pinkerton, Cyberguide: a mobile context-aware tour guide, Wireless Networks 3 (5) (1997) 421–433.

[8] J. Froehlich, M.Y. Chen, S. Consolvo, B. Harrison, J.A. Landay, MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones, in: MobiSys '07: Proceedings of the 5th International Conference on Mobile Systems, Applications and Services, ACM, New York, NY, USA, 2007, pp. 57–70.

[9] S. Carter, J. Mankoff, J. Heer, Momento: support for situated UbiComp experimentation, in: CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, 2007, pp. 125–134.

[10] S.S. Intille, J. Rondoni, C. Kukla, I. Ancona, L. Bao, A context-aware experience sampling tool, in: CHI '03 Extended Abstracts on Human Factors in Computing Systems, ACM, Ft. Lauderdale, Florida, USA, 2003, pp. 972–973.

[11] Siafu: An Open Source Context Simulator. <http://siafusimulator.sourceforge.net/> (last accessed 27.02.12).

[12] J. Barton, V. Vijayaraghavan, A Simulator for Ubiquitous Computing Systems Design, Tech. Rep. HPL-2003-93, Hewlett-Packard Labs, 2003.

[13] E. O'Neill, M. Klepal, D. Lewis, T. O'Donnell, D. O'Sullivan, D. Pesch, A testbed for evaluating human interaction with ubiquitous computing environments, in: Proceedings of the 1st International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom), IEEE, 2005, pp. 60–69.

[14] S. Jang, Y. Lee, W. Woo, CIVE: Context-based interactive system for distributed virtual environment, in: Proceedings of the 14th International Conference on Artificial Reality and Telexistence, ICAT, 2004, pp. 495–498.

[15] H. Nishikawa, S. Yamamoto, M. Tamai, K. Nishigaki, T. Kitani, N. Shibata, K. Yasumoto, M. Ito, UbiREAL: Realistic smartspace simulator for systematic testing, in: P. Dourish, A. Friday (Eds.), UbiComp 2006: Ubiquitous Computing, vol. 4206 of Lecture Notes in Computer Science, Springer, Heidelberg, Berlin, 2006, pp. 459–476.

[16] J. Barbosa, R. Hahn, D. Bonatto, F. Cecin, C. Geyer, Evaluation of a large-scale ubiquitous system model through peer-to-peer protocol simulation, in: Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications, IEEE, 2007, pp. 175–181.
[17] D.M. Guinness, Performance Modelling of an EJB System: A Simulation Approach, Master's Thesis, Department of Computer Sciences, University College Dublin, August 2005.
[18] E. Cecchet, J. Marguerite, W. Zwaenepoel, Performance and scalability of EJB applications, ACM SIGPLAN Notices 37 (2002) 246–261.
[19] Y. Liu, I. Gorton, A. Liu, S. Chen, Evaluating the scalability of Enterprise JavaBeans technology, in: Proceedings of the 9th Asia–Pacific Software Engineering Conference, IEEE, IEEE Computer Society, Los Alamitos, CA, USA, 2002, pp. 74–83.
[20] I. Gorton, A. Liu, Evaluating the performance of EJB components, IEEE Internet Computing 7 (2003) 18–23.
[21] A. Stylianou, G. Ferrari, P. Ezhilchelvan, A comparative evaluation of EJB implementation methods, in: 10th IEEE International Symposium on Object and Component Oriented Real–Time Distributed Computing (IOSRC), IEEE, IEEE Computer Society, Los Alamitos, CA, USA, 2007, pp. 204–213.
[22] S. Kounev, Performance modeling and evaluation of distributed component-based systems using queueing Petri Nets, IEEE Transactions on Software Engineering 32 (2006) 486–502.
[23] W. Xiong, T. Altiok, An analytical approach for performance analysis of J2EE application servers. <http://ie.rutgers.edu/resource/research_paper/paper_07-016.pdf> (last accessed 27.02.12).
[24] F.N. Souza, R.D. Arteiro, N.S. Rosa, P.R.M. Maciel, Using stochastic Petri Nets for performance modelling of application servers, in: Proceedings of the 20th International Conference on Parallel and Distributed Processing, 2006, pp. 332–332.
[25] N. Sharifimehr, S. Sadaoul, Markovian workload modeling for enterprise application servers, in: Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering, ACM, New York, NY, USA, 2009, pp. 161–168.
[26] P.C. Brebner, Real-world performance modelling of enterprise service oriented architectures: delivering business value with complexity and constraints, in: Proceedings of the 2nd Joint WOSP/SIPEW International Conference on Performance Engineering, ACM, New York, NY, USA, 2011, pp. 85–96.
[27] D.M. Guinness, L. Murphy, A simulation model of a multi-server EJB system, in: ACM SIGSOFT Software Engineering Notes, vol. 30, ACM, 2005, pp. 1–7.
[28] M. Knappmeyer, N. Baker, S. Liaquat, R. Tönjes, A context provisioning framework to support pervasive and ubiquitous applications, in: Proceedings of the 4th European Conference on Smart Sensing and Context (EuroSSC), Springer-Verlag, Berlin, Heidelberg, 2009, pp. 93–106.
[29] M. Knappmeyer, S.L. Kiani, R. Tönjes, N. Baker, Modular context processing and provisioning: prototype experiences, in: Proceedings of the 4th ACM International Workshop on Context-Awareness for Self-Managing Systems, CASEMANS '10, ACM, New York, NY, USA, 2010, pp. 8:53–8:58.
[30] R.T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Ph.D. Thesis, University of California, 2000.
[31] M. Knappmeyer, S.L. Kiani, C. Frá, B. Moltchanov, N. Baker, ContextML: a light-weight context representation and context management schema, in: Proceedings of IEEE International Symposium on Wireless Pervasive Computing, ISWPC'10, IEEE Press, Piscataway, NJ, USA, 2010, pp. 367–372.
[32] M. Knappmeyer, A Context Provisioning Middleware with Support for Evolving Awareness, Ph.D. Thesis, University of the West of England, Bristol, UK (January 2012).
[33] I.A. Moschakis, H.D. Karatza, Enterprise HPC on the clouds, in: Z. Mahmood, R. Hill (Eds.), Cloud Computing for Enterprise Architectures, Computer Communications and Networks, Springer, London, 2011, pp. 227–246.
[34] I. Moschakis, H. Karatza, Evaluation of gang scheduling performance and cost in a cloud computing system, The Journal of Supercomputing 59 (2012) 975–992.