Performance modeling and analysis of the Universal Control Hub

Elena Gómez-Martínez[1] and José Merseguer[2]

[1] R & D Department, Fundosa Technosite - ONCE Foundation
C/ Albasanz, 16 28037 Madrid, Spain.
`megomez@technosite.es`

[2] Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza
C/María de Luna,1 50018 Zaragoza, Spain.
`jmerse@unizar.es`

Abstract. People with special needs may find difficulties using electronic consumer devices, user interfaces limit their chances of having full control on them. The Universal Remote Control (URC) is an ISO standard that promotes pluggable and interoperable user interfaces to remotely operate electronic devices. The Universal Control Hub (UCH) is the software architecture that materialises URC, and several implementations are currently available. However, users and developers wonder about UCH feasibility to respond to future needs regarding performance. In this paper, we conduct a study to analyze whether UCH can face multiple concurrent users. Serious problems are exposed at this regard in this paper, they may contribute to question a solution that initially and from the interoperability point of view was very-well suited.

1 Introduction

In industrial societies, people massively use electronic devices in everyday life: mobile phones, TV sets or washing machines are some of several examples. Nevertheless, their use may became very complicated, and even impossible, to people with special needs, such as impaired or elderly people. Their user interfaces are not generally designed considering their needs neither *Design for all* principles [16]. According to last reports of Eurostat office [6], the majority of the European countries own more mobile subscriptions than inhabitants. Internal studies of ONCE[3] foundation demonstrates similar trends for people with disabilities. So, the achievement of moving the proper control of electronic devices to adapted devices (e.g., mobile phone) may solve most user interface accessibility issues. Therefore, interoperability is critical to realizing the vision of personalized and pluggable user interfaces for electronic devices and services. An International Standard on pluggable user interfaces has here a key role to play, Universal Remote Console (URC) [13]. Such a standard would facilitate

[3] National Organization of blind people in Spain.

user interfaces that adapt or can be adapted to user's personal needs and preferences. It would allow easy to use interfaces that employ various modalities for input and output.

Limitations in URC advised to develop an architecture called Universal Control Hub (UCH) [26] to make URC practical in real scenarios. In short, UCH is a URC realization that acts as a gateway for communicating devices. UCH has been implemented, using different languages and technologies [24, 23], and currently is offering adequate and interoperable service within environments of a reduced number of users. However, for UCH is not only interoperability the critical issue, performance is or will be a must when the amount of plugged devices depletes the infrastructure to the point of exhausting its resources. This may happen in real scenarios such as intelligent buildings where hundreds or thousands of users will concurrently access the deployed architecture. Although UCH and underlying implementations have not been tested in such environments, it is a goal for [11] to assess if UCH can offer quality of service in these interesting settings. Note that the costs (in budget as well as in technical difficulties) prevent the architecture from testing in the new proposed environments, then in our opinion the use of predictive performance models can play a role in this context. Besides, in case of identifying adversities that turn UCH into a non practicable solution, the testing investment would be a waste in resources. So, the assessment should be also useful to pinpoint where in the UCH architecture are the problems located, again predictive models can offer cheaper solutions than real experimentation.

The study of the performance of UCH in future real situations is then a necessity that we will carry out using the formal model of Petri nets [1] in the context of the PUMA [25] methodology. The contributions of this paper are: *(i) the use of PUMA in a real-complex case study in the industrial setting and (ii) the assessment of the performance of the UCH architecture in situations where over budget hampers the benefits of the evaluation.* To the best of our knowledge, PUMA has been applied in examples or academic studies, but not in an industrial setting.

The rest of the paper contains the following sections. Section 2 gives details of the URC-UCH architecture to understand potential performance problems it may provoke. Section 3 recalls the PUMA methodology we have followed to analyze UCH. Section 4 follows the suggested steps in previous section for us to gain a performance model for UCH. From the performance model, we propose, in section 5, a set of experiments that will allow to compare their results with results obtained from current real implementations, then they will validate the performance model. From the validated model, the system will be tested to assess its usefulness for future necessities above described. The paper ends recalling the lesson learned from the system analysis, proposing future work and giving a conclusion.

2 Interoperable Architecture

The Universal Remote Console (URC) is an International Standard published in 2008, ISO/IEC 24752 [13]. URC describes an interoperable architecture with a set of elements that allow users to control one or various devices by means of a remote console, in a transparent way for them. So, URC, or remote console, defines a framework of remote access to control devices or services. It can be designed both, as a dedicated hardware (e.g., a universal remote control), or as a URC-complaint software to run on specific devices such as personal computer, PDA or mobile phone. Therefore, it is a device or software architecture (gateway) through which the user accesses other devices, then being capable of rendering its user interface. This fact allows to develop adaptable user interfaces, which can satisfy users with special needs. In the following, the devices or services that the user wants to control are referred as *targets*, and the *controller* may be any user device. For instance, a blind person can control the washing machine, in this case the target device, by means of his/her mobile phone (controller device). So, URC allows to show washing machine functionalities in accessible manner.

ISO/IEC 24752 does not impose how it must be implemented. Besides, it does not assume a specific network protocol between controller and targets, but only network requirements. So, a URC interaction could be implemented on top of existing networking platforms as long as they support device discovery, control and eventing, such as UPnP (universal plug and play), Web services and/or HomePlug (IEEE 1901 [10]). Among others, URC defines the following XML documents: *Target Description* (TD) and *User Interface Implementation Description* (UIID). The TD document permits the remote console to learn how to use the target device, locate its functionalities, current status, and other interesting information. The main advantage of UIID is that delivers a generic user interface, so the remote console can implement it under the most adaptive way to the user (optical, audible, tactile), addressing *Design for All* principles [16]. Nevertheless, URC presents some issues: lack of devices with URC technology, lack of plugging in several targets and multiplicity of communication protocols.

The Universal Control Hub (UCH) [26] architecture fixes some of the above mentioned problems. Indeed, UCH is seen as an "open box" between the target and the controller, acting as gateway between various controllers and various targets, which overcomes communication limitations of URC. Basically, UCH is a manner to implement URC, that focusses on normalizing how the Control Hub works. So, UCH defines APIs and interfaces between internal modules of Control Hub, inheriting the URC XML documents. Figure 1 depicts the components in UCH:

- **User Interface Protocol Module** (UIPM): is a "pluggable user interface" that specifies a protocol between the controller and the Socket Layer via an API. URC-HTTP protocol is a UIPM specification based on HTTP.
- **Socket Layer**: is the core part of UCH, hosting the sockets of the targets.
- **Target Adaptor** (TA): synchronizes one or multiple targets with their sockets (running in the Socket Layer). TAs can be dynamically loaded at runtime.

– **Target Discovery Module** (TDM): discovers specific targets, connects to the Socket Layer via API, and to the targets via any protocol. TDMs can also be dynamically loaded at runtime.
– **UIList**: contains a dynamic list of available user interfaces, as given by the currently loaded UIPMs.

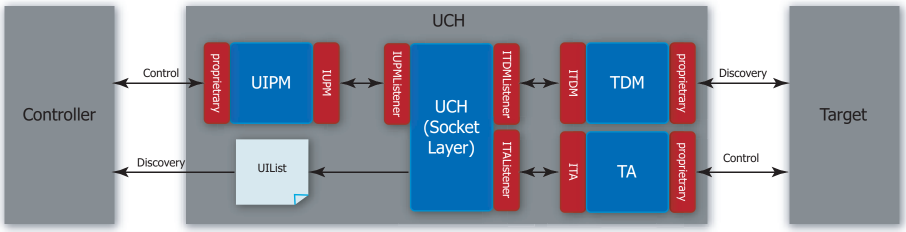


Fig. 1. Components of UCH architecture taken from [21].

Currently, there are three implementations of UCH, two of them developed under open source: UCHj [24] and UCHe [23], and another one under proprietary software. UCHj is a Java implementation designed for a closed delimited network, such as an office or home. UCHe is developed in C/C++ for embedded systems. Recently, a UIPM client for iPhone smart phone has been published [22], however this one does not implement UCH core.

These different implementations could seriously affect performance in a scenario with concurrent users. As URC and UCH are based on exchanging XML messages, they suggest poor performance, as previous studies have observed [5, 4, 9]. Since both UCHj and UCHe implement UIPM on HTTP, then UIPM performance should be also taken into account. Moreover, dynamic loading of modules (TA, TDM) will impact system performance. Considering that the Socket Layer is the UCH core module, then it will play a decisive role from a performance point of view, since it is attending all system requests.

3 Performance by Unified Model Analysis

UCH and related implementations comprise a complex software for which, as above described, their performance was considered critical in project [11]. Complexity advised to carry out the evaluation from different points of view, so to allow comparison, gain insights on the products and also validate results. Therefore, it was decided that performance of current implementations should be traced both, experimentally and within a benchmark approach [2], but also it was pointed out the interest of an evaluation with formal methods, hence to be able to test the system not only in its current form but under future variations (mainly concurrent users). Among available choices we decided to use

Performance by Unified Model Analysis (PUMA) [25] based on different reasons. Firstly due to our previous experience with it, also because of the existence of related tools that may simplify its application, and finally because PUMA incorporates formal methods within a methodological and pragmatic framework.

PUMA is a methodology for the performance evaluation of software systems. It allows different kinds of software design models (first and foremost, UML diagrams) as sources, and different kinds of performance models (e.g., queuing networks or Petri nets) as targets. PUMA uses an intermediate performance model, called Core Scenario Model [19] , as a bridge among sources and targets, then smartly solving the problem of translating N sources into M targets.

PUMA uses the standard UML Profile for Schedulability, Performance and Time Specification (UML-SPT) [18]. It introduces stereotypes and tagged values that can be applied to design model elements in the UML diagrams, specially in the behaviour and deployment specifications. UML-SPT allows to describe the input performance values of the system and also the metrics that will characterize the performance analysis.

The Core Scenario Model (CSM) [19] is based on the domain model of the UML-SPT. The benefits of intermediate models, which are discussed in [19], basically bring the choice to be transformed into different formal models. CSM is focused on describing *performance scenarios*. A scenario is a sequence of *Steps*, linked by *Connectors* that include sequence, branch/merge, fork/join and *Start* and an *End* points, where it begins and finishes. A step is a sequential piece of execution. A start connector is associated with a *Workload*, which defines arrivals and customers, and may be open or closed. There exist two kinds of *Resources*: *Active*, which execute steps, and *Passive*, which are acquired and released during scenarios by special *ResAcquire* and *ResRelease* steps. Steps are executed by (software) *Components* which are passive resources. A primitive step has a single host processor, which is connected through its component.

Petri nets (PN) are a graphical and mathematical modelling tool for describing concurrent systems. We used a temporal extension, the class of Generalized Stochastic Petri Nets (GSPN) [1], which distinguishes three kind of transitions: immediate transitions; transitions with probabilities; and transitions with exponentially distributed random firings.

Regarding automation, it is worth noticing that PUMA offers tools to translate a UML-SPT annotated model into CSM models [19], and also a tool to translate from CSM models into GSPN [3].

4 Performance Model

The construction of the performance model was conducted as PUMA indicates, i.e., firstly identifying the scenarios that represent the common usage of the system. UML was the design notation used.

4.1 Design Models

For an initial understanding and in order to determine the interactions that mostly affect system performance, we start summarizing the necessary steps to control a *target* device, see sequence diagram in Figure 2.

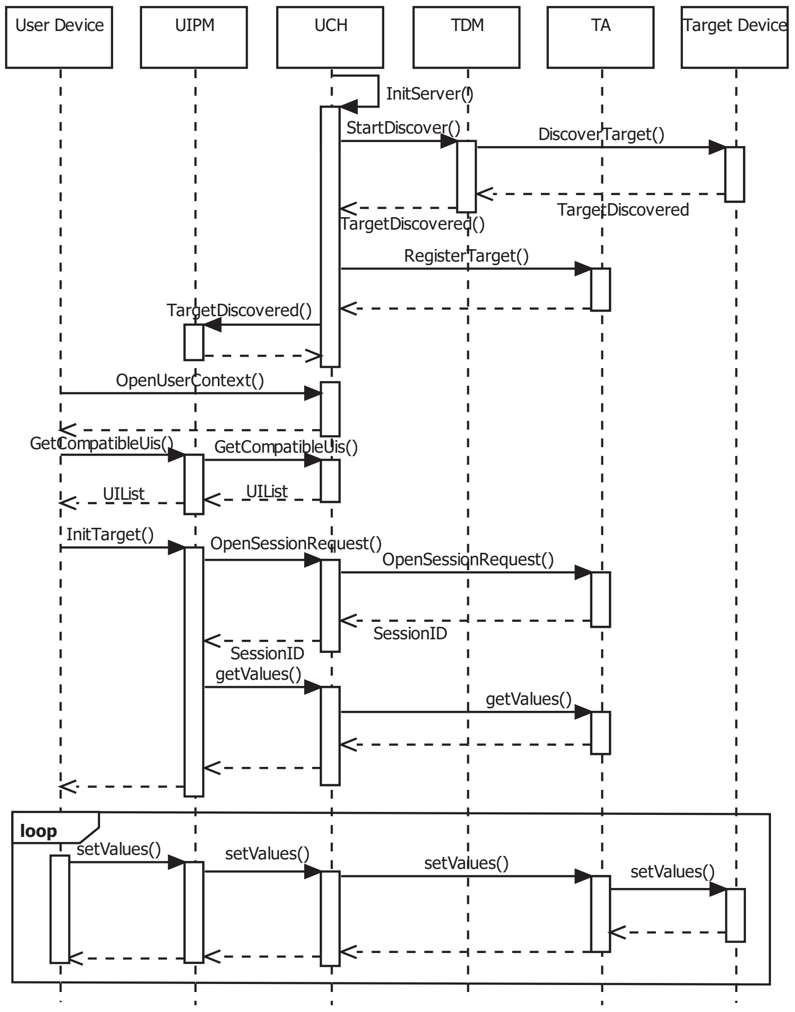


Fig. 2. Sequence diagram summarizing the target device control process.

In a first step, the UCH core is initialized and then it discovers and registers connected target devices by means of TDM module. Targets are listed (UIPM) as accessible devices for users to eventually manage their services. Then UCH waits for requests from user devices. When a request arises and compatibility is checked, the UIPM module opens a session and obtains the target devices list and corresponding services or functionalities which are granted in the form of a list (UIList document), that is eventually shown in the user interface. Hitherto, the system has performed two complex processes, discovery and user interface auto-adaptation, that obviously spend a considerable amount of time and resources. However, we will leave them out of our performance study since they are executed only once, i.e., they are the equivalent to start up the system, and all we understand the need for this process and its implications. So, we assume that from now on, the user is able to control the target device (i.e., to invoke commands through `setValues` message), which also means to modify the device status and variables. Indeed, this is the normal usage of the system and it repeats as many times as invoked commands (as indicated by the loop in the diagram), besides, several concurrent users (all those initialized in the system) will be executing. Then this loop interaction turns to be the performance critical part of the system.

Sequence diagram in Figure 3 models how a user requests a *target* by means of `setValue()` operation, i.e., it details the previous critical loop. Firstly, the User Device communicates to UCH core by means of UIPM via URC-HTTP protocol. The Socket Layer module, i.e. UCH core, connects to TA module in order to send a `setValue()` request to the Target Device. Once the request is made, the response is rendered in the User Device in an adaptive way.

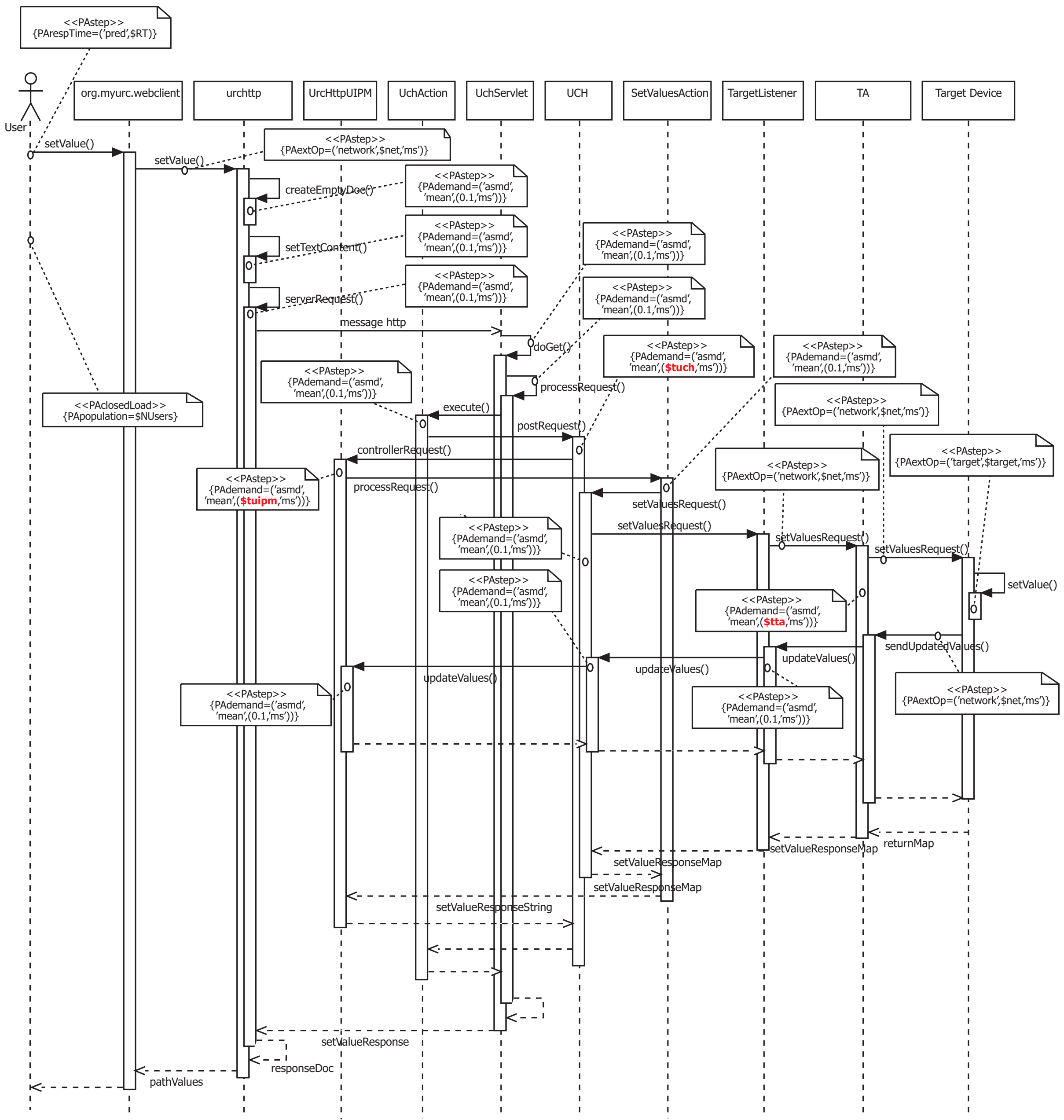
The physical structure of the system is necessary to describe the resources where to allocate the modules of the architecture, as well as their connections through a network, which obviously will delay the interchange of messages among modules according to the size of the messages. A UML deployment diagram, Figure 4, will help to understand these issues.

4.2 Performance Annotations

Once the detailed design has been carried out, the models of interest (Figs. 3 and 4) have to be annotated as PUMA proposes, i.e., with performance information according to the *UML Profile for Schedulability, Performance and Time Specification* (UML-SPT) [18]. They will help to introduce input parameters and metrics in the eventual performance model.

Table 1 summarizes this performance information concerning atomic actions duration collected by experimental tests, which have considered both UCHj and UCHe implementations. These actions are represented by `<<PAstep>>` stereotype, where `PAdemand` tag specifies its corresponding average execution time as an exponentially distributed random variable.

Other parameter that may affect system performance is the access to the *target* device, which is tagged by the `PAextOp` value. In the following, let us assume that this time is negligible, since it is independent of UCH architecture (e.g., the whole cycle time of a washing machine is very different from a TV set), and obviously we have to take it as an external and non-controlable part of our system.



<<PAstep>> {PArespTime=('pred',$RT)}
User
org.myurc.webclient
urchttp
UrcHttpUIPM
UchAction
UchServlet
UCH
SetValuesAction
TargetListener
TA
Target Device
setValue()
setValue()
<<PAstep>> {PAextOp=('network',$net,'ms')}
createEmptyDoc()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
setTextContent()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
serverRequest()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
message http
doGet()
processRequest()
<<PAstep>> {PAdemand=('asmd', 'mean',($tuch,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
<<PAstep>> {PAextOp=('network',$net,'ms')}
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
<<PAclosedLoad>> {PApopulation=$NUsers}
execute()
postRequest()
controllerRequest()
<<PAstep>> {PAextOp=('target',$target,'ms')}
<<PAstep>> {PAextOp=('network',$net,'ms')}
<<PAstep>> {PAdemand=('asmd', 'mean',($tuipm,'ms'))}
processRequest()
setValuesRequest()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
setValuesRequest()
setValuesRequest()
setValuesRequest()
setValue()
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',($tta,'ms'))}
sendUpdatedValues()
updateValues()
updateValues()
updateValues()
<<PAstep>> {PAextOp=('network',$net,'ms')}
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
<<PAstep>> {PAdemand=('asmd', 'mean',(0.1,'ms'))}
setValueResponseMap
returnMap
setValueResponseMap
setValueResponseMap
setValueResponseString
setValueResponse
responseDoc
pathValues

Fig. 3. Sequence diagram describing a key performance scenario: `SetValue()` request.

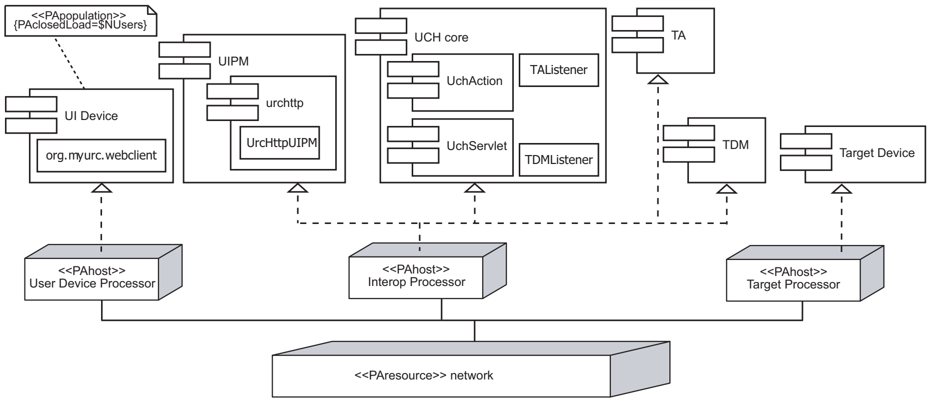


Fig. 4. Deployment diagram of UCH architecture.

Parameter	UCHj	UCHe
`$tuipm`	243.05	81.96
`$tuch`	3.00	1.18
`$tta`	51.45	1.27

Table 1. Mean execution time in milliseconds.

The metric to be calculated will be the system response time, it has been annotated in the sequence diagram attached to the first message. The workload of the system is the number of users, annotated in the first life line of the sequence diagram. The rest of parameters of interest can also be seen in this diagram.

From the UML-SPT models we obtained the corresponding CSM model that will not be depicted here for lack of space. Actually this model, being an intermediate one, offers none further interesting details to understand the system.

4.3 GSPN Model

The next step in PUMA advises to transform the CSM into a performance model (GSPN in our case), for which we used the CSM2GSPN translator [3] then to obtain the GSPN in Figure 5. The value of `$tuimp` in Table 1 corresponds with the duration of transition `controllerRequest` in the GSPN, as indicated by annotation attached to the same message (`controllerRequest`) in Figure 3. In the same way, values for GSPN transitions `postRequest` and `setValuesRequest` correspond to variables `$tuch` and `$tta`. On the other hand, `setValue, sendUpdateValues` and `setValueResponse` are external operations whose duration is given by variables `$net` and `$target` that were set to 0.01 milliseconds. These values were taken from real experimentation, they are low because they depend on the network infrastructure that in this case was the corporate intranet. Finally, transitions `createEmptyDoc, setTextContent, serverRequest, doGet, processRequest, execute` and `updateValues` represent simple operations or calls, being their execution time around 0.01 milliseconds. The accuracy of the latter values is imposed by the system clock function. Resources are indicated with tokens in corresponding places. So, the number of concurrent users, or system closed workload, is the number of tokens in place `users`, then matching to variable `$NUsers` in the sequence and deployment diagrams. Tokens in place *p_userDevice* represent the user device (and its corresponding user interface) and hence the concurrent threads, while places *p_uipm*, *p_uch* and *p_ta* represent the UCH modules as resources. A first glance to the GSPN reveals that the net sequentially executes the activities once resources are acquired step by step, hence the performance will be hampered by the number of concurrent users, place `users`, and alleviated by the number of available threads, *p_userDevice*, *p_uipm*, *p_uch* and *p_ta*.

5 Performance Analysis

Once the performance model has been built, we used TimeNET [20] in order to solve the GSPN by means of simulation techniques. Our first analysis goal was to study UCH scalability considering the current open source implementations, UCHj and UCHe. Later, we will try to determine a system "optimal configuration" in a context with several concurrent users.

5.1 Scalability

UCH was initially designed as an interoperable architecture for smart homes, which means that relatively few people will be simultaneously using the system to control different devices. Nevertheless, this architecture may be projected in more complex environments, such as intelligent buildings, business buildings, hospitals or hotels. In this case, the system will have to support requests from several concurrent users.

Firstly, both implementations, UCHj and UCHe, were experimentally tested within the INREDIS project [2, 12]. These experiments assumed that each user wanted to control his/her own device, i.e., one user per *target* device. Results regarding response time [2, 12] could be hardly obtained up to forty users due to the difficulties of real experimentation. We reproduced these experiments using our performance model, which meant to put as many tokens as users in places *users* and *p_userDevice* of the GSPN, so to also match one user to one interface, and then we obtained the results in Figure 6. Differences in the results between our performance model and the Java and C/C++ real implementations accounted for less than a ten percent, then we assumed our GSPN as a valid performance model and ready to address experiments initially not feasible to carry out with the real implementations.

On the other hand, the discussion about what could be considered a good response time is controversial, since besides the times so far considered, it may

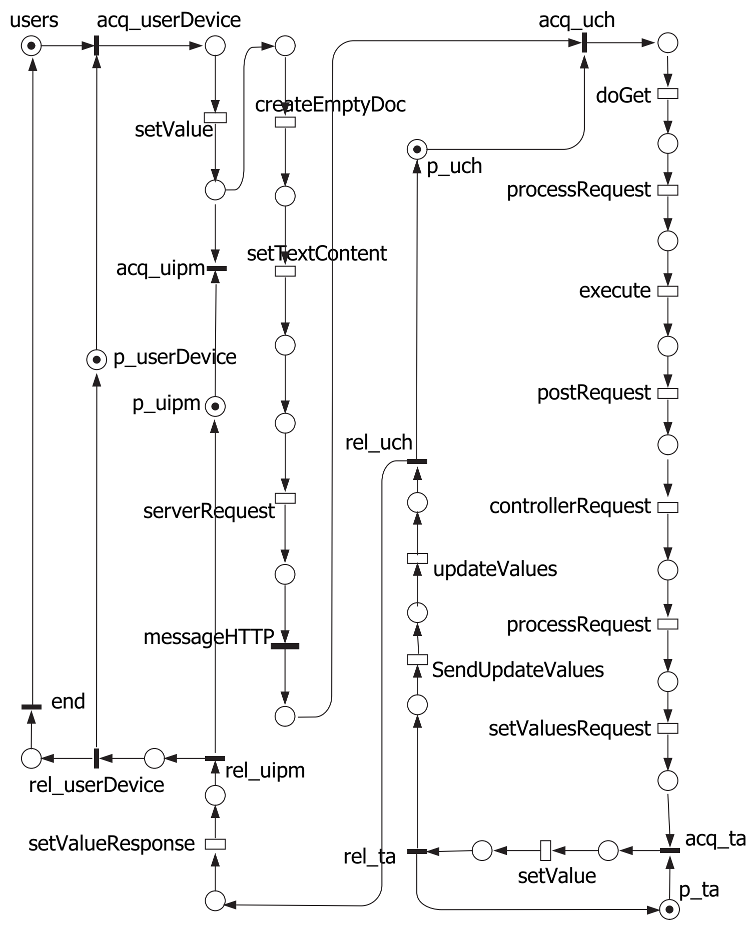


Fig. 5. Petri net describing the key performance scenario.

also depend on the kind of impairment the user has and on the kind of *target* device the user wants to control. For example, elderly people could request commands in their personal telecare device at a rate of few seconds. However for a blind person it could last much more time to operate for instance the washing machine. Pragmatically, we will assume quantities around ten seconds as acceptable response times, according to [15, 17]. This is so because in the experiments (both, real and GSPN) we did not want to consider the time spend by the impaired persons and neither the time to operate the target[4]. Therefore, for concrete scenarios (persons and targets with defined profiles) the response times will be higher.

Our next step, assuming valid the performance model, was to exercise the same for a larger amount of concurrent users. Figure 7 extends experiments in

[4] Note that this is not a limitation to evaluate the UCH architecture.

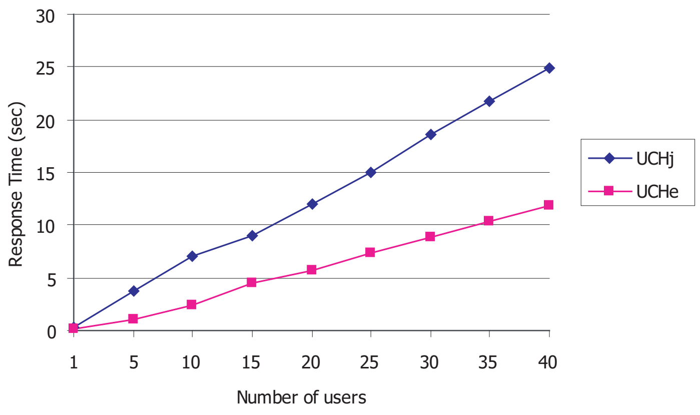


Fig. 6. Response time of UCHj and UCHe implementations from 1 to 40 concurrent users.

Figure 6 up to 1000 users, so offering response time of the GSPN w.r.t. both implementations, where we observe that UCH performs poorly, specially Java implementation. Therefore, although UCHe outperforms UCHj, UCH should not be considered as a practicable architecture in a real time environment with hundreds of concurrent users. Now, we will try to get solutions by means of replication.

5.2 Replication

UCH specification does not define whether UIPMs, TDMs and TAs modules should be executed as independent processes or threads, or if they should be allocated in different memory spaces, hence these are choices for each specific implementation. In the case of both UCHj and UCHe, all UCH components execute in the same space of memory and are attended by a unique process.

Now, we want to study an "optimal configuration" for the architecture by means of modules replication. In fact, we replicated the two implementations of UCH modules, i.e. UIPM, Socket Layer and TA modules (represented by places p_uipm, p_uch, p_ta in Figure 5), which were populated with threads ranging from 1 to 25 in the same space of memory. Figure 8(a) shows the effect of adding threads in UCHj implementation and Figure 8(b) in UCHe. Although both graphics have similar shape, the order of magnitude is quite different. As expected, UCHe outperforms UCHj. Note that using 15 threads, the response times improve significantly in both cases, but adding more threads they do not perform better. For a few hundreds of users, UCHe may get acceptable values with 15 threads, around 8 seconds, however UCHj in these cases still is not feasible, around 50 seconds. Figure 9 summarizes the response times of both implementations with 15 threads.

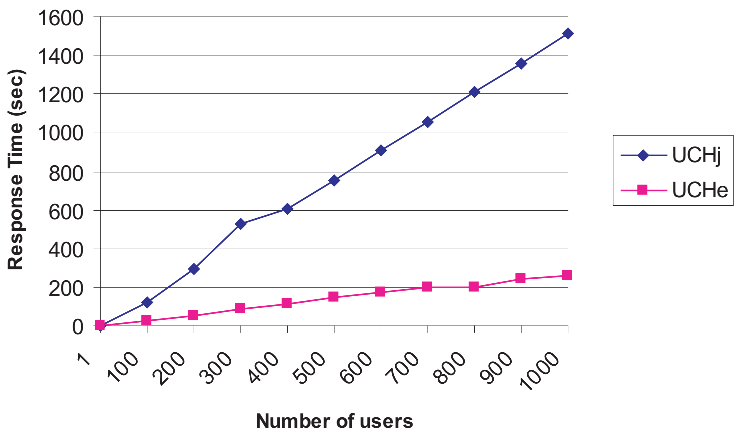


Fig. 7. Response time of UCHj and UCHe implementations.

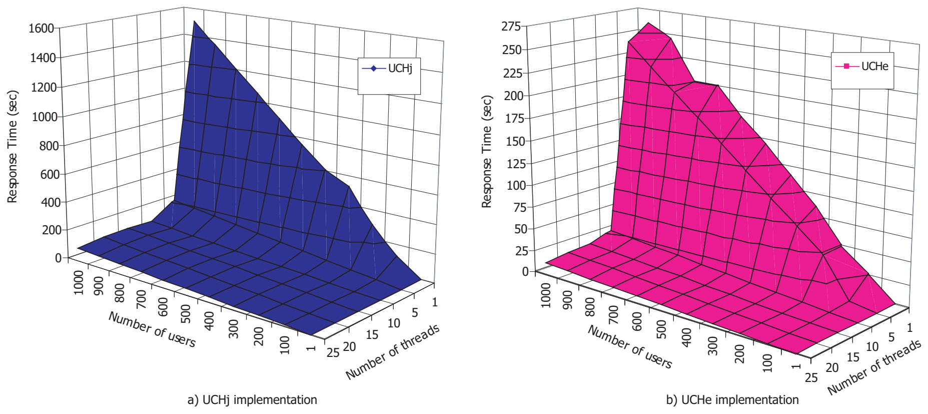


Fig. 8. Response time of UCH implementations adding threads.

As a conclusion, a solution for an "optimal configuration" for populated environments could be a UCHe implementation of 15 threads, since as it can be observed in the graph, UCHe response times in this setting may be acceptable.

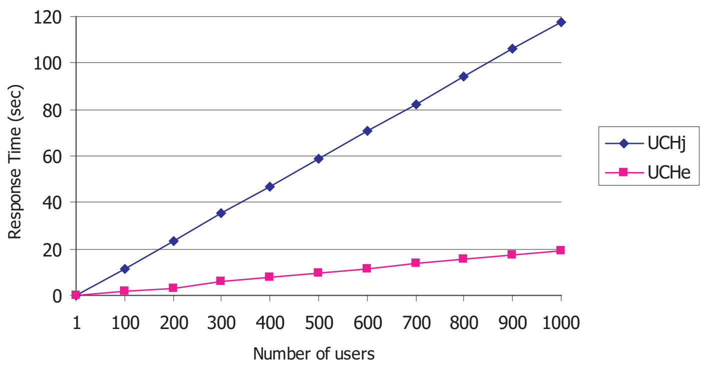


Fig. 9. Response times of UCHj and UCHe implementations using 15 threads.

6 Conclusions and Further Work

This paper has analysed the performance of the UCH interoperable architecture through two open source implementations, UCHe and UCHj. The paper has demonstrated that PUMA is useful for the assessment of an industrial case study. The use of GSPNs has made possible to validate experimental results and to analyse scenarios that otherwise could not be afforded with real experimentation.

The performance results demonstrate that current UCH implementations fit in a very delimited context, with very few users. However we assessed that system performance can be improved by adding threads, but also that UCHe will always outperform UCHj, confirming that it is the best option for achieving user requirements.

Regarding complex software projects of "similar" characteristics to UCH that had been assessed using formal methods, we have found none in literature to be compared. However, we can say that previous experiences with PUMA have been reported in [25, 8, 7, 14], but these works are examples or studies for academic purposes.

We think that further analyses of the GSPN can help improving URC architecture and consequently related implementations. The solution explored in this paper, i.e. module replication, have to be supplemented with other architectural decisions that indeed we hope they could be assessed by the GSPN analysis. The final objective of these assessments is to gain insight in closing the "assessment loop" (Design $\rightarrow$ Performance Model $\rightarrow$ Analysis $\rightarrow$ Results $\rightarrow$ new

Design). Actually, the first transitions in the loop are well-known today and even some tool support exists for them. However, very different is the case for the last one (from Results to a new Design), and our interest is to further exploit this project to gain insight at this regard and then to try to automate some aspects of this transition, i.e. how to automate design decisions based on analysis results.

Acknowledgments The research described in this paper arises from a Spanish research project called INREDIS (INterfaces for RElations between Environment and people with DISabilities)[11]. INREDIS is led by Technosite and funded by CDTI (Industrial Technology Development Centre), under the CENIT (National Strategic Technical Research Consortia) Programme, in the framework of the Spanish government's INGENIO 2010 initiative. The opinions expressed in this paper are those of the authors and are not necessarily those of the INREDIS project's partners or of the CDTI.
José Merseguer has been supported by the European Community's 7th FP project DISC (Grant Agreement n.INFSO-ICT-224498)
We would like to thank José Antonio Gutiérrez for his work in the experimental tests.

References

1. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* John Wiley Series in Parallel Computing - Chichester, 1995.
2. E. Catalán and M. Catalán. Performance Evaluation of the INREDIS framework. Technical report, Departament Enginyeria Telematica, Universitat Politècnica de Catalunya, 2010.
3. The CSM to GSPN translator. http://webdiis.unizar.es/~jmerse/csm2pn.html.
4. D. Davis and M. Parashar. Latency Performance of SOAP Implementations. In *CCGRID*, pages 407–412. IEEE Computer Society, 2002.
5. R. Elfwing, U. Paulsson, and L. Lundberg. Performance of SOAP in Web Service Environment Compared to CORBA. In *APSEC*, pages 84–. IEEE Computer Society, 2002.
6. Eurostat. Statistical Office of European Union. http://epp.eurostat.ec.europa.eu/.
7. E. Gómez-Martínez, S. Ilarri, and J. Merseguer. Performance Analysis of Mobile Agents Tracking. In *Sixth International Workshop on Software and Performance (WOSP 2007)*, pages 181–188. ACM, February 2007.
8. E. Gómez-Martínez and J. Merseguer. Impact of SOAP Implementations in the Performance of a Web Service-Based Application. In Geyong Min, Beniamino Di Martino, Laurence Tianruo Yang, Minyi Guo, and Gudula Rünger, editors, *ISPA Workshops*, volume 4331 of *Lecture Notes in Computer Science*, pages 884–896. Springer, 2006.
9. M.R. Head, M. Govindaraju, A. Slominski, P. Liu, N. Abu-Ghazaleh, R. van Engelen, K. Chiu, and M.J. Lewis. A Benchmark Suite for SOAP-based Communication in Grid Web Services. In *SC*, page 19. IEEE Computer Society, 2005.
10. IEEE. *IEEE 1901 Draft Standard 3.0 for Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications.* 2010.

11. INREDIS. *INterfaces for RElations between Environment and people with DISabilities.* http://www.inredis.es/.
12. INREDIS. *Deliverable-78.2.1. Final Guide to a Generic Platform Deployment*, 2010.
13. ISO. *ISO 24752:2008 Information technology – User interfaces – Universal remote console – Part 1: Framework.* ISO, Geneva, Switzerland, 2008.
14. C.K.M. Marques, S. Ilarri, J. Merseguer, and G.C. Barroso. Performance analysis of a dynamic architecture for reconfiguration of web servers clusters. In *Proceedings of the 6th International Conference on Networking and Services (ICNS'10)*, pages 224–229.
15. R. B. Miller. Response time in man-computer conversational transactions. In *AFIPS '68 (Fall, part I): Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277, New York, NY, USA, 1968. ACM.
16. A. F. Newell. Accessible computing – past trends and future suggestions: Commentary on "computers and people with disabilities". *ACM Transactions on Accessible Computing (TACCESS)*, 1(2), 2008.
17. J. Nielsen. *Usability Engineering.* Morgan Kaufmann, 1993.
18. Object Management Group, `http://www.uml.org`. *UML Profile for Schedulabibity, Performance and Time Specification.*, January 2005. Version 1.1.
19. D. Petriu and M. Woodside. An intermediate metamodel with scenarios and resources for generating performance models from UML designs. *Software and Systems Modeling*, 6(2):163–184, 2007.
20. The TimeNET tool, http://pdv.cs.tu-berlin.de/~timenet/.
21. URC Consortium. http://myurc.org.
22. URC Consortium. *iPhone client for UCH (iUCH).* http://myurc.org/tools/iPhone/.
23. URC Consortium. *Universal Control Hub for C++ (UCHe).* http://myurc.org/tools/UCHe/,.
24. URC Consortium. *Universal Control Hub for Java (UCHj).* http://myurc.org/tools/UCHj/.
25. C.M. Woodside, D.C. Petriu, D.B. Petriu, H. Shen, T. Israr, and J. Merseguer. Performance by unified model analysis (PUMA). In *WOSP*, pages 1–12, 2005.
26. G. Zimmermann and G. C. Vanderheiden. The Universal Control Hub: An Open Platform for Remote User Interfaces in the Digital Home. In Julie A. Jacko, editor, *HCI (2)*, volume 4551 of *Lecture Notes in Computer Science*, pages 1040–1049. Springer, 2007.