Empir Software Eng manuscript No.
(will be inserted by the editor)

Performance Assessment of an Architecture with Adaptative Interfaces for People with Special Needs

Industrial experience report

Elena Gómez-Martínez · Rafael González-Cabero · José Merseguer

Received: date / Accepted: date

Abstract People in industrial societies carry more and more portable electronic devices (e.g., smartphone or console) with some kind of wireless connectivity support. Interaction with auto-discovered target devices present in the environment (e.g., the air conditioning of a hotel) is not so easy since devices may provide inaccessible user interfaces (e.g., in a foreign language that the user cannot understand). Scalability for multiple concurrent users and response times are still problems in this domain. In this paper, we assess an interoperable architecture, which enables interaction between people with some kind of special need and their environment. The assessment, based on performance patterns and antipatterns, tries to detect performance issues and also tries to enhance the architecture design for improving system performance. As a result of the assessment, the initial design changed substantially. We refactorized the design according to the Fast Path pattern and The Ramp antipattern. Moreover, resources were correctly allocated. Finally, the required response time was fulfilled in all system scenarios. For a specific scenario, response time was reduced from 60 seconds to less than 6 seconds.

Keywords Software architecture · Performance assessment · ICT for people with special needs · Industrial report · Performance patterns and antipatterns

Elena Gómez-Martínez*
Babel Group, Universidad Politécnica de Madrid (Spain)
E-mail: egomez@babel.ls.fi.upm.es

Rafael González-Cabero
Ontology Engineering Group, Universidad Politécnica de Madrid (Spain)
E-mail: rgonza@fi.upm.es

José Merseguer
Dep. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza (Spain)
E-mail: jmerse@unizar.es

1 Introduction

Universal Access continues being a critical quality target for Information and Communication Technologies (ICTs), as Stephanidis (2001) stated, especially in industrial societies where there is a growing number of people with functional diversity, including those with aging-related conditions. Indeed, ICTs may require particular skills and abilities to interact with platforms, the plethora of wireless communication systems and smart devices such as kiosks or ATMs. The inexistence of these skills and abilities extends in some cases the traditional concept of disabled people towards people with functional diversity or special needs. The growing gap between their abilities and access to ICT is called the *digital divide*.

The INREDIS project[1] (INterfaces for RElations between Environment and people with DISabilities) aimed to develop environments that enable the creation of communications and interaction channels between people with some kind of special need and their context, where the targets are a set of auto-discoverable devices. More than 200 researchers from 14 Spanish companies and 19 research organizations collaborated to carry out this project during 48 months and a budget of €23.6 millions.

Although goal of the INREDIS project was to completely develop an accessibility architecture for disabled people, here we only focus on the analysis and design steps of the project, in particular in the performance assessment carried out. The rationale for the assessment is to explore the feasibility of deploying this architecture in environments with a large number of concurrent users. Early performance assessment for the system architecture is highly desirable to prevent underperformance during system deployment.

The assessment is carried out using principles and techniques of the Software Performance Engineering (SPE, Smith 1990; Cortellessa et al 2011). SPE represents the entire collection of software engineering activities and related analyses used throughout the software development cycle, which are directed to meeting performance requirements (Woodside et al 2007). The paper applies SPE at the software architecture level. To the best of our knowledge, a complete report of an industrial experience in the SPE field based on performance patterns and antipatterns has not yet been reported.

This paper is an industrial experience report since we report results regarding the application of a performance assessment methodology to a real industrial project, the INREDIS project. We have followed recommendations from Runeson and Höst (2009) about case study research methodology for software engineering. Thus, the objective of the paper is twofold. First, we want to describe how we applied SPE in the project, with special attention to performance patterns and antipatterns. So, this paper can be a blueprint for practitioners needing to evaluate performance in a software project. We call this *external objective* of the paper. On the other hand, we want to assess the INREDIS architecture for performance, which is of interest not only for the

[1] http://www.inredis.es/default.aspx

project engineers but also for designers of accessible user interfaces. We call this *internal objective*.

The rest of the paper contains the following sections. Section 2 describes the INREDIS Interoperable architecture. Section 3 outlines the assessment approach. Section 4 accomplishes the external objective. Section 5 addresses the internal objective. Section 6 discusses both objectives. Section 7 covers related work and Section 8 concludes the paper. The paper also includes four appendices. They detail some aspects of interest for the project, but the paper can be understood without a deep reading of these appendices.

2 An Interoperable Architecture

The INREDIS architecture further develops the idea of Universal Control Hub (UCH) proposed by Zimmermann and Vanderheiden (2007). Its rationale is that a person with its adapted device (e.g., smartphone, PDA or universal controller) should be able to interact and control different devices (television, door locks, ATMs, and a long etcetera), as well as external software services. For instance, a blind person can control the washing machine (target device), by means of his/her mobile phone (controller device). The controller device allows to introduce assistive technologies to bridge the gap between the user and the target device.

The INREDIS architecture was conceived as a universal solution capable to provide disabled and elderly people with accessible and personalized interfaces according to their preferences and needs. Consequently, the architecture was designed for a general purpose context of use. Nevertheless, some running prototypes were built for different environments, covering a wide range of real world scenarios, among them leisure services (location and purchase tickets for events), smart home (Sainz et al 2011), urban networking (Giménez et al 2012), social networks (Murua et al 2011), eGoverment (Alvargonzález et al 2010) and banking services (ATMs) (Pous et al 2012). While users with functional diversity are able to fully exploit the architecture capabilities, "any" user should be able to obtain benefits when using the system (e.g., using their mobiles as universal remote controllers in the smart environment).

The most important components of the INREDIS architecture are depicted in the UML deployment diagram[2] in Figure 1:

- **Knowledge Base** (KB in Figure 1). It stores ontologies and instances sets that provide formal descriptions of the elements in the INREDIS domain (e.g. user, assistive software instances, devices, software, etc.). The KB also stores the terminology and a collection of rules. It also provides mechanisms for reasoning with each of these type of knowledge and allows querying all the instances set using SPARQL (Prud'hommeaux and Seaborne 2006).

[2] The reader should note that we have added some grey notes in the UML diagrams. They are performance annotations that will be explained in Section 4.

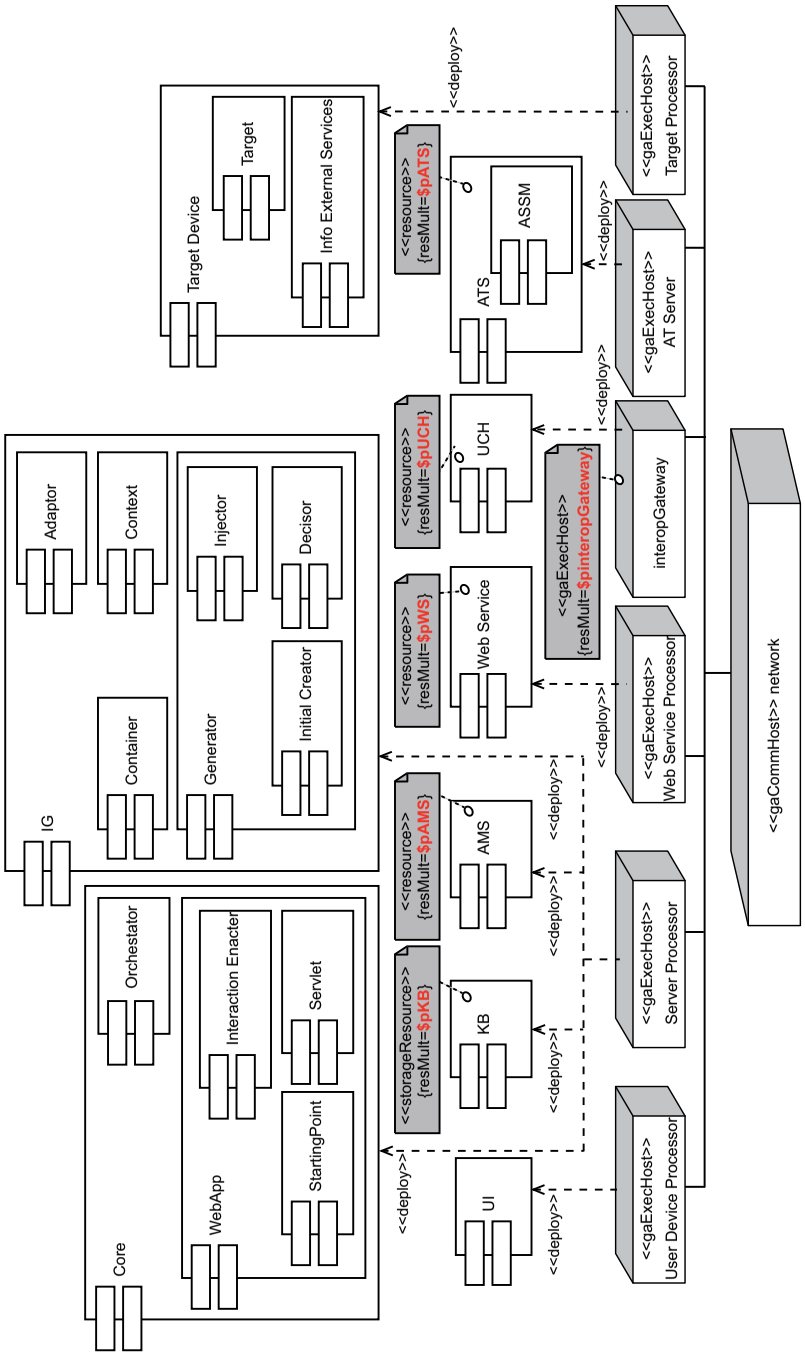

Fig. 1 UML deployment diagram of the INREDIS Interoperable Architecture

– **Adaptive Modelling Server** (AMS in Figure 1). It keeps updated the KB content using information from different and heterogeneous sources (application context, user interaction logs or complex events processing).
– **Assistive Technology Server** (ATS server in Figure 1). It provides automatic discovery and configuration of assistive technologies, in a smart and transparent fashion reducing the existing accessibility gap that may exist between the users and their universal controller device.
– **Interface Generator** (IG in Figure 1). It adapts interfaces expressed in a generic and abstract language, a subset of the User Interface Markup Language (Phanouriou 2000), into concrete utilizable and accessible ones (implemented in XHTML (2010)). This activity is made in terms of the user characteristics, the device capabilities and the context. All this is possible using the reasoning capabilities provided by the Knowledge Base.

The main processes performed by the INREDIS architecture are pictured by the UML Interaction Overview Diagram (IOD) in Figure 2.

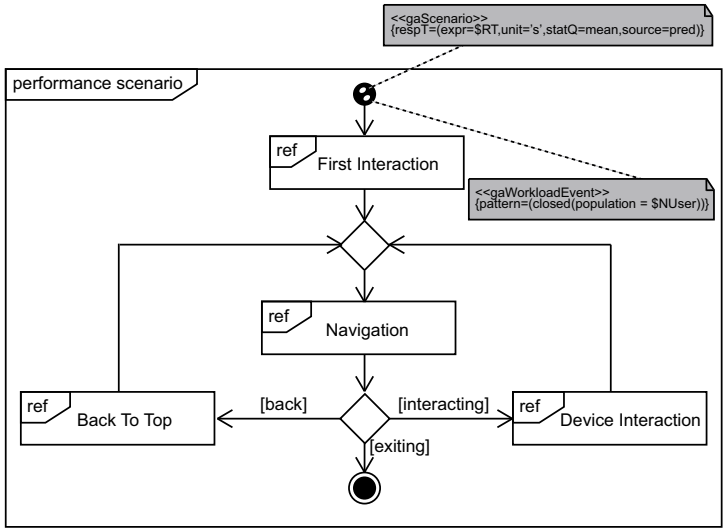

Fig. 2 UML Interaction Overview Diagram of the main processes of the architecture

– **First Interaction.** It consists in the creation of the initial interface that acts as the access medium to the environment for the user. In the generation of such interface the system must take into consideration the relevant set of devices and services for the user (the INREDIS perimeter) and their state (without forgetting the special needs of the user). This process involves an interface generation subprocess, for building an accessible XHTML interface, and the determination of the set of assistive software instances that permit the user to interact which such interface.

- **Navigation.** Once the user has selected the device or service to interact with, the navigation process starts. Devices and services are defined by complex multi-staged interface descriptions that users can navigate. Through navigation, we simplify the information offered at a time to the user and we allow complex conversations with the device.
- **Device interaction.** When user navigation ends, or when the user performs certain actions in the device interface, interactions with the end device occur. The architecture supports interactions with devices either as a UCH Target or as a Web service transparently.
- **Back to top.** The user can at any moment reset its interaction with the device, going back to the first interface that the device offers. An updated initial interface of the device must be rendered again.

These processes can be summarized with the following example: A user wants to turn the TV at home on. Firstly, the user logins with his/her nickname and password using his/her mobile phone (*device controller*). A screenshot with the available devices and services, grouped by environment, is displayed (**First Interaction**), e.g. it appears "Smart Home", "Products and Services" and "Health Care", among others. These devices and services depend on the user's location. The user navigates through the screenshots until s/he identifies the device or service that s/he wants to control (**Navigation**); for instance, in the "Smart Home" display, s/he selects "TV set" (*target device*) and "Turn on/Turn off" options. S/he turns on the TV (**Device Interaction**) and waits for the notification of the new status. Finally, s/he comes back to the first screenshot in order to interact with other device or service (**Back to top**). Obviously, all the screenshots must be accessible and adapted to the specific needs and preferences of this user.

Besides of these processes, special attention deserves the **Assistive Software Selection Mechanism** (ASSM) (Gómez-Martínez et al 2013). The ASSM makes the environment able to automatically select the most suitable assistive technologies provided by the ATS. It considers possible discrepancies between the user and the environment, namely in the case of functional diversity.

Each of these four processes and the ASSM are carefully explained in Appendix A. The appendix also includes UML sequence diagrams, which describe the behaviour of the system and make up the design of the INREDIS architecture. The rest of the paper can be understood without checking the full design, however, we have considered of interest to include the design, since:

- Being the paper *industrial*, the reader can neatly realize the magnitude of the project through its design.
- Being *empirical* the scope of the journal, the design of the system illustrates this aspect and it can be very useful for practitioners since it provides guidance for future projects.
- The design of the architecture is the cornerstone for performance evaluation, the performance engineer needs to use these diagrams in his/her work, as we later describe.

Finally, although the paper is exclusively focussed on the analysis and design stages of the INREDIS architecture, it is worth mentioning that the actual architecture implementation and the development of some target devices and services was carried out cooperatively by all the INREDIS partners[3].

3 Overview of the Methodology

For achieving the *internal objective* of the paper, i.e., the performance assessment of the architecture, we have followed a methodology and the principles of Software Performance Engineering (SPE) (Smith 1990).

The performance assessment was intricate, due to several reasons:

- INREDIS is a very large system, various developing teams of tens of people were involved.
- Technologies were new for these teams. So, it was unknown how to capitalize these technologies for system performance maximization.
- Being the product targeted to people with special needs, performance requirements may differ from the habitual ones.
- We expected to deploy the system in various settings, most of them not yet completely defined. For example, hotels or facilities where hundreds of users could leverage the system.

Performance evaluation of software systems has been traditionally accomplished after deployment. This is the well-known *fix-it later approach* and it has well-known problems (Smith 1990). For example, the cost of re-architecting, re-implementing and re-deploying the system when performance goals are not fulfilled. Also the over-budget for being out of schedule as Woodside et al (2007) described. Moreover, our project had specific reasons for rejecting the fix it later approach:

- Although the operative versions of the system should be deployed at the very end of the project, we needed to deploy prototypes at the beginning of the project, for users experimentation.
- We needed to experiment with the potential environments previously referred. So, to gain some insight about their potential system performance. Otherwise, successful implementations in real deployments could not be reused in potential environments, which could imply to start a new project for each new deployment.

Software Performance Engineering (SPE) was defined by Smith (1990) as a research field that tries to overcome the problems previously described. The proposal is to leverage, for evaluation purposes, the software models created by designers. So, performance evaluation can be carried out early in the lifecycle when implementation has not been accomplished yet. Prototypes can help to validate evaluation results, evaluation is then seen as a *by-product* of the software design process.

[3] `http://www.inredis.es/consorcio.aspx`

In this project, we resorted to well-established SPE principles and techniques for performance assessment of our architecture. The methodology we followed, depicted in Figure 3, loops to decide whether the performance objectives are met and to obtain the architecture that can meet these objectives. The methodology is a simplified version of the PUMA methodology (Performance by Unified Model Analysis) proposed by Woodside et al (2005, 2013). We based our work on PUMA for several reasons, among others because it was developed by one of the most experienced performance evaluation groups world-wide[4], and because we have had satisfactory industrial experiences using it in the past (Gómez-Martínez et al 2007; Gómez-Martínez and Merseguer 2010). We simplified PUMA because in some respects it is difficult to use. In particular, we could avoid the use of multiple formalisms because in our industrial project only one formalism, in this case Petri nets, would be used for analysis (PUMA was designed for managing various formalisms in the same project). Although we chose PUMA for the reasons above, there exist other methodologies that could have been used provided that the authors would have had previous industrial experiences with them. Among these proposals it is worth mentioning Q-ImPrESS (2009), PASA by Williams and Smith (2002) or the Palladio Component Model by Becker et al (2009); these methodologies are reviewed in Section 7, moreover we highlight some works where they were applied.

Our simplified methodology proposal is then composed of the following phases:

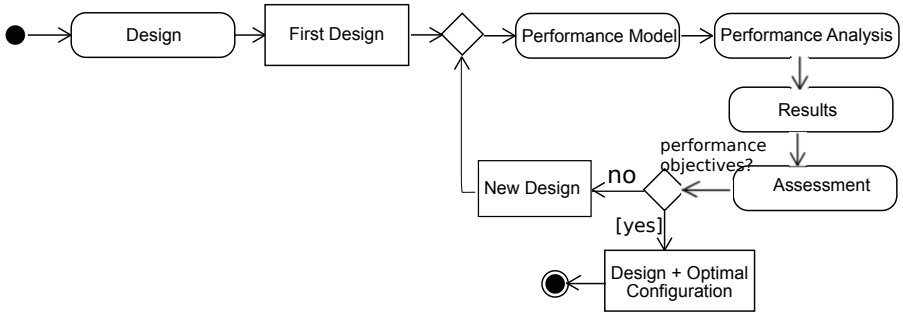

Fig. 3 Performance assessment methodology

– *Design*. The methodology begins by modeling the system architecture using UML diagrams. We also address the behaviour of those scenarios of the system critical for performance. Section 2 and Appendix A reported these two first steps of this stage for our project. Finally, in the design it is also introduced the *performance view* of the system, next section reports this step for our project.

[4] `http://www.sce.carleton.ca/rads/index.html`

- *Performance Model*. As proposed by PUMA, for each critical scenario, a performance model is obtained. We used Generalized Stochastic Petri nets (Ajmone Marsan et al 1995) as performance model.
- *Performance Analysis*. Measures of interest in our project are response time, scalability and resources utilization. They are computed by analysis or simulation of the performance model. We will carry out sensitive analysis, which means to modify performance parameters to test different system configurations. Sensitive analysis is managed in the assessment step to locate performance issues (e.g., high response times or overused resources) and solutions. In fact, the assessment of these outcomes will help us to get a responsive and scalable system.
- *Assessment*. The assessment stage proposes alternatives to meet performance objectives and to improve the architecture. Resource replication, threading and improvement of service times are the choices commonly explored. In our project, we also considered performance patterns and performance antipatterns as choices that could influence performance.

We have applied the methodology at software architecture design level. In fact, architecture design is a crucial part of the software design process, where decisions about which software elements will make up the system and their relationships are taken. Software architectures have emerged in the last years as the cornerstone for early evaluation of qualitative and quantitative properties of the software (QoSA 2005-2013). In the SPE field, architecture design is recognized as an asset for performance assessment.

4 Methodology: Approach for Performance Assessment

The following subsections accomplish our *external objective*, hence, we aim to apprise practitioners of the use of the methodology. We offer advise by indicating how we actually applied the methodology in the INREDIS project and which were our choices (e.g., which languages, performance models or tools we used).

4.1 Design

Section 2 and Appendix A described the UML design of the system, which was elaborated by the software engineers of the INREDIS project. In particular, the architectural description has a focus on the behavioral view, which is of primary importance for performance assessment. The overall architecture was presented in the deployment in Figure 1. The IOD in Figure 2 defines a general system scenario made of four sub-scenarios, each one describing a part of the system behaviour.

Following SPE principles, we now introduce the *performance view* of the system. The usual way in SPE for introducing a performance specification is by annotating the design diagrams. Annotations account for properties such

as workload, host demands or routing rates. Profiling is the mechanism UML offers to enhance a design with specifications beyond the typical structural and behavioral views. The UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) described by Object Management Group (OMG) (2011) is a standard that customizes UML for the modeling and analysis of performance properties. MARTE builds on previous UML profiles and it is the most comprehensive proposal for performance assessment using UML. Key features of MARTE are the non-functional properties (NFP) framework and the Value Specification Language (VSL). The former[5] is used to define data-types characterized by several properties, such as the *source* - that allows to specify whether an NFP is a requirement or a measure to be predicted, the type of *statistical measure* associated to an NFP (e.g., mean), the type of the *quality order* relation in the value domain of a NFP, for comparative analysis purposes. Instead, the VSL enables the specification of variables and complex expressions according to a well-defined syntax.

In this step, the software engineer defines performance objectives of interest for analysis of the architecture. MARTE annotations help to this end. We present INREDIS performance objectives in detail in Section 4.3. MARTE annotations appear as gray notes in the UML diagrams we have presented. In the following, some of the most interesting annotations are commented. They capture properties, measures and requirements of interest for carrying out performance analysis.

- The **workload** has been specified in the IOD in Figure 2 using *GaWorkloadEvent* annotation. This is a closed workload, then specifying the number of concurrent users in the system through variable $\$NUsers$ in VSL. This variable will allow to parameterize the performance model with values to carry out system sensitivity analysis.
- The **response time** has also been specified in the IOD in Figure 2, in this case using *GaScenario* annotation. Response time is a measure to be predicted during analysis as indicated by *source=pred*. The result will be gathered in variable $\$RT$. The unit of measurement are seconds and the *statistical measure* is a mean.
- **Host demands** and the **size of the messages** are requirements needed to compute system duration activities. They are provided by the engineer during analysis. Figure 19 offers some examples. The rest of the sequence diagrams complete these specifications.
- Sequence diagrams also capture system **routing rates**, in the alternative fragments. See for example in Figure 20 the *prob* annotation in the *GaStep* attached to the alternative fragments.
- System **resources** are expressed in MARTE as lifelines in the sequence diagrams. Annotations *acqRes* and *relRes* attached to *GaCommStep* specify their **acquisition** and **release**. Figure 20 depicts several examples, see one of them attached to the KB lifeline. For specifying the number of system available **resources**, annotation *resMult* in the deployment (Figure 1) is

[5] See the top grey note in Figure 19 for an illustrative example.

used. Variables (for example $\$pKB$ or $\$pAMS$) will allow to perform sensitive analysis parameterizing the system with different number of resources.

The performance view of the INREDIS project was completely developed using MARTE. The diagrams in Section 2 and Appendix A show this view, which enabled the INREDIS design for performance evaluation. In the following we summarize the relevant scenarios in the performance view:

- First interaction scenario in Figure 19, for creating an initial interface.
- Navigation scenario in Figure 22, for the user to navigate the interface.
- Device interaction scenario in Figure 24, for describing user interactions with devices and services.
- Back to top scenario in Figure 25, for going back to the device top interface.

4.2 Performance Model

Following the methodology, we need to obtain a performance model for each critical scenario that we have previously annotated.

Performance models are formal models that help to obtain measures of interest (e.g., system response time) by analysis or simulation. There are different kinds of performance formalisms widely accepted in SPE: queuing networks (Lazowska et al 1984), stochastic process algebras (Hermanns et al 2002) and stochastic Petri nets (Ajmone Marsan et al 1995). We used stochastic Petri nets (SPN) and concretely generalized SPN (GSPN). Appendix B offers a brief introduction to GSPNs. Some of the reasons for using GSPNs were: their capacity to represent routing rates, competition for shared resources, stochastic duration of the host demands, parallel executions and forks and joins. All these parameters were present in the INREDIS project. Moreover, we had experience applying GSPN, which is an asset in an industrial project, mainly to minimize risks and to meet deadlines and schedules.

Fortunately, there exist SPE methodologies that translate performance-annotated UML models into the formalisms above mentioned. For example, the work in (Petriu and Woodside 2002) to obtain queuing networks, the work in (Tribastone and Gilmore 2008) to obtain process algebras or (Bernardi and Merseguer 2007; Distefano et al 2011) to obtain Petri nets. Some of these methodologies have associated tools that automate the translation process. Concretely, we used ArgoSPE (Gómez-Martínez and Merseguer 2006) which translates UML into GSPNs. We translated each critical scenario and obtained the structure of a GSPN, Figure 4 depicts the GSPN for the First Interaction scenario. The rest of GSPN models of the INREDIS project, obtained from the design, appear in Appendix D. The translation, although automatic, required some additional effort, Section 6 discusses these issues.

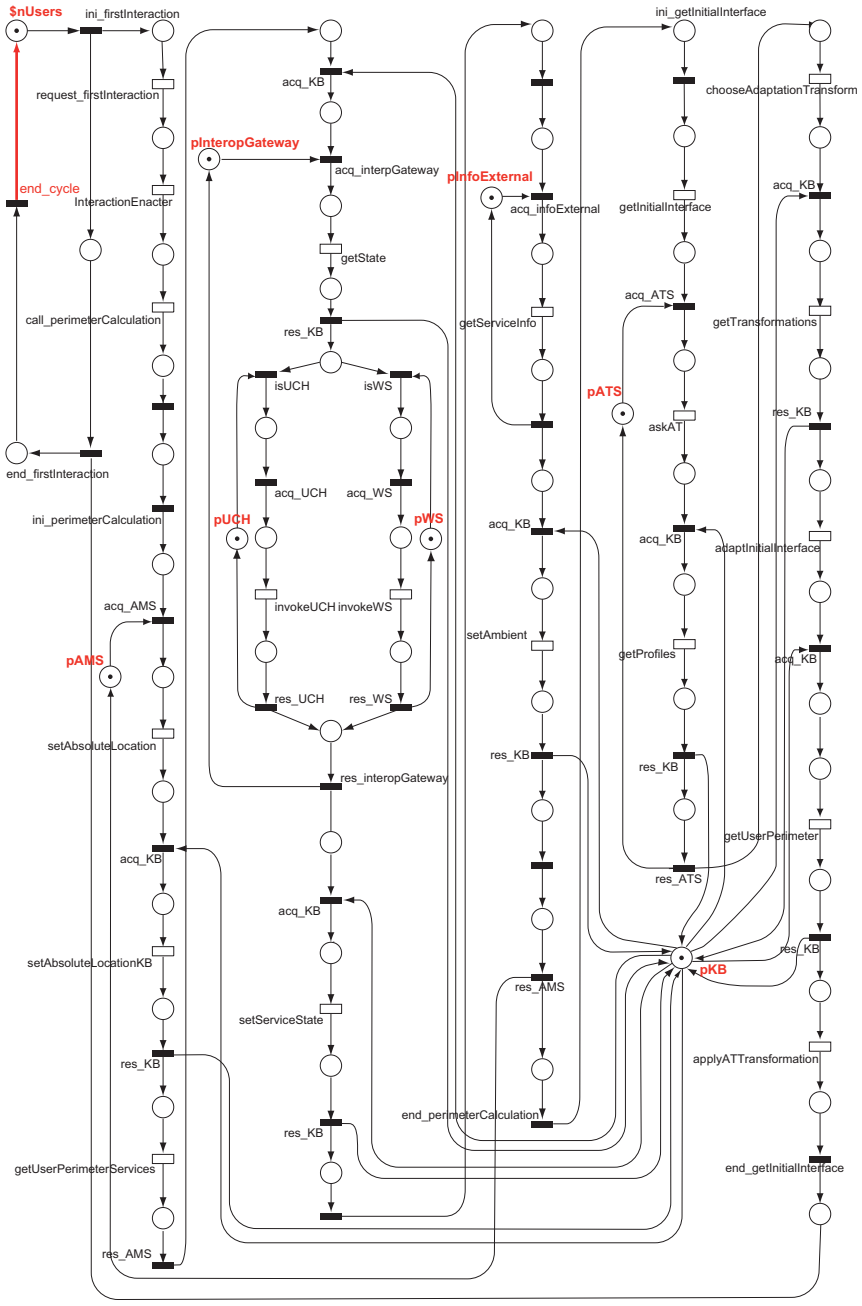

Fig. 4 GSPN representing `First Interaction` scenario.

4.3 Performance Analysis

In this step, the software engineer reviews the performance objectives, which were defined during the design step, and carries out the analysis of the perfor-

mance model. Performance objectives are quantitative measures that can be computed in the performance models.

There were two important objectives in the INREDIS project: responsiveness and scalability. *Responsiveness* is the ability of a system to meet its objectives for response time (or throughput). *Scalability* is the ability of a system to continue to meet its response time as the demand for the software function increases.

Responsiveness is a property of primary importance for software systems, so we can build the right system but if it does not meet the expected response time it will not be useful from the user perspective. In general, people with special needs demand software with response times equal to people without those needs, see discussion in Section 4.3.1. However, there is a group not so demanding, those with intellectual disabilities, for which INREDIS is also intended. In any case, being the scope of INREDIS people with all kind of special needs, the adapted interfaces have to be timely created, for the user not to lose the focus. Regarding scalability INREDIS has to be deployed in very different environments, e.g., building automation, urban, leisure or financial. The number of concurrent users can vary considerably, even for the same kind of environment it changes by orders of magnitude, for example, in the building automation case we could have smart homes, asylums, hospitals or hotels. Considering that the architecture has to be the same for all environments, it needs to scale accordingly. Moreover, the architecture had to be developed by the core INREDIS team, while each partner develops target devices and specific services without worrying about scalable aspects.

Beyond these objectives, that were established by the project leaders, we determined as performance specialists, software resource *utilization* also as a performance objective, due to its relation to scalability. Utilization appropriately measures the effect of software as it scales in usage (Smith and Williams 2002b). Performance objectives based on the system capacity, static or dynamic, were not considered. Static capacity refers to how many entities of a particular type can the system store permanently. This was not an issue in INREDIS since the architecture was not targeted to store information. Dynamic capacity refers to how much demand can be placed on the system at the same time, in this case, scalability is a more general measure.

The important task in this step is the analysis of the performance models, according to the performance objectives, for obtaining results. We present such results in Section 5. Next, we discuss implications of performance objectives in the project and how these measures are computed in the GSPN models.

4.3.1 Responsiveness

From the user's perspective, the response time is the number of seconds required to response to a user request. Basic advice regarding response times has been studied by Miller (1968) and Card et al (1991), among others. The Usability Engineering principles, proposed by Nielsen (1993) establish the following intervals:

- 0.1 second is about the limit for having the user feel that the system is reacting instantaneously.
- 1.0 second is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay.
- 10 seconds is about the limit for keeping the user's attention focused on the dialogue. Users should be given feedback indicating when the computer expects to be done.

Although target audience in our system has special needs, the response times must be similar to users without those needs[6] if we do not consider the time spent by the disabled people to operate the target device. Then, in our architecture, all the expected response times should be within these intervals. Pragmatically, we will assume quantities around ten seconds as acceptable response times. Nevertheless, we know that response time may also depend on the kind of impairment the user has and on the kind of *target* device or service 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. On the other hand, it is important to note that slow response times could prove frustrating for a person with cognitive disabilities, it also has serious consequences for the usability of the system.

Computation of measures in the GSPN models We compute all the measures (response time, utilization and scalability) under steady state assumption. Steady state means that the system reaches an equilibrium, so, measures obtained will continue in the future, which is a more general assumption than transient state. In a GSPN, steady state analysis can be carried out when the net is cyclical. However, the translation of a UML sequence diagram produces an acyclical GSPN, it starts with a resource place (see in Figure 4, place $\$nUsers$) and ends with a transition for the last scenario message (see in Figure 4, transition *end_cycle*). Therefore, we need to add an arc from this last transition to the starting place, then achieving a cyclical net (see the red arc in Figure 4). Now, the scenario can be analyzed under steady state assumption.

Computation of responsiveness in the GSPN models The response time of a scenario is calculated as the inverse of the throughput of the transition that closes the entire execution cycle (see transition *end_cycle* in Figure 4).

4.3.2 Utilization

Lazowska et al (1984) defined the utilization of a resource as the proportion of time the resource is busy, or, equivalently, as the average number of customer in service. From the SPE perspective, Smith and Williams (2002b) denote the determination of software resource utilization to appropriately measure effect

[6] For example, blind people interact with tactile interfaces by means of an immediate audible feedback.

of software as it scales in usage. Therefore, resource utilization analysis detects resource saturation and potential bottlenecks when the system is highly populated and consequently, it permits to tune up the resource configuration. In our project, apart from detecting bottlenecks, it is crucial an appropriate resource utilization, since the architecture is planned to be deployed into cloud-based infrastructures, which imply pay-per-use services. Being each new instance of a thread independently invoiced, resource utilization must be optimized.

Computation of the utilization in the GSPN models Each resource, software or hardware, is represented by a place in the GSPN. The number of tokens M in the place represents the available copies of the resource. According to Sereno and Balbo (1997), the utilization of a place is given by the steady state probability that the place is non-empty. On the other hand, the average number of tokens in steady state represents the mean occupancy of the place n. Therefore, the utilization U of a place p can be calculated as:

$$U(p) = \frac{M - n}{M}$$

A resource is saturated when its utilization ratio is closed to 1. Nevertheless, Lazowska et al (1984) advice that percentages higher than 80% should be analyzed.

4.3.3 Scalability

As discussed at the beginning of Section 4.3, depending on the environment, the number of potential users in INREDIS could be small (e.g., smart home) or large (e.g., intelligent buildings, public banking). Moreover, in an embedded system, as our architecture, the scalability is not only conditioned by the number of users but also for the internal demand of resources and services. As such, the execution context is also crucial for scaling up the system.

Although implementations in large environments are not accomplished yet in INREDIS, we strive for evaluating the scalability of the architecture also in these contexts. In fact, as discussed in Section 3, SPE promotes evaluation early in the life-cycle, before implementations. On the other hand, our architecture considers not only physical devices, but also software services available on the Internet. Therefore, to cover all these cases, the system must support requests from a large number concurrent users. We will parameterize such number through the system workload, taking into account that currently around 10 per cent of the total world population live with a disability, according to United Nations[7].

[7] http://www.un.org/disabilities/default.asp?id=18

Computation of the architecture scalability in the GSPN models We determine the scalability of the system by calculating the response times using future workload intensities (Smith and Williams 2002b). The closed workload in the GSPN specifies the number of concurrent users, which is represented in each scenario by the $NUsers$ parameter. To study the system scalability, we compute the system response time varying this parameter.

4.4 Performance Assessment

In the light of the analysis results, the aim of the assessment is to introduce changes in the system for getting the best possible architecture configuration. For each assessment iteration, we consider to apply at least: resource replication, performance patterns and performance antipatterns. Next, we describe to which extend we use these techniques. Section 5 presents the results of applying the techniques in our project.

4.4.1 Resource Replication

Utilization resource analysis detects resources which would be potential system bottlenecks. In that case, resource replication is a choice. Software replication relies on multithreading to serve multiple requests in parallel. Nevertheless, this solution does not always work, since the bottleneck can be in the hardware resources, such as I/O devices or CPU capacity. In the latter case, the solution will be to add more CPU capacity (e.g., adding additional computational nodes).

Replication is modeled in the GSPN by populating the resource place with new tokens.

4.4.2 Performance Patterns and Antipatterns

Gamma et al (1995) defined a pattern as a common solution to a problem that occurs in many different contexts. Thereby, patterns provide generic solutions for many architectural, design and implementation problems. Smith and Williams (2002b) proposed performance patterns, which are inspired by design patterns and describe best practices for producing responsive and scalable software. Table 2, in Appendix C, summarizes performance patterns, other important design patterns can be found in (Grand 1998, 2001; Lea 1999; Schmidt et al 2000).

Antipatterns extend the notion of patterns to capture design errors and their solution (Brown et al 1998). Smith and Williams (2000) defined performance antipatterns as "bad practices" that affect software performance in a negative way. Table 3, in Appendix C, summarizes some of the antipatterns gathered and analyzed by Smith and Williams (2000, 2001, 2002a, 2003). Each antipattern is characterized by its name, problem and textual solution description. Cortellessa et al (2012) formalized this description, by means of logical

predicates, in order to systematize their identification. Thus, they built an engine to automatically detect performance antipatterns and to refactorize them. We follow Cortellessa's approach to automatically identify performance antipatterns in the INREDIS software architecture.

5 Results

This section develops our *internal objective*, hence, we pursue performance results for assessing and eventually improving the INREDIS architecture. Our first objective is the validation of the performance models. Later, we get results from the valid models to assess an optimal system configuration according to the performance objectives of the project.

5.1 Empirical Results and Validation of the Performance Models

In order to validate the architecture for running prototypes, some pivotal pieces (modules) were implemented and tested within the INREDIS project. Tests considered diverse users disabilities, preferences and profiles. Catalán and Catalán (2010) tested the architecture experimentally, by using a set of user controlled tests. The main challenge was to measure the satisfaction of the user experience with diverse interaction modes of services and devices for people with special needs. This level of satisfaction included usability aspects as well as performance objectives. Additional experimental results can be found in (INREDIS 2010).

As described in Section 2, the main modules making up the INREDIS architecture are: Interface Generator, Knowledge Base, Assistive Technology Server (ATS) and Adaptive Modelling Server (AMS). For experimental evaluation, only the Interface Generator module was completely implemented. The Knowledge Base module, which stores ontologies, was very low populated, only with basic knowledge mechanisms, a minimum for experimentation. The ATS was implemented to support only the users profiles and interaction modes required for the tests. Finally, the basic functionality of the AMS was implemented. These modules were executed to carry out the four system scenarios (First Interaction, Navigation, Back to Top and Device Interaction), depicted in Figure 2. The experimental tests were targeted to analyze the responsiveness and scalability of the system. The tests were mainly focused on interoperability and usability. Figure 5 shows the measured average response times of these key performance scenarios in this user testing phase.

However, the burden of real experimentation with complex interoperable architectures, elderly people, and people with special needs, raised quickly and it greatly limited the evaluation. In this user testing phase, the number of concurrent users never exceeded 5 due to the logistical difficulties of real experimentation, as Sainz et al (2011) described. On one hand, some of the users needed caregiver or additional assistive products, also the Spanish Law fixes

directives for working with this kind of users. On the other hand, some tests were made individually to specifically study user interactions. Finally, the cost of the team for supporting the experiments and the facilities (e.g. smart home) were also important issues for carrying out more complex experimentations.

Hence, experimentation problems and limitations of real implementations advocated the use of models, specially in these initial phases of the system life-cycle. Models can represent the system in a variety of hypothetical situations and can perform analysis at a lower cost. SPE, as summarized in Section 3, offers techniques and tools that can overcome these problems.

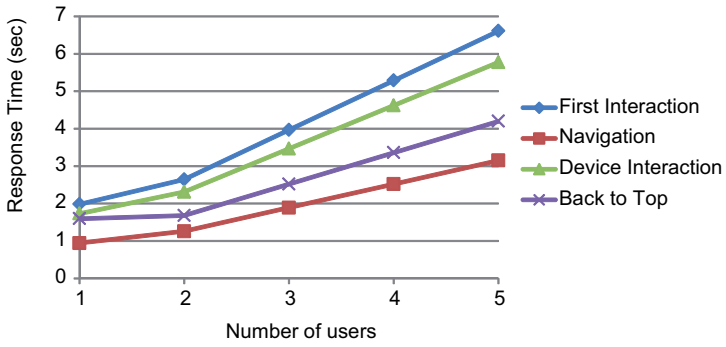

Fig. 5 Empirical response times for a set of test users.

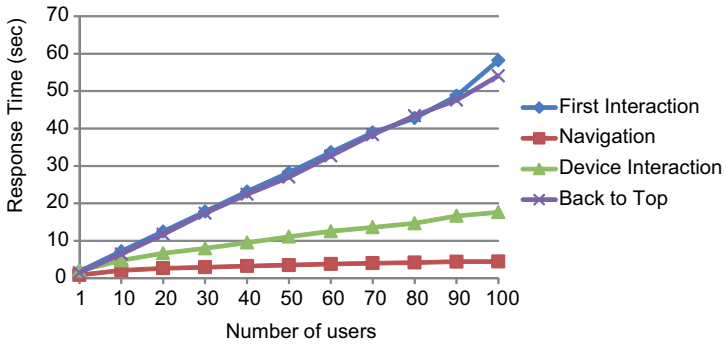

Fig. 6 Response times for concurrent users using models.

We then reproduced these experiments using the performance models obtained by the second step of the methodology. We got the results in Figure 6. Note that using models we obtained results for one hundred users, which was enough for our purposes. We could have obtained results for larger populations using the same GSPN models by changing the workload.

Table 5.1 compares for each scenario the results obtained in real experimentation (*Real* rows) with those obtained by our GSPN models (*Model* rows)

We appreciate that differences (*Var.* rows) between our models and real experimentation are around a five percent in most cases, differences never went beyond ten per cent, except for the Device Interaction scenario in the case of three concurrent users. In this latter case we assume that the variation might be caused by the accuracy level in the computation of the GSPN models. Moreover, we observed that tendencies in the graphs were similar. So, we can assume that our performance models can be useful to address experiments initially not feasible to carry out with the real implementations.

Table 1 Results in seconds for each scenario

		Number of users							
Scenario		1	2	3	4	5	10	50	100
First	Real	1.983	2.644	3.966	5.287	6.609	NE	NE	NE
Interaction	Model	1.841	2.674	3.776	4.912	6.210	7.212	28.229	58.231
	Var.	$\simeq 5\%$	$< 5\%$	$\simeq 5\%$	$> 5\%$	$> 5\%$	-	-	-
Navigation	Real	0.945	1.260	1.891	2.521	3.151	NE	NE	NE
	Model	0.875	1.130	1.684	2.338	2.793	3.109	3.520	4.470
	Var.	$> 5\%$	$\simeq 10\%$	$\simeq 10\%$	$\simeq 10\%$	$\simeq 10\%$	-	-	-
Device	Real	1.732	2.310	3.465	4.619	5.774	NE	NE	NE
Interaction	Model	1.808	2.351	2.994	3.937	3.380	4.752	11.114	17.648
	Var.	$\simeq 5\%$	$\simeq$	$> 10\%$	$\simeq 10\%$	$\simeq 10\%$	-	-	-
Back to	Real	1.596	1.679	2.519	3.359	4.199	NE	NE	NE
Top	Model	1.497	2.121	2.745	3.369	3.993	6.488	27.013	54.093
	Var.	$\simeq 5\%$	$\simeq 10\%$	$> 5\%$	$\simeq$	$< 5\%$	-	-	-
		NE - The experiment could not be carried out							

Discussion of the Results: Performance View

First, we note that, as requested in Subsection 4.3.1, the experiments (both, real and GSPN) did not consider the time spent by the disabled people, neither the time to operate the target device or service[8].

The discussion about what could be considered a good response time was introduced in Subsection 4.3.1 from the Usability Engineering point of view (Nielsen 1993). Pragmatically, we decided that quantities around ten seconds could be considered as acceptable response times. From results in Figure 6, we observe that both Device Interaction and Navigation scenarios have acceptable response times. The Navigation scenario never goes beyond six seconds, while the Device Interaction scenario is below ten seconds until it reaches forty concurrent users. However, Back To Top and First Interaction scenarios perform poorly. The reason is that both of them must calculate the user context perimeter, which depends on the number of devices or services, and their corresponding available operations. Therefore, our assessment loop, developed in next subsection, concentrates on how to decrease response in these scenarios mainly.

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

5.2 Results of the Performance Assessment

Alternatives discussed in Section 4.4 for improving responsiveness were: resource replication (using utilization resource analysis) and application of performance patterns and antipatterns. In the following, we conduct the study following these alternatives for getting an "optimal system configuration".

5.2.1 Resource Replication

Utilization resource analysis detects those resources/tasks[9] which would be potential system bottlenecks. Sub-section 4.4.1 explained how we compute utilization in the GSPNs. Our analysis considered that some resources were shared between several scenarios (e.g., the Adaptive Modelling Server (AMS), which appears in the Back To Top and in the First Interaction scenarios).

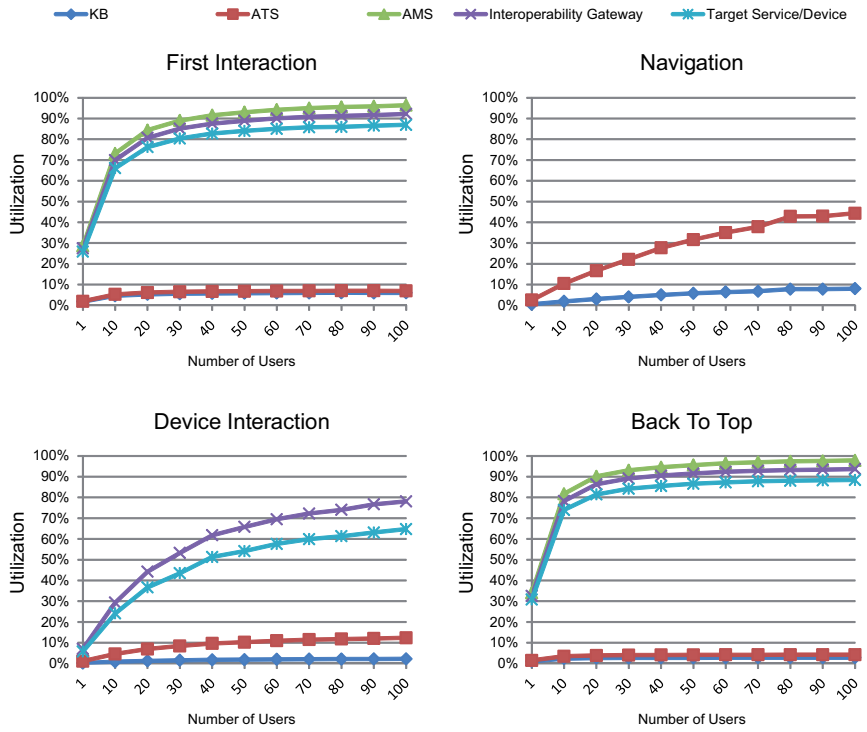

Fig. 7 Resource utilization of each key scenario.

Then, for each of the four key scenarios, we obtained the utilization of all resources involved. However, Figure 7 depicts only the utilization of some

[9] We recall that resources are represented: a) in the sequence diagrams by life-lines, b) in the GSPN by shared places, highlighted in red in Figures.

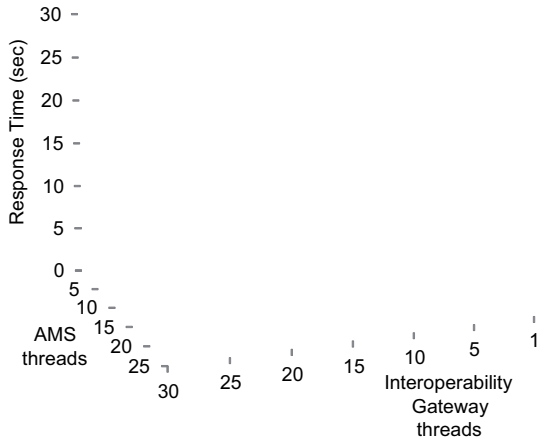

Fig. 8 Response times when multithreading AMS and Interoperability Gateway.

resources, to avoid cluttering. As we can observe, in the First Interaction and Back To Top scenarios, both Interoperability Gateway and AMS resources are highly saturated, with maximum utilizations of 94% and 98% respectively.

We replicated resources (added threads) for the AMS and the Interoperability Gateway and computed response times for the Back to Top scenario. Figure 8 presents results obtained for the case of 50 users (which is representative of all the experiments we performed). We can observe that the response time does not improve, it is around 30 seconds, same as in Figure 6 where no replication was introduced. We thought that saturation could be caused not only because of these resources. Therefore, we computed resource utilizations, for all the possible multithreading situations, in the Back to Top scenario.

Figure 9 presents only a representative part of these results. It depicts the case of 50 users, with a variable number of threads of the Interoperability Gateway and the AMS, for the rest of the resources it considers one thread only. As observed, the AMS and the Interoperability Gateway are no longer saturated. However, the Target Service/Device becomes saturated. This resource, although not initially considered, appears in different scenarios, e.g. Device Interaction (in Figure 24) or Perimeter Calculation (in Figure 20).

We performed all the experiments again, from one to one hundred users, in the Back to Top scenario. In this case, replicating threads for the AMS, the Interoperability Gateway and the Target Service/Device. Figure 10 shows the results for the case of 50 concurrent users. Now, the response time has reached an acceptable threshold according to the usability principles, around 5 seconds in the best situations. We perform experiments, although not depicted in the figure, and observed that, from 30 threads on, the system did not perform better.

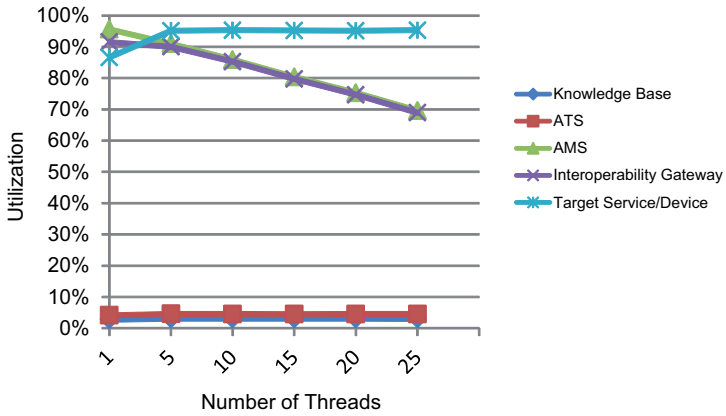

Fig. 9 Resource utilization when threading Interoperability Gateway and AMS.

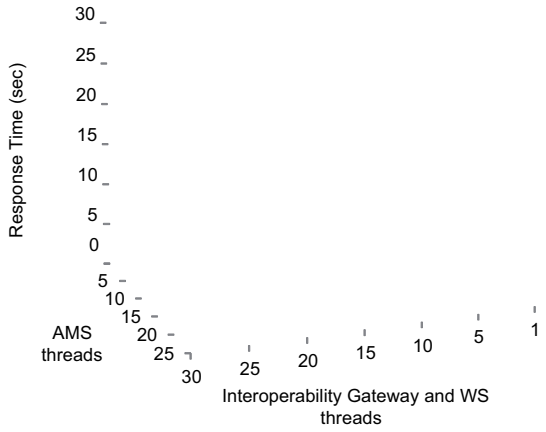

Fig. 10 Response times when multithreading AMS, Interoperability Gateway and Target Service/Device.

Finally, once we had identified all critical resources (AMS, IG and Target Service/Device), we replicated them, according to our investigations, and computed response times in all the scenarios. Figure 11 presents these results, it shows that the response times have significantly improved.

5.2.2 Performance Patterns

Although the results obtained satisfied the usability principles, our objective, at this stage, was to discover whether we could improve system responsiveness and scalability. We then aim at applying some performance patterns to the architecture design. We used the algorithm proposed by Bergenti and Poggi (2000) and applied it throughout the architecture. As a result, we found that

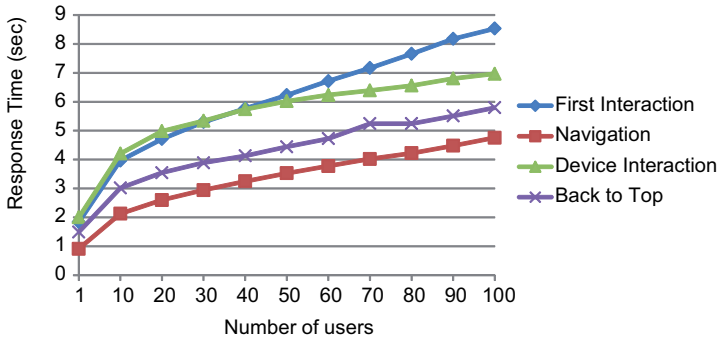

Fig. 11 Response times when multithreading AMS, Interoperability Gateway and Target Service/Device for concurrent users.

the Fast Path performance pattern could be applied for improving the Perimeter Calculation process, one of the processes most used in the system. In fact, this pattern caters to the centering principle, which means to focus attention on the performance of the scenarios that are exercised the most or have large performance impact. The Fast Path pattern is summarized in Table 2 in Appendix C.

Perimeter Calculation, depicted in Figure 20, is a process used by the First Interaction and Back to Top scenarios, which were compromising system responsiveness. The *perimeter* represents the list of devices and services available and this process updates the status of each device and service by consulting the Interoperability Gateway. This is shown by the two consecutive loops in the sequence diagram in Figure 20. Therefore, if the number of available devices is x and each of them has f functionalities on average, the second loop is executed $x \times f$ times. The Fast Path can be applied here for providing an alternative execution path which minimizes the steps of execution or dedicates more resources here than to other scenarios. Consequently, response time should improve.

One manner to apply the Fast Path pattern in the INREDIS architecture is to request only those functionalities that will be displayed (e.g., if a user would like to control the air conditioning, it would be not necessary to load leisure services). In other words, the user's context (or locality) must be taken into account in order to calculate the perimeter. According to the deployment tested, which had a total of 30 target devices or services and each of them had about 2 or 3 functionalities, if the Fast Path pattern is applied, then the number of times that loops are executed is reduced around 60% on average, which significantly reduces the response time.

We applied the Fast Path pattern in our architecture design and obtained new performance models for the First Interaction and Back to Top scenarios. We carried out the whole set of experiments with the new performance models, but without taking into account the multithreading discussed in pre-

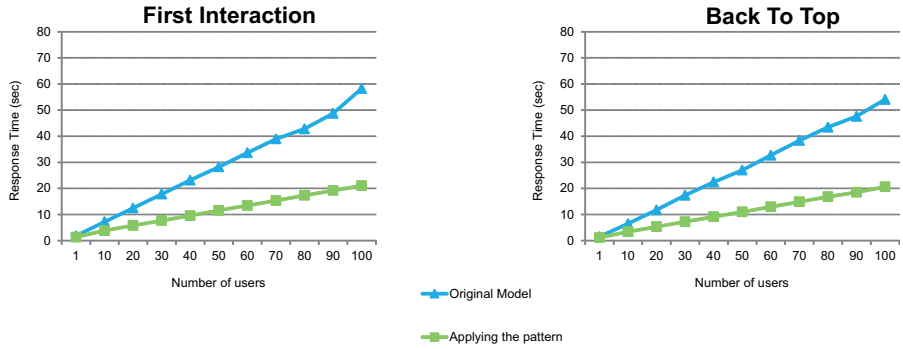

Fig. 12 Comparison of response times when applying Fast Path pattern: baseline and new results.

vious section, since we wanted to know how much, by itself, the performance pattern could improve system responsiveness. Figure 12 shows these results. If we compare them with those in Figure 6, we observe that response times, although not fitting the usability principles yet, are less than half.

5.2.3 Performance Antipatterns

As mentioned in Section 4.4.2, we used the logical predicates defined by Cortellessa et al (2012) in order to systemize the identification of performance antipatterns in our architecture. After applying all these logical predicates, we detected The Ramp antipattern in the Assistive Selection Software Mechanism (ASSM)[10]. The problem arises since the ASSM searches incrementally in the Knowledge Base. The ASSM was completely developed in the INREDIS project, however it was populated with only ten Assistive Products in the testing phase[11]. Figure 13 depicts the response times for the ASSM obtained in the user testing phase.

The Ramp antipattern, summarized in Table 3 in Appendix C, *occurs when processing time increases as the system is used*. Cortellessa et al (2012) formalized it as follows:

$$\exists OpI \in \mathbb{O} \mid \frac{\sum_{1 \leq t \leq n} |F_{RT}(OpI,t) - F_{RT}(OpI,t-1)|}{n} > Th_{OpRtVar} \quad (1)$$
$$\wedge \frac{\sum_{1 \leq t \leq n} |F_{T}(OpI,t) - F_{T}(OpI,t-1)|}{n} > Th_{OpThVar}$$

where:

[10] The ASSM process was introduced in Section 2 and it is detailed in Appendix A.5.

[11] According to EASTIN (www.eastin.eu), the principal Assistive Technology Information Network in Europe, the number of Assistive Products available in the EU increased to more than 39.221 products in 2009.

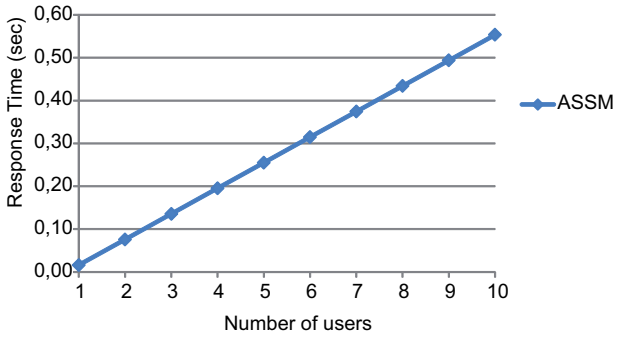

Fig. 13 Empirical response times for the Assistive Software Selection Mechanism.

- OpI is an operation instance whose response time increases along n time slots,
- $\mathbb{O}$ represents the set of all operation instances in the system,
- F_{RT} and F_T are functions that respectively compute the mean response time and throughput of an operation instance observed in a time slot,
- $Th_{OpRtVar}$ and $Th_{OpThVar}$ are thresholds for the response time and throughput, respectively.

The Ramp occurs when the average response time and throughput of the operation increases in n consecutive time slots and the increments overmatch some predefined thresholds.

The critical operation, in the ASSM, is the incremental search in the Knowledge Base. The ASSM process affects all four key performance scenarios since it is called by two subscenarios, Initial Interface Generation and Interface Generation. The former subscenario belongs to the First Interaction scenario, while the latter is present in the Navigation, Device Interaction and Back to Top scenarios.

On the other hand, Smith and Williams (2002a) determined the following relation in The Ramp:

$$RT = \frac{i \cdot \frac{ds}{dt} \cdot s}{1 - X \cdot (i \cdot \frac{ds}{dt} \cdot s)} \tag{2}$$

where:

- RT is the response time of the operation,
- i is the number of items in the data set of the operation,
- s is the amount of service time required to process a single item,
- $\frac{ds}{dt}$ is the slope of the ramp,
- X is the arrival rate of queries to the operation.

Combining equations 1 and 2, we get the response time for the operation:

$$RT_{OpI} = \frac{i \cdot (F_{RT}(OpI,t) - F_{RT}(OpI,t-1)) \cdot F_{RT}(OpI,1)}{1 - X \cdot (i \cdot (F_{RT}(OpI,t) - F_{RT}(OpI,t-1)) \cdot F_{RT}(OpI,1))} \quad (3)$$

Taking the experimental results obtained for the ASSM in Figure 13 and applying equation 3, we calculated response times, in the four key scenarios, for 100 users. As it can be observed in Figure 14, The Ramp antipattern greatly impacts in the response times. The reader should note that estimated response times in Figure 6 did not take into account the effect of The Ramp, since the Knowledge Base was very few populated, only with ten Assistive Software (AS) products. Therefore, we detected the impact in this phase since the Knowledge Base was populated with ten thousand AS products[12]. Consequently, the response times in Figure 14 are so different from those in Figure 6.

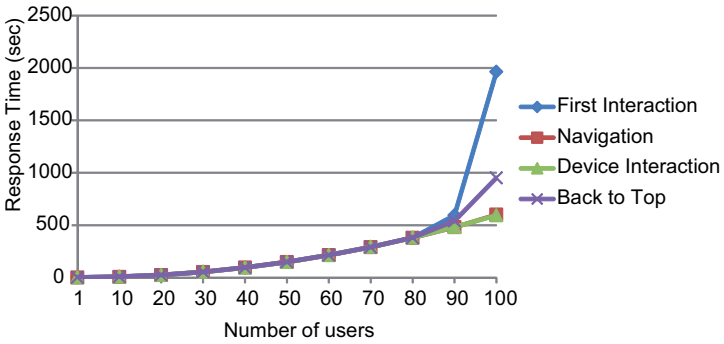

Fig. 14 Impact analysis of The Ramp antipattern in the response times of key scenarios.

To solve this antipattern, both Smith and Williams (2002a) and Dugan-Jr. et al (2002) propose to select another search algorithm more appropriate for large amount of data. The ASSM is based on a simple filtered search in SPARQL. This search can be improved by changing the recommender process and using a specific "recommend" operator, as Levandoski et al (2011) suggest. Thus, the response time for a search performs better in 33%, independently of the number of users. We then recalculate response times for the scenarios considering the improvement in the search algorithm. Figure 15 shows the results, which considerably improve those in Figure 14.

5.3 Optimal Configuration

Once all the alternatives for improvement were analyzed, we applied them to the original configuration of the INREDIS architecture, in order to achieve an optimal configuration. These improvements are summarized in the following:

[12] From all the Assistive Products in the marketplace, we considered those that can be integrated into the architecture, i.e., Assistive Software products.

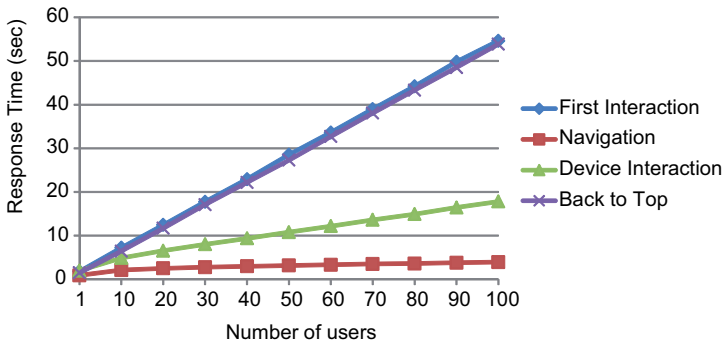

Fig. 15 Response time applying specific "recommend" operator in search algorithm of ASSM.

- *Utilization and Multithreading*: We detected the AMS, the Interoperability Gateway and the Target Service/Device resources as bottlenecks. They were mitigated by adding threads as indicated in Figure 11.
- *Performance patterns*: Applying performance patterns helps to improve the software design and the system performance. We identified the Perimeter Calculation subscenario as candidate for the Fast Path pattern, then we refactorized this scenario in order to apply the pattern.
- *Performance antipatterns*: We detected The Ramp antipattern using the logical predicates in (Cortellessa et al 2012). Then, we analyzed its potential consequences and changed the search algorithm in ASSM process.

Figure 16 depicts the response times when we applied to the system model all the aforementioned alternatives for improvement. As it can be observed, all these improvements help to meet performance objectives based on Usability Principles. Otherwise, another iteration of the assessment loop would have been necessary.

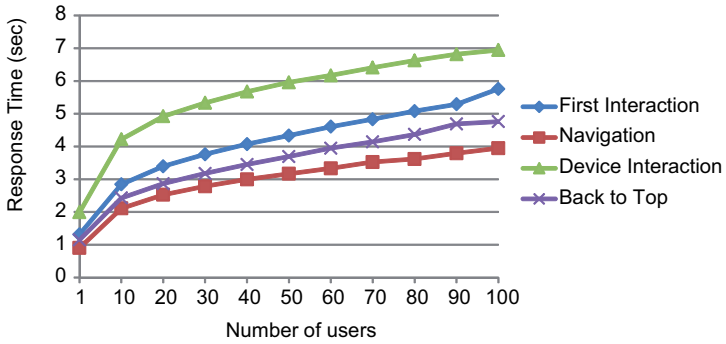

Fig. 16 Response time for the optimal configuration of the INREDIS Interoperable architecture.

5.4 Validation of the Performance Assessment

Once our assessment has produced an optimal design of the architecture, we are committed to apply the improvements to our initial prototype. By doing so we want to assess whether the results of the model match the results of the architecture. Next we detail the important issues in the new prototype implementation:

- *Resource replication*: We fully applied to the initial prototype all the proposed improvements. We replicated the resources and applied multithreading to overcome all detected bottlenecks.
- *Performance patterns*: We satisfactorily applied the Fast Path pattern for perimeter calculation, then modifying the location of services and target devices. Hence, this assessment was also completely applied.
- *Performance antipatterns*: We could not apply this assessment in the prototype since the update of the search algorithm was very complex and affected other processes (out of scope of the INREDIS project). However, the effects of the assessed antipattern were almost negligible in the results of the initial prototype because the Knowledge Base was populated with only ten Assistive Products, as previously explained.

Figure 17 (last line in the caption) plots the response times of this optimal prototype. The prototype was deployed in the same servers as the initial one and the experiments replicated in the same facility (automation house). As described in Section 5.1, the experiments were very difficult to carry out due to several issues (legal, logistic or user selection among others), in this case the situation was even worst since we were out of budget. We could involve three concurrent users, hence the results were extrapolated for five users through a linear function. We observe in Figure 17 that the results of our optimal model and those of the optimal prototype are very similar. In some scenarios our model is slightly more optimistic but slightly more pesimistic in others. As mentioned in Section 5.1, these low variations between empirical and predicted results might be mainly caused by the accuracy in the computation of the GSPN models.

5.5 Comparison of Results

Figures 17 and 18 have been introduced to depict a throughout comparison of all the results obtained so far. Figure 17 plots for each scenario the following information:

- The empirical results obtained with the initial prototype, i.e., it replicates the information presented in Figure 5.
- The results we obtained using the initial model, i.e., it replicates the information presented in Figure 6, but for five users only, due to the aforementioned limitations of experimental user tests.

– The results we obtained using the optimal model -the model of the optimal configuration-, i.e., it replicates the information presented in Figure 16, but for five users only.
– The results obtained with the optimal prototype. The optimal prototype is the initial prototype plus the improvements obtained by our assessment. Subsection 5.4 explained how we developed the optimal prototype.

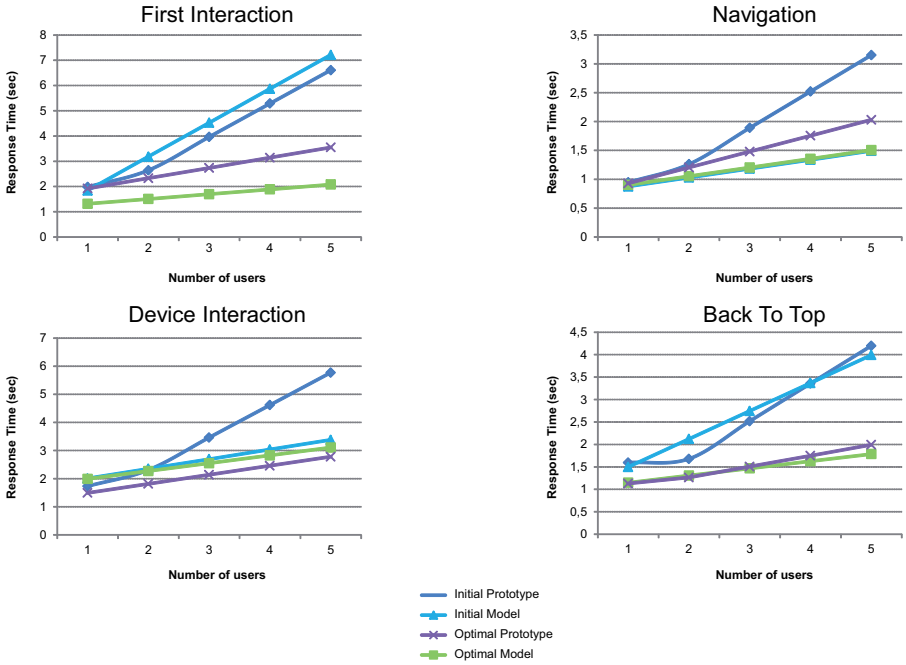

Fig. 17 Comparison between initial results (prototype and model) and results of the optimal configuration (prototype and model).

Figure 17 shows, for all the four scenarios, that the results given by the optimal model improves both, the initial empirical results and the initial model prediction.

The low differences between empirical and predicted results might be mainly caused due to the accuracy in the computation of the GSPN models. Nevertheless, although empirical and predicted data have similar trend for almost all graphs, it is observed that they differ for the Navigation and Device Interaction with the initial prototype, particularly in the latter case. However, for the optimal prototype the trend is the same in all scenarios. We guess that this could be due to imprecisions in gathering data during the testing phase session, corresponding to the initial prototype, since there were great variabilities in all actions carried out by users. However, for getting data from the models we do not express such variability since we only used average execution times. All in all, in the first user testing phase, the number of users involved

was small but they had not any restriction to interact with the system. However, tests with the optimal prototype were more controlled. Therefore, the quality, more than the quantity, of the empirical data used determined the worth of the predicted results.

Figure 18 extends results in Figure 17 for 100 users, the information empirically obtained with both prototypes is missing since it could be obtained only for 5 users. Figure 18 shows that the results of the optimal model are far better than those of the initial model. These figures clearly demonstrate that the changes, due to the application of the SPE approach, have improved the models to the degree of compliance with the performance requirements for a large number of users.

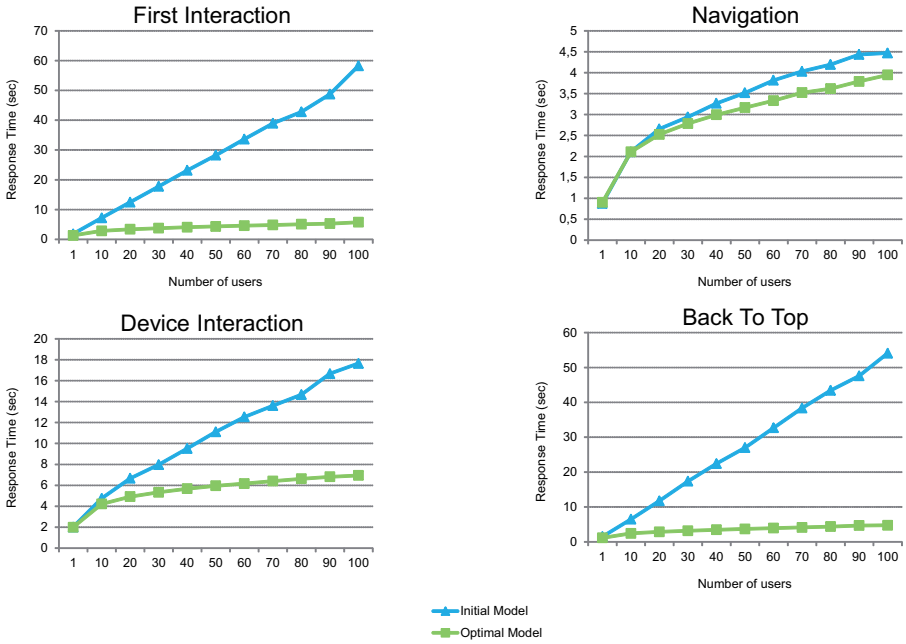

Fig. 18 Comparison between results of the initial and optimal models.

6 Discussion

The assessment of software architectures is a process that is acquiring increasing importance in industrial practice. The work carried out allowed us to determine an optimal configuration and, therefore, to improve the final product. In the following, as suggested by Runeson and Höst (2009), we discuss limitations of the results obtained, lessons learned and issues disclosed while applying the assessment process, we also explain some of the consequences of all these matters.

The outcomes of this research can be interpreted from two perspectives. First, we discuss the outcomes explicitly related to the assessment methodology, what we denoted as *external objective*. Second, we analyze the collected data obtained by applying the methodology in order to achieve an optimal configuration, what we called the *internal objective*.

6.1 External Objective

Section 4 carried out the external objective, which meant to carefully revise each step of the methodology to teach practitioners how we applied these steps, but also to offer a blueprint of the INREDIS project that could be used as a guide for practitioners in future applications.

We discuss issues of the methodology according to the evaluation criteria proposed by Isa and Jawawi (2011), which consider: process related aspects and modeling related aspects. Finally, we also consider issues related to the tools used.

Concerning the process for performance assessment, the SPE methodology is intended to support general-purpose domains. The methodology explicitly influences the development process by focusing on performance properties. In particular, the assessment process manages the system performances from the requirements and analysis phases until the design phase by analyzing a set of key performance scenarios. The methodology systematically defines all the steps needed to discover potential performance problems and how to mitigate them.

Another aspect related to the process concerns to the tradeoffs that the engineer needs to consider for achieving performance. Bass et al (2005) defend that quality attributes can never be achieved in isolation, the achievement of any one will have an effect, sometimes positive and sometimes negative, on the achievement of others. In fact, this happened when we applied the SPE methodology, the improvement of system performance influenced other quality attributes, such as maintainability or cost, and in the worst case, the improvement of performance decreased other quality attributes. In particular, for improving performance we introduced performance patterns, in this case the tradeoff positively influenced the maintainability of the system since it is widely recognized the benefit of using design patterns for this quality of the system. Regarding antipatterns we can say the same. They capture design errors, therefore, by using them we not only gain in system performance but eventually in maintainability and system testability. On the other hand, when we replicated resources, we incurred in a cost, i.e., the influence of improving performance was negative. For example, replication of CPU capacity implies a monetary cost, while multithreading implies a software more difficult to test and maintain.

Concerning the modeling criterion, it analyses how the performance requirements and system functionalities are specified and developed. Being the approach centered on the architectural level, it perfectly captures the system

structure model definition and the behavioral issues, then allowing assessment of the complete software architecture. As mentioned, we used a simplified version of the PUMA methodology (Woodside et al 2005, 2013), which addresses a systematic performance modeling with the support of UML, which allows annotations with MARTE profile for the performance properties. Nevertheless, the application of the complete PUMA approach is limited, mainly due to the characteristics of the INREDIS project. One of these characteristics is the discrete number of users and resources, thus, the workload identification only considers closed ones with discrete values. A similar situation occurs with resources, since we did not model some low level issues, such as random access memory, disk storage or cache memory.

Concerning the tools, we used ArgoSPE (Gómez-Martínez and Merseguer 2006), an ArgoUML[13] plugin, to partly automate the assessment process in a transparent way for software architects. Unfortunately, performance annotations supported by ArgoSPE are not in MARTE, but in UML-SPT (2005) profile format. Thereby, we had the choice of translating the annotations into UML-SPT or to introduce some performance parameters in the GSPN manually. For example, the number of system resources, we solved it looking at the GSPN places representing resources, such as $pATS$, pWS or pKB in Figure 4, then populating them with as many tokens as resources indicated in the MARTE annotation *resMult* (Figure 1).

ArgoSPE internally calls GreatSPN (Chiola et al 1995) to analyze or simulate GSPNs. We used simulation programs since the large size of the models prevented the analysis programs. The problem stems from the reachability graph of the GSPN. Simulation outcomes can be obtained almost immediately for simple samples. However, the computation times for some of the results obtained in this paper consumed long time (several hours), even some of them lasted for a couple of weeks. Concretely, those for which the number of concurrent users was greater than 50 and some resources were multithreaded. To reduce the computation times, we decreased the simulation default accuracy, i.e. the precision of the approximation in the parameters estimation. This reduction affected the response times, i.e, the results we obtained, in the order of ± 10 milliseconds. Although this significantly decreased the computation times, few of the experiments lasted for two or three hours yet.

Moreover, ArgoSPE lacks other plugins, such as tools for identifying performance patterns and antipatterns automatically. Thus, we had to use external applications manually, as those developed by Cortellessa et al (2012). Therefore, ArgoSPE is still a very limited tool for performance assessment, specially for complex case studies such as the INREDIS architecture. Consequently, we have detected the need for developing a new framework which integrates all these functionalities: UML modeling, GSPN simulation and analysis, patterns and antipatterns detection in a transparent way to the user and efficient computation times. This framework could also include assessment of other

[13] http://argouml.tigris.org/

functional and non-functional properties, such as dependability, security or model checking.

6.2 Internal Objective

The internal objective, developed in Section 5, tried to reveal how good the architecture proposed by the INREDIS software engineers was, from the performance point of view exclusively. The performance results demonstrated that the original architecture and configuration fitted for limited contexts with very few concurrent users. This usage scenario might occur, for example, when users interact with electronic devices at smart homes or students in a classroom. Nevertheless, our results disclosed that this configuration performs poorly in contexts with several concurrent users. Thus, we systematically assessed system performance by changing the software design and the configuration, concretely adding threads and refactoring some components. These improvements helped to meet performance objectives as well as to scale the system in more challenging performance usage scenarios, such as web services, urban networking, hospital and/or retirement homes, where multiple users with different capabilities can simultaneously access the system.

However, some limitations are still unsolved. First of all, in real implementations, the Knowledge Base had not been fully populated with users preferences and capabilities, Assistive Software products in the ASSM and interaction modes. Therefore, a specific sensitivity performance analysis of the Knowledge Base and ASSM would be desirable.

A more detailed study of the target devices or services would also be desirable, since both their usage and their corresponding functionalities can affect the architecture. In this paper, we have assumed that this time is negligible, since it is independent of the 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-controllable part of our system. However, real implementations did not consider increments in the number of devices or services in the user perimeter. Thus, as noted in Sec. 5.2.2 through our experiments, the number of nested iterations in the Perimeter Calculation depends on the amount of available target devices and services and their functionalities.

Finally, as above mentioned, the architecture was implemented considering most flexible and cutting-edge technologies at that moment. However, some of the communication protocols used had poor performance, such as the ESB (Enterprise Service Bus) architecture designed by Chappell (2004) combined with services implemented in SOAP (Liu et al 2007). A similar situation can be found in the Knowledge Base component, as observed by Liang et al (2009). Consequently, our performance models considered the measured times of these prototype implementations. In particular, we used them as host demands for the Petri net transitions. However, it would be feasible to include an additional stage in our performance assessment proposal, which carries out sensitivity analysis to assess technological alternatives for implementation.

7 Related Work

Software architecture assessment constitutes an important stage in the software design process, in order to guarantee non-functional requirements. Nevertheless, to the best of our knowledge, there are very few initiatives to assess architectures based on SPE principles at industrial level. An exception is the PASA (Performance Assessment of Software Architectures) method, proposed by Williams and Smith (2002).

PASA, focussed on performance scenarios, is a performance-based software architecture analysis method that provides a framework for the whole assessment process. PASA inspired us in order to automatically systematize the process to detect performance issues, as well as to propose the corresponding potential solutions. As in PASA, our methodology, carries out performance analysis considering responsiveness, but also resource utilization and scalability. Moreover, we have included automatic detection of performance patterns and antipatterns, by considering the work of Cortellessa et al (2012). As above stated, PUMA (Woodside et al 2005, 2013) also guided our work.

Pooley and Abdullatif (2010) defined Continuous Performance Assessment of Software Architecture (CPASA). This method adapts PASA to the agile development process. To the best of our knowledge, CPASA has not been applied to an industrial case yet.

Regarding industrial experience reports that assess performance at architectural level, we have found a few:

- Kauppi (2003) conducted a case study using PASA for analyzing mobile communication software systems. They used Rate Monotonic Analysis and layered queuing networks (LQN, Woodside et al 1995) instead of Petri nets for system analysis. Results of improvements were not explicitly given due to the confidentiality of the project.
- Koziolek et al (2012) reported their experience on performance and reliability analysis in a large-scale control system. They applied the method Q-ImPrESS (2009) (Quality Impact Predictions for Evolving Service-oriented Systems), which is supported by an IDE that combines tools for creating and editing models, performing predictions, and conducting a tradeoff analysis. LQNs were used for performance prediction, results were impressive for throughput estimation since they deviated only around 0.2 percent. The authors explain that such good results were obtained because resources were not saturated.
- Huber et al (2010) described an industrial case study where they applied the Palladio Component Model (Becker et al 2009) to a storage system. A model was firstly implemented, next they conducted several experiments on a prototype to derive the resource usage of each model component and finally, the model was calibrated with realistic resource demands and validated.
- Kounev (2006) modelled a new industry-standard benchmark for measuring the performance and scalability of J2EE hardware and software platforms. In this work Petri nets are used as performance model. The method-

ology is also based on SPE principles, however they did not explicitly use patterns and antipatterns, as we do. They could implement the system and the models accurately reflected the real system performance.
- The work of de Gooijer et al (2012) re-architects a legacy system, for remotely diagnose industrial devices, in ABB company. The goal was to improve system performance and scalability. The problems addressed to re-architect a system for performance are very different, although not easier, to those to design for performance from scratch, as it was our case. They could start from real system measurements to calibrate their models, which were constructed using the Palladio Component Model and translated to LQNs. They used the PerOpteryx (Koziolek et al 2011) tool to find new architectural candidates, in contrast we used patterns and antipatterns.
- Jin et al (2007) developed an approach that combines benchmarking, monitoring and performance modeling for database-centric legacy information systems. As in the work previously analysed, important challenges relate to measuring the production system to calibrate the model. They could not match their predictions with the planned system since the implementation was not ready, but established an accuracy of their models within 8%. This work uses a performance model different to ours, in particular they use LQNs. Also different is the application domain, concretely they target the approach towards "legacy systems" in the database field.
- Liu et al (2005) developed a methodology for component-based applications to predict their performance under various workloads. As in our approach patterns play an important role, but at architectural level in this case. Patterns are modeled by means of UML activity diagrams, however system scenarios are modeled using UML sequence diagrams as in our approach. Queuing networks were used for prediction. To verify the approach they implemented different systems and stablished errors of prediction around 11 and 15 percent.

Concerning related work about the application domain of the INREDIS architecture, i.e, adaptive interfaces for people with special needs, we have found the following:

- The INREDIS architecture further develops the idea of Universal Control Hub (UCH) proposed by Zimmermann and Vanderheiden (2007)(which is also aligned with the initial ideas that Llinás et al (2009) propose on how disabled people can take advantage of adaptative interfaces from the ubiquitous computing perspective).
- Kadouche et al (2009) proposed a semantic framework to enhance environment services for people with special needs, namely the SMF (Semantic Matcher Framework). Our proposal bares similarity with this work, but even though we share the use ontologies for representing the elements (both implemented in the OWL (W3C 2012) representation language), on the one hand our approach makes the reasoning at a class level to reason with taxonomies of concepts and relationships; and the other hand, we take into account assistive software in our process whereas the SMF does not.

– The work of Chi et al (2012) is related to the presented work since they use different Artificial Intelligence techniques for similar tasks, but they provide a solution just for the problem of assistive software selection based on a decision; Cortés et al (2003) propose the use of a Multi-Agent System to controlling and configuring a very specific assistive technology instance, an electric wheelchair, and the intelligent environment that surrounds it; and finally, Woodcock et al (2012) propose a decision support system developed to assist in the planning and evaluation of assistive technology, but not like our approach for end users in usage scenarios, but for assistive technology market stakeholders decision support.

Finally, to the best of our knowledge, there is no literature concerning the performance of such architectures that realize adaptive interfaces for people with special needs.

8 Conclusion

We have assessed the INREDIS architecture for performance. This software architecture tries to provide a global solution for universal access to disabled and elderly people with special needs. It automatically adapts user interfaces for both UCH devices and web services, according to users' needs and preferences, improving their accessibility. The results of the external objective lead us to conclude that the SPE methodology effectively helps the software architect to improve designs. Besides, UML and MARTE are languages that can address performance specification challenges, however, tools for specification and analysis are not mature enough. The integration of the specification and analysis tools, to carry out the whole cycle, is also a weak point. The results of the internal objective helped to improve the system response time by refactoring extensive parts of the design. The initial design, proposed by INREDIS software engineers, only met performance requirements in one out of the four main system scenarios. After the refactoring process, all system scenarios met the required response time. For the case of one hundred users, the better results were obtained in the First Interaction and Back to Top scenarios. The former was reduced from 60 seconds to 6, while the latter from 55 to less than 5 seconds.

We believe that the results gathered in this report are relevant for both, researchers and SPE practitioners. From the research viewpoint, we provide evidence that the ideas and theory behind SPE can be applied for assessing and improving a large software architecture in an industrial project. SPE practitioners, which are the target of our work, can use this industrial report as a blueprint, it can help them to develop a strategy, for assessing the performance of a software architecture, according to the needs of their projects. Furthermore, other target audience can be interested in this paper, such as accessibility experts or user experience designers.

Acknowledgements The research described in this paper arises from a Spanish research project called INREDIS (INterfaces for RElations between Environment and people with DISabilities). 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 CICYT DPI2010-20413 and GISED (partially co-financed by the Aragonese Government (Ref. T27) and the European Social Fund).
We would like to thank José Antonio Gutiérrez for his work in the experimental tests and Marta Alvargonzález, Esteban Etayo and Fausto Sainz for their help. Last but not least, the authors thank the anonymous reviewers for their valuable help to improve this work.

References

Abrams M, Phanouriou C, Batongbacal AL, Williams SM, Shuster JE (1999) UIML: An Appliance-Independent XML User Interface Language. Computer Networks 31(11-16):1695–1708

Ajmone Marsan M, Balbo G, Conte G, Donatelli S, Franceschinis G (1995) Modelling with Generalized Stochastic Petri Nets. J. Wiley

Alvargonzález M, Etayo E, Gutiérrez JA, Madrid J (2010) Arquitectura orientada a servicios para proporcionar accesibilidad. In: Proc. 5th Jornadas Científico-Ténicas en Servicios Web y SOA (JSWEB'10), *In Spanish*

Bass L, Clements P, Kazman R (2005) Software Architecture in Practice. SEI Series in Software Engineering, Addison-Wesley

Becker S, Koziolek H, Reussner R (2009) The Palladio component model for model-driven performance prediction. J Syst Softw 82(1):3–22

Bergenti F, Poggi A (2000) Improving UML Designs Using Automatic Design Pattern Detection. In: In Proc. 12th. Int. Conf. on Softw Engineering and Knowledge Engineering (SEKE'00), pp 336–343

Bernardi S, Merseguer J (2007) Performance evaluation of UML design with Stochastic Well-formed Nets. J Syst Softw 80(11):1843–1865

Brown W, Malveau R, McCormick H, Mowbray T (1998) AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. John Wiley

Card SK, Robertson GG, Mackinlay JD (1991) The information visualizer: An information workspace. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI'91), ACM, pp 181–186

Catalán E, Catalán M (2010) Performance Evaluation of the INREDIS framework. Technical report, Departament d'Enginyeria Telemàtica, Universitat Politècnica de Catalunya

Chappell D (2004) Enterprise Service Bus. O'Reilly Media, Inc.

Chi CF, Tseng LK, Jang Y (2012) Pruning a Decision Tree for Selecting Computer-Related Assistive Devices for People With Disabilities. IEEE Trans on Neural Systems and Rehabilitation Eng 20(4):564–573

Chiola G, Franceschinis G, Gaeta R, Ribaudo M (1995) GreatSPN 1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets. Perform Eval 24:47–68

Cortellessa V, Di Marco A, Inverardi P (2011) Model-Based Software Performance Analysis. Springer

Cortellessa V, Di Marco A, Trubiani C (2012) An approach for modeling and detecting software performance antipatterns based on first-order logics. Softw & Syst Modeling pp 1–42

Cortés U, Annicchiarico R, Vázquez-Salceda J, Urdiales C, Cañamero L, López M, Sànchez-Marrè M, Caltagirone C (2003) Assistive technologies for the disabled and for the new generation of senior citizens: the e-Tools architecture. AI Commun 16(3):193–207

Distefano S, Scarpa M, Puliafito A (2011) From UML to Petri Nets: The PCM-Based Methodology. IEEE Trans Softw Eng 37(1):65–79

Dugan-Jr RF, Glinert EP, Shokoufandeh A (2002) The Sisyphus Database Retrieval Software Performance Antipattern. In: Proc. 3rd Int. Workshop on Software and Performance (WOSP'02), ACM, pp 10–16

Gamma E, Helm R, Johnson R, Vlissides J (1995) Design Patterns: Elements of Reusable Object-Oriented Software. Addison–Wesley

Giménez R, Pous M, Rico-Novella F (2012) Securing an Interoperability Architecture for Home and Urban Networking: Implementation of the Security Aspects in the INREDIS Interoperability Architecture. In: Proc. 26th Int. Conf. on Advanced Information Networking and Applications Workshops(WAINA'12), IEEE Computer Society, vol 0, pp 714–719

Gómez-Martínez E, Merseguer J (2006) ArgoSPE: Model-based software performance engineering. In: Proc. 27th Int. Conf. on Applications and Theory of Petri Nets and Other Models of Concurrency (ICATPN'06), Springer-Verlag, LNCS, vol 4024, pp 401–410, http://argospe.tigris.org

Gómez-Martínez E, Merseguer J (2010) Performance Modeling and Analysis of the Universal Control Hub. In: Proc. 7th European Performance Engineering Workshop (EPEW'10), Springer, LNCS, vol 6342, pp 160–174

Gómez-Martínez E, Ilarri S, Merseguer J (2007) Performance Analysis of Mobile Agents Tracking. In: Proc. 6th Int. Workshop on Software and Performance (WOSP'07), ACM, pp 181–188

Gómez-Martínez E, Linaje M, Iglesias-Pérez A, Sánchez-Figueroa F, Preciado JC, González-Cabero R, Merseguer J (2013) Interacting with Inaccessible Smart Environments: Conceptualization and evaluated recommendation of Assistive Software *Submitted to publication*

González-Cabero R (2010) A Semantic Matching Process for Detecting and Reducing Accessibility Gaps in an Ambient Intelligence Scenario. In: Proc. 4th Int. Symposium of Ubiquitous Computing and Ambient Intelligence (UCAmI'10), IBERGACETA Publicaciones, pp 315–324

de Gooijer T, Jansen A, Koziolek H, Koziolek A (2012) An Industrial Case Study of Performance and Cost Design Space Exploration. In: Proc. 3rd ACM/SPEC Int. Conf. on Performance Engineering (ICPE'12), ACM, pp 205–216

Grand M (1998) Patterns in Java, volume 1: a catalog of reusable design patterns illustrated with UML. John Wiley & Sons, Inc.

Grand M (2001) Java Enterprise Design Patterns: Patterns in Java Volume 3. John Wiley & Sons, Inc.

Hermanns H, Herzog U, Katoen JP (2002) Process algebra for performance evaluation. Theoretical Computer Science 274(1-2):43 – 87

Huber N, Becker S, Rathfelder C, Schweflinghaus J, Reussner RH (2010) Performance Modeling in Industry: a Case Study on Storage Virtualization. In: Procs. 32nd ACM/IEEE Int. Conf. on Software Engineering (ICSE'10), ACM, pp 1–10

INREDIS (2010) Deliverable-78.2.1. Final Guide to a Generic Platform Deployment

International Standard Organization (2009) ISO 24756:2009–Framework for specifying a common access profile (CAP) of needs and capabilities of users, systems, and their environments

International Standard Organization (2011) ISO 9999:2011–Assistive products for persons with disability – Classification and terminology

Isa MA, Jawawi DNA (2011) Comparative Evaluation of Performance Assessment and Modeling Method for Software Architecture. In: Software Engineering and Computer Systems, Communications in Computer and Information Science, vol 181, Springer-Verlag, pp 764–776

Jin Y, Tang A, Han J, Liu Y (2007) Performance Evaluation and Prediction for Legacy Information Systems. In: Proc. 29th Int. Conf. on Software Engineering (ICSE'07), IEEE Computer Society, pp 540–549

Kadouche R, Abdulrazak B, Giroux S, Mokhtari M (2009) Disability Centered Approach in Smart Space Management. Int Journal of Smart Home 3(3):13–26

Kauppi T (2003) Performance analysis at the software architectural level. Technical Report 512, VTT Technical Research Centre of Finland

Kounev S (2006) Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets. IEEE Trans Softw Eng 32(7):486–502

Koziolek A, Koziolek H, Reussner R (2011) PerOpteryx: Automated Application of Tactics in Multi-Objective Software Architecture Optimization. In: Proc. 7th Int. Conf. on the Quality of Software Architectures (QoSA'11), ACM, pp 33–42

Koziolek H, Schlich B, Becker S, Hauck M (2012) Performance and Reliability Prediction for Evolving Service-Oriented Software Systems. Empir Softw Eng pp 1–45

Lazowska E, Zahorjan J, Scott G, Sevcik K (1984) Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice-Hall

Lea D (1999) Concurrent Programming in Java. Second Edition: Design Principles and Patterns, 2nd edn. Addison-Wesley Longman Publishing Co., Inc.

Levandoski JJ, Ekstrand MD, Ludwig M, Eldawy A, Mokbel MF, Riedl J (2011) RecBench: Benchmarks for Evaluating Performance of Recommender System Architectures. PVLDB 4(11):911–920

Liang S, Fodor P, Wan H, Kifer M (2009) OpenRuleBench: an analysis of the performance of rule engines. In: Proc. 18th Int. Conf. on World Wide Web, ACM, pp 601–610

Liu Y, Fekete A, Gorton I (2005) Design-Level Performance Prediction of Component-Based Applications. IEEE Trans Softw Eng 31(11):928–941

Liu Y, Gorton I, Zhu L (2007) Performance prediction of service-oriented applications based on an enterprise service bus. In: Proc. 31st Annual Int. Computer Software and Applications Conf. (COMPSAC'07), IEEE Computer Society, pp 327–334

Llinás P, Montoro G, García-Herranz M, Haya P, Alamán X (2009) Adaptive Interfaces for People with Special Needs. In: Proc. 10th Int. Work-Conf. on Artificial Neural Networks: Part II: Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living (IWANN'09), Springer-Verlag, pp 772–779

Miller RB (1968) Response time in man-computer conversational transactions. In: Proc. AFIPS Fall Joint Computer Conf. (AFIPS'68), vol 33, pp 267–277

Murua A, González I, Gómez-Martínez E (2011) Cloud-based Assistive Technology Services. In: Proc. Federated Conf. on Computer Science and Information Systems (FedCSIS'11), pp 985–989

Nielsen J (1993) Usability Engineering. Morgan Kaufmann

Object Management Group (OMG) (2011) A UML profile for Modeling and Analysis of Real Time Embedded Systems (MARTE) Version 1.1. http://www.omgmarte.org/

Petriu DC, Woodside CM (2002) Software Performance Models from System Scenarios in Use Case Maps. In: Proc. 12th Int. Conf. on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS'02), Springer-Verlag, pp 141–158

Phanouriou C (2000) UIML: A Device-Independent User Interface Markup Language. Tech. rep., Virginia Polytechnic Institute and State University

Pooley RJ, Abdullatif AAL (2010) CPASA: Continuous Performance Assessment of Software Architecture. In: Proc. 17th IEEE Int. Conf. and Workshops on the Eng of Computer-Based Systems (ECBS'10), IEEE Computer Society, pp 79–87

Pous M, Serra-Vallmitjana C, Giménez R, Torrent-Moreno M, Boix D (2012) Enhancing accessibility: Mobile to ATM case study. In: Proc. IEEE Consumer Communications and Networking Conf. (CCNC'12), IEEE Computer Society, pp 404–408

Prud'hommeaux E, Seaborne A (2006) SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/

Q-ImPrESS (2009) Q-ImPrESS Consortium: Project website. http://www.q-impress.eu

QoSA (2005-2013) Int. ACM Sigsoft Conf. on the Quality of Software Architectures, SIGSOFT

Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 14(2):131–164

Sainz F, Casacuberta J, Díaz M, Madrid J (2011) Evaluation of an Accessible Home Control and Telecare System. In: Proc. 13rd Human-Computer Interaction (INTERACT'11), LNCS, vol 6949, Springer-Verlag, pp 527–530

Schmidt DC, Stal M, Rohnert H, Buschmann F (2000) Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, 2nd edn. John Wiley & Sons, Inc.

Sereno M, Balbo G (1997) Mean Value Analysis of Stochastic Petri Nets. Perform Eval 29(1):35–62

Smith CU (1990) Performance Engineering of Software Systems. Addison–Wesley

Smith CU, Williams LG (2000) Software Performance Antipatterns. In: Proc. 2nd Int. Workshop on Software and Performance (WOSP'00), ACM, pp 127–136

Smith CU, Williams LG (2001) Software performance antipatterns; common performance problems and their solutions. In: Proc. 27th Int. Conf. Computer Measurement Group (CMG'01), pp 797–806

Smith CU, Williams LG (2002a) New software performance antipatterns: More ways to shoot yourself in the foot. In: Proc. 28th Int. Conf. Computer Measurement Group (CMG'02), pp 667–674

Smith CU, Williams LG (2002b) Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software. Addison–Wesley

Smith CU, Williams LG (2003) More new software antipatterns: Even more ways to shoot yourself in the foot. In: Proc. 29th Int. Conf. Computer Measurement Group (CMG'03), pp 717–725

Stephanidis C (2001) Adaptive Techniques for Universal Access. User Modeling and User-Adapted Interaction 11:159–179

Tribastone M, Gilmore S (2008) Automatic Translation of UML Sequence Diagrams into PEPA Models. In: Proc. 5th Int. Conf. on the Quantitative Evaluation of Systems (QEST'08), pp 205–214

UML-SPT (2005) UML Profile for Schedulabibity, Performance and Time Specification. Version 1.1, formal/05-01-02

W3C (2012) OWL 2 Web Ontology Language. http://www.w3.org/TR/owl2-overview/

Williams LG, Smith CU (2002) PASA^{SM}: A Method for the Performance Assessment of Software Architectures. In: Proc. 3rd Int. Workshop on Software and Performance (WOSP'02), ACM, pp 179–188

Woodcock A, Fielden S, Bartlett R (2012) The user testing toolset: a decision support system to aid the evaluation of assistive technology products. Work: A Journal of Prevention, Assessment and Rehabilitation 41:1381–1386

Woodside C, Petriu D, Petriu D, Shen H, Israr T, Merseguer J (2005) Performance by Unified Model Analysis (PUMA). In: Proc. 5th Int. Workshop on Software and Performance (WOSP'05), ACM, pp 1–12

Woodside CM, Neilson JE, Petriu DC, Majumdar S (1995) The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software. IEEE Trans Comput 44(1):20–34

Woodside M, Franks G, Petriu DC (2007) The Future of Software Performance Engineering. In: Future of Software Engineering (FOSE'07), IEEE Computer Society, pp 171–187

Woodside M, Petriu DC, Merseguer J, Petriu DB, Alhaj M (2013) Transformation challenges: from software models to performance models. Softw & Syst Modeling In Press

XHTML (2010) eXtensible HyperText Markup Language. http://www.xhtml.org/

Zimmermann G, Vanderheiden GC (2007) The Universal Control Hub: An Open Platform for Remote User Interfaces in the Digital Home. In: Proc. 12th Int. Conf. Human-Computer Interaction (HCI'07), Springer, LNCS, vol 4551, pp 1040–1049

A Design of the System

A.1 First Interaction Scenario

First Interaction, depicted in the UML sequence diagram (SD) in Figure 19, consists in the creation of the INREDIS initial interface, which acts as the access medium to the environment for the user. It lists all the available devices and services along with their current state and related information; and allows the user to select which one she wants to interact with. Its creation involves two processes detailed in the following sections:

- The calculation of the INREDIS parameter for that concrete user (`Perimeter Calculation`).
- The generation of the initial interface in terms of this newly calculated perimeter (`Initial Interface Generation`).

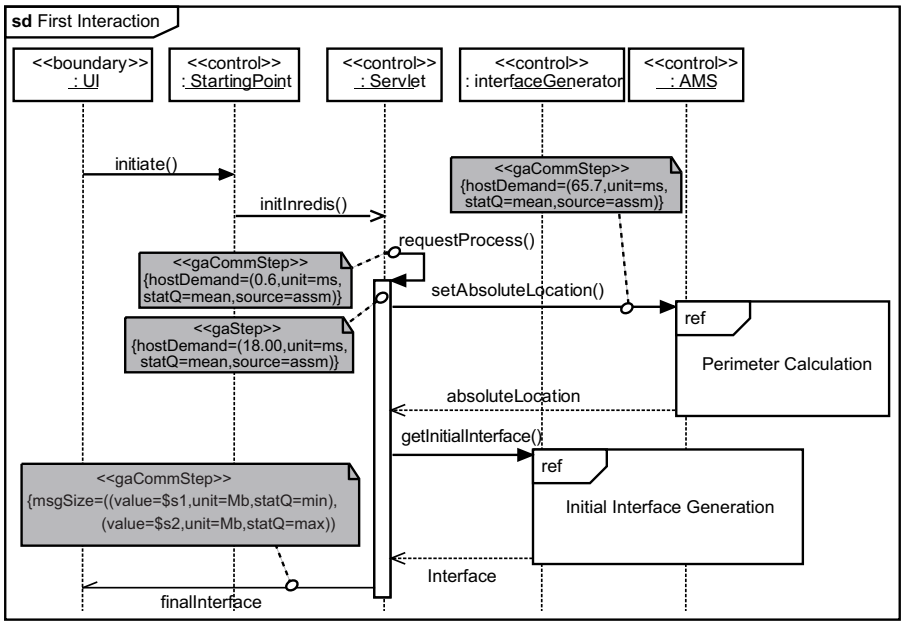

Fig. 19 UML SD representing the user's first interaction

A.1.1 Perimeter Calculation Process

The user perimeter represents the list of devices and services available to the user in a given moment. This kind of information is stored in the KB, but its calculation is made by the AMS. This module makes the necessary updates in the KB, keeping updated the situation of the user, the state of the surrounding devices, and the current state and information of the available services. Figure 20 shows the SD describing this process.

This task involves the following steps: First it must update the current location of the user. It starts with the `setAbsoluteLocation()` method that updates the information about the user in the KB. After setting the current location of the user, the AMS updates the current status of each device in the user's INREDIS perimeter. It first requests the

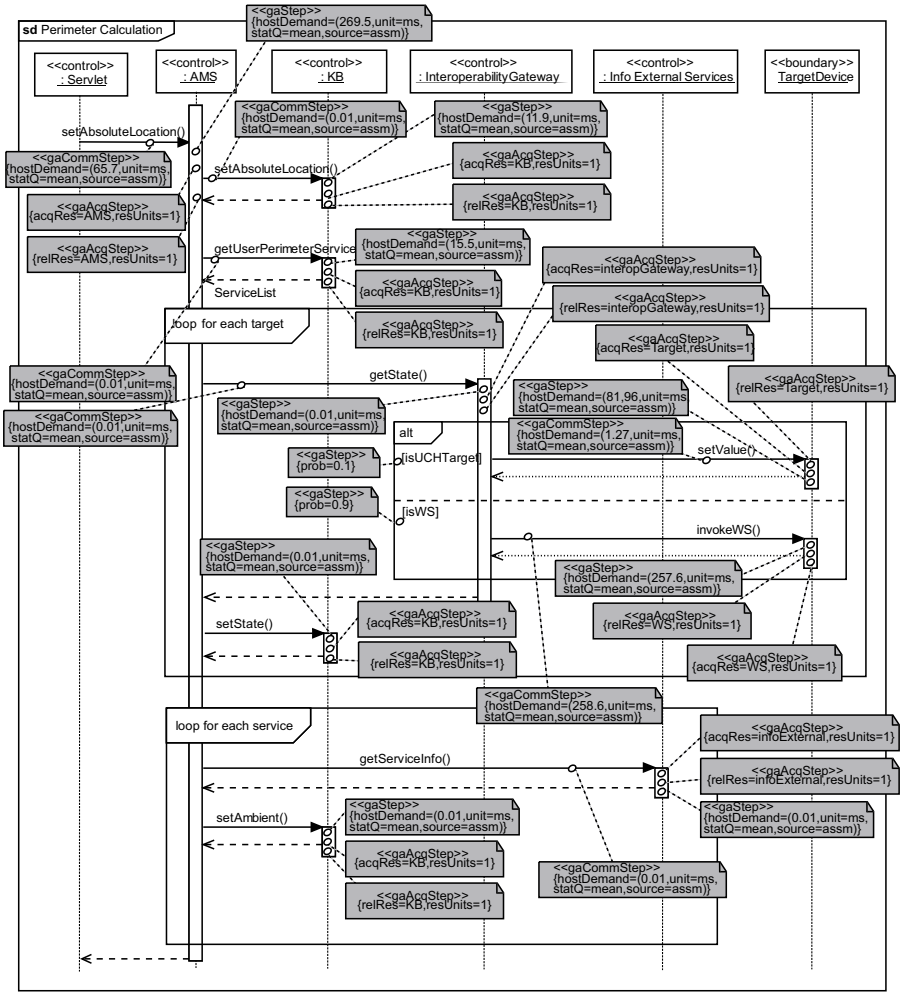

Fig. 20 UML SD showing the `Perimeter Calculation` process.

list of device and services in the user's perimeter (the `KB` `getUserPerimeterServices()` method) and for each of these devices:

- It requests to the `Interoperability Gateway` module the current state of each device (the `getState()` method). The `Interoperability Gateway` obtains this information no matter whether the device is exposed as a Web service or for UCH Target in a transparent fashion.
- It updates their current state on the `KB` accordingly (the `KB` `setState()` method).

A similar process is performed for the services in the user's INREDIS parameter:

- It requests to the services in the perimeter information about their current state (the `getServiceInfo()` method).
- It updates the current state of the `KB` accordingly (the `KB` `setState()` method).

A.1.2 Initial Interface Generation Process

Once the system has ensured that the interaction is possible, the first interface is created, see Fig 21.

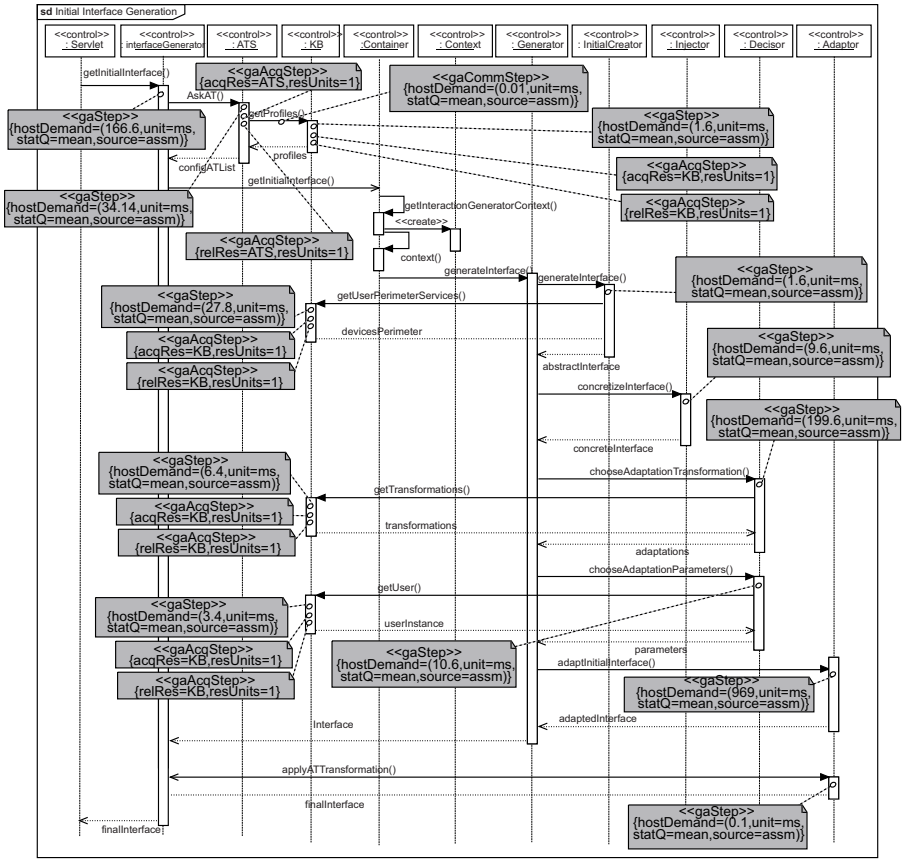

Fig. 21 UML SD modelling the Initial Interface Generation.

Before creating the initial interface the system has firstly to guarantee that the user is able to interact with its controller device. In consequence, it is necessary to determine the assistive technology that is necessary to enable such interaction. The ATS is the module responsible of such task; and also of determining how this software should be configured (method AskAT()). Using the user URI (Uniform Resource Identifier) the ATS makes queries to the KB to obtain user's profile, which is according to CAP (International Standard Organization 2009). With such profile, the ATS creates the list of the necessary assistive technology along with its configuration. The next step is the creation of the interface generator context where the variables are stored, such as the user URI, the controller device URI, navigation graph and its variables. Now the initial interface is created. As we have stated, in order to define interfaces we use a set of UIML interfaces. The case of the initial interface is no exception, but instead of having a static UIML document, in the case of the first interaction the UIML interface definition is generated, a UIML document that contains all the available devices and services, allowing the user to choose among of them. The `Generator` module,

the module that generates the UIML documents, delegates the creation of such interface to the `Initial Creator` module.

The `Initial Creator` creates what we refer as an abstract interface. It is a UIML document that still includes some context-dependent variables that have not been substituted, and a set of initialization rules that have not been performed. Such interfaces are made concrete by the Injector module (method `concretizeInterfaze()`). This module executes the initialization rules and retrieves context related values from the `IG` context.

Once the UIML interface has been made concrete, it is time to determine the process that transforms this UIML document in to an accessible XHTML user interface. For that we use a collection of XSLT transformations that address different UIML components and users special needs, which after being applied to the UIML document translate it into an XHTML document tailored to user concrete needs. The `Decisor` module of the Interface Generator in charge of determining the set of transformations (`chooseAdaptationTransformation()` method). It does so by communicating with the `KB` (`getTransformations()` method) that given a user URI and the user's controller URI determines which is the proper transformation to be applied. The selection of this transformation takes into account many orthogonal aspects, such as user's special needs, preferences and the controller interaction capabilities, see (González-Cabero 2010).

Once the concrete UIML interface has been generated and the proper XSLT transformations have been selected, there are a set of parameters that are needed to tailor the transformations. We call them the adaptation parameters, and they include the final interface language and other lower-level implementation topics. They are determined in an analogous manner to what we did for selecting the XSLT transformation.

The `Decisor` module (`chooseAdaptationParameters()` method) gathers such information asking to the `KB` for information about the user, and it also takes into account information contained in the `IG` context. Once the `IG` posses all the necessary information (i.e. the initial interface as a concrete UIML interface, the set of transformations, and the adaptation parameters) it invokes the `adaptInitialInterface()` method of the `Adaptor` module. It returns the XHTML document that represents the initial user interface.

Finally, there may be some transformations that must be applied to the initial interface XHTML document that stem from the set of assistive software provided and configured by the `ATS`. They are applied by the `Adaptor` module (the `applyATTransformation()` method). After these transformations have been applied, the final version of the interface has been created and can be delivered to the user's controller device.

A.2 Navigation Scenario

As we have already stated users interaction with a device often implies navigating through different atomic interfaces. Figure 22 shows the SD of the Navigation process.

The interaction starts with the interface requesting a navigation to the `Starting Point` that acts as a gateway between the user interface and the system. The `Interaction Enacter` is the module that handles the navigation between interfaces. It is so because the Navigation activity is considered a subclass of the Device Interaction activity, as from a user perspective, the kind of buttons that perform device interaction activities are the same as those that allow the user navigate within the complex interface. In the request `Interaction Enacter` accesses to the navigation graph of the complex interface and determines which is the next interface that should be generated. This information is stored in the context of the `IG`. Finally, a new interface generation process starts. As the context of the `IG` has been updated with the next interface to be rendered, this is the one that is rendered.

A.2.1 Interface Generation Process

INREDIS devices UIML interfaces are composed of two types of documents:

- Views, a set of UIML documents that describe structure and its interaction elements of each atomic interface. As described in (Abrams et al 1999), the use of UIML allows the abstract and platform-independent definition of user interfaces.

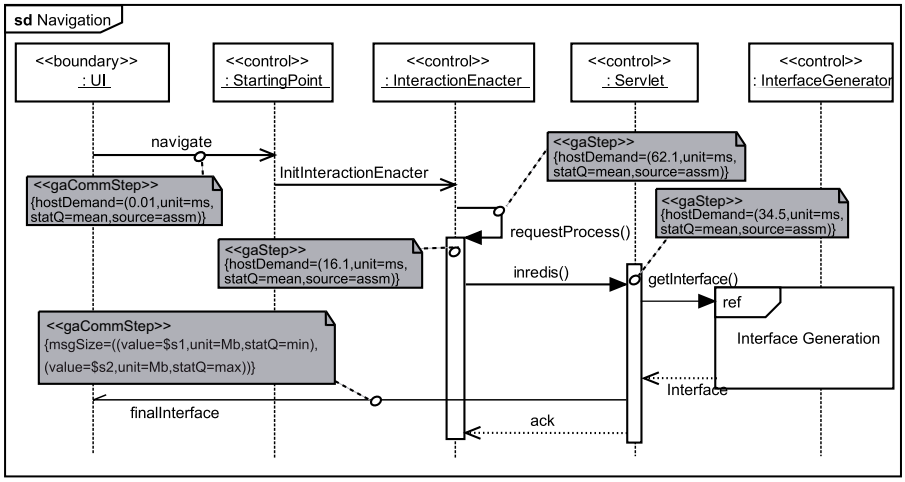

Fig. 22 UML SD modelling a simple navigation

– Navigability graph, which defines the how and on what conditions the complex interfaces navigates throw the different views. Only one view at a time is shown, we refer to it as the current view.

Generating an interface for a user means to transform the current view of an interface into an accessible XHTML document taking into account the characteristics of the user and the needed assistive technology. Most part of the process is identical to the one defined for the first interaction. The difference is that this process does not adapt the initial interface, but it transforms the current view of the device interface that the user is using at present (which like in the case of the initial interface created by the Initial Creator is a abstract UIML interface).

The first part of the diagram, the one related with the detection of accessibility gaps and the determination of the necessary assistive technology, is the same as the one defined for the first interaction with the INREDIS system. When the Generator module receives the petition of generating an interface by means of the `generateInterface()` module the first step is to determine which is the current view of the interface. This information is stored in the context of the `IG` (`getCurrentView()` method). The current view is a URL that points to the location of the abstract UIML document that should be used as the starting point of the final user interface. The `Generator` module invokes the `retrieveXMLSource()` method of the helper class `Resource Manager`, and retrieves an abstract UIML interface.

The rest of the steps of the generation of the interface are exactly as the ones described for the first interaction. Instead of using the abstract UIML interface created by the Initial Creator, they use the abstract UIML interface retrieved by the `Resource Manager` from the interface current view URL.

A.3 Device Interaction Scenario

The interactions with devices, and services are realized by the `Interaction Enacter`, see the SD in Figure 24.

This module once initialized executes the action involved in the device/service interaction. In order to do so it invokes the `executeAction()` method of the `Interoperability Gateway`, which is a class that decouples the `Interaction Enacter` from the underneath technology used to interact with the device. The `executeAction()` method may result in:

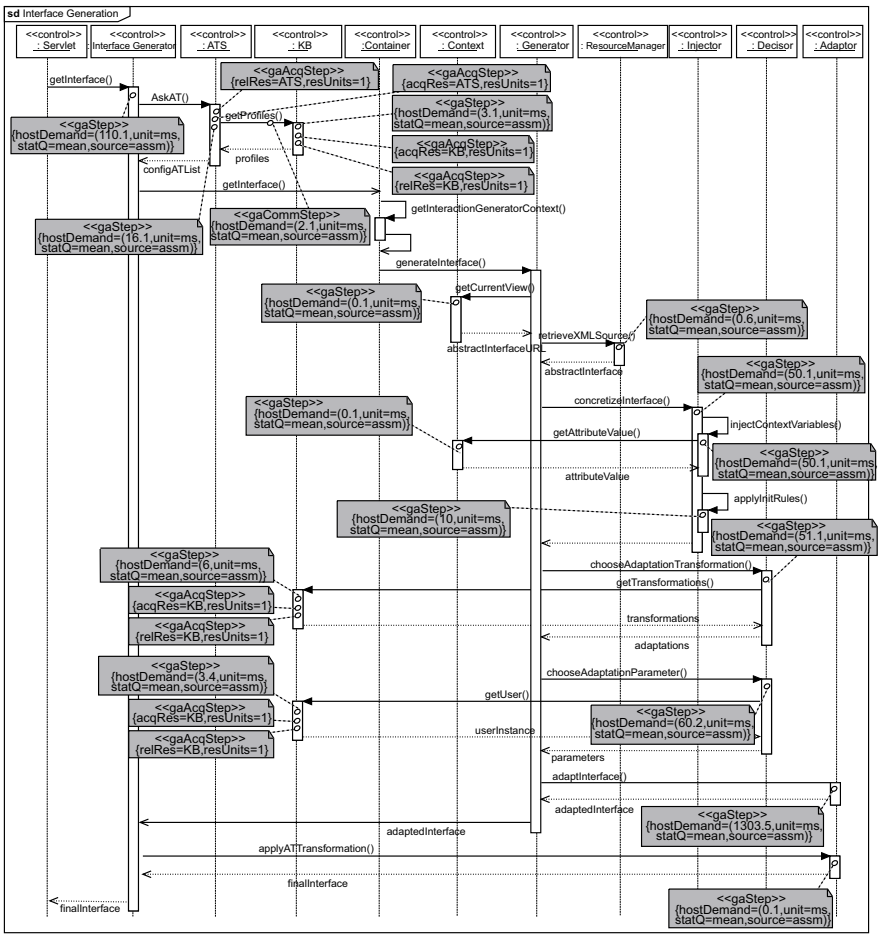

Fig. 23 UML SD modelling the Interface Generation process

- `setValue()` method invocation, in case that the device is exposed as a UCH target. The user interaction is translated into the change of one or more values of the UCH Target.
- `invokeWS()` method invocation, in case that the device is exposed as a Web service (or when there is no device and we are dealing with a Web service invocation)

The result is stored in the context of the `IG`, for later use in case of need. Once the interaction has been carried out, a new interface is generated (invoking the `Orchestrator` method `getInteface()`). This new interface is generated to make sure that it reflects the changes and latest state after the interaction with the device.

A.4 Back to Top Scenario

This process, illustrated in the SD of Figure 25, means going back to the device initial interface.

As in the case of the navigation, the `Interaction Enacter` is the module that handles the back to top process. It is so because this activity is considered a subclass of the Device

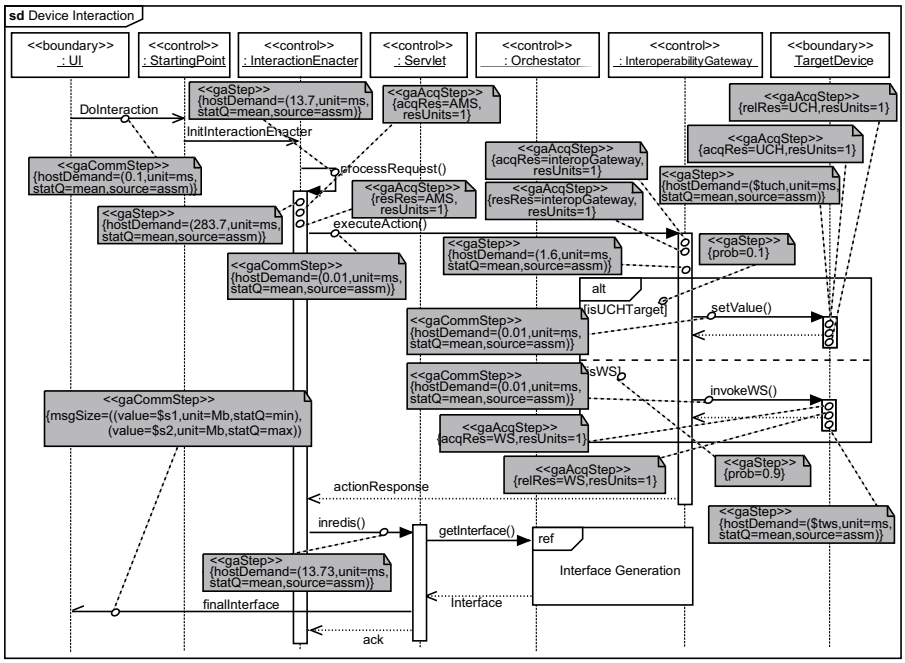

Fig. 24 UML SD representing a user's device interaction

Interaction activity, as from a user perspective, the kind of buttons that perform device interaction activities are the same as those that allow going back to the device top interface. In order to keep all the information in the `KB` up-to-date we begin updating the location of the device and we recalculate the information about the user's INREDIS perimeter .The next step is to generate top interface of the device, which is made using the `Interface Generation` process that we have already described in Section A.2.1.

A.5 Assistive Software Selection Mechanism

Assistive technology is the hardware or software that is added or incorporated within a system than increases accessibility for an individual, as defined in International Standard Organization (2011). Assistive Software (AS) is understood as a piece of software used to increase our ability to manage some kind of information in a digital device. The AS selection mechanism (ASSM) makes the environment able to automatically select the most suitable AS for a given interaction with a specific electronic target device taking advantage of the user's context (user, controller device and target device) and considering the possible discrepancies between the user and the environment, namely in the case of functional diversity.

The ASSM uses different knowledge based on ontologies to achieve this goal, so this process consists of five main activities, one of them split into six, see the UML activity diagram in Figure 26. The complete AS selection mechanism (ASSM) is described by Gómez-Martínez et al (2013). The following is a summarized description of each activity.

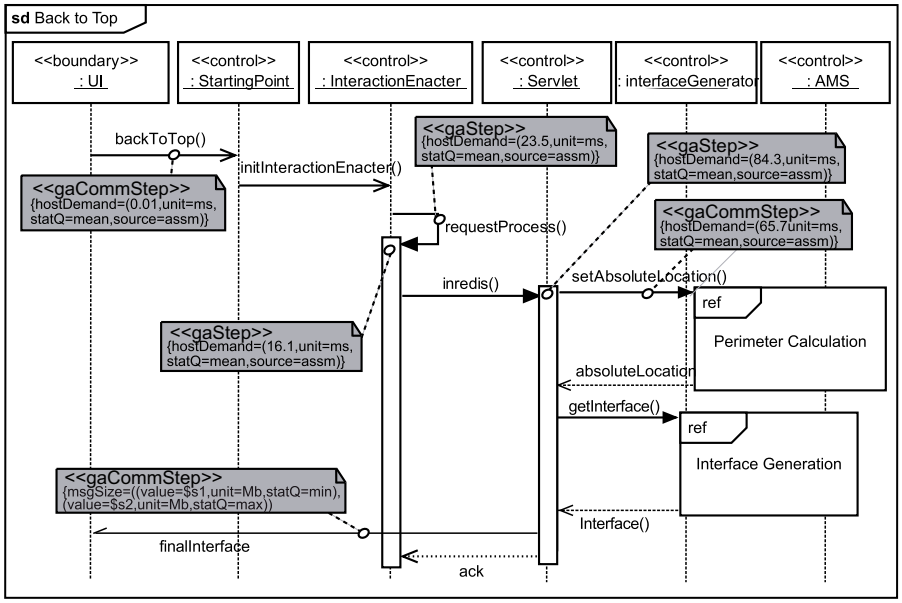

Fig. 25 UML SD representing the back-to-top process.

Detecting discrepancies The first activity detects any accessibility issues that might prevent the user from being able to use a controller. In order to detect discrepancies we use a set of specific rules stored in the KB that compare the characteristics of the interaction that the user is able to perform with those that the controller is able to emit/receive. The complete catalogue of rules is specified by González-Cabero (2010).

Checking feasibility Each discrepancy found in the previous step, is analyzed to determine whether mediation by the AS can enable the interaction. The following activities are intended to ascertain which AS is most appropriate.

Matching by History log When the user has already employed the system to interact with the same target using the same context, it is possible to retrieve the most suitable AS without further reasoning, just by querying the KB and retrieving the matching set from the AS History.

Matching by score This activity triggers the reasoning process where four subsets of concepts are simultaneously queried in the KB using parallel activities. This activity is divided into the next activities:

- *Retrieve Standard Fulfillment.* This activity performs an evaluation where the best scoring AS will be those that follow worldwide *accessibility standards* established by *recognized accessibility entities*.
- *Retrieve Privacy.* This activity checks that the AS complies with the *data protection measures* issued by security bodies. It is important to note that, according to many laws in different countries, when an *AS* complies with a *data protection act* level, it also complies some *data protection measures*. This is taken into account here via rules to assert those facts in the KB. This is the case for e.g., Federal Data Protection and Information Commission of Switzerland or Ley Orgánica de Protección de Datos in Spain.

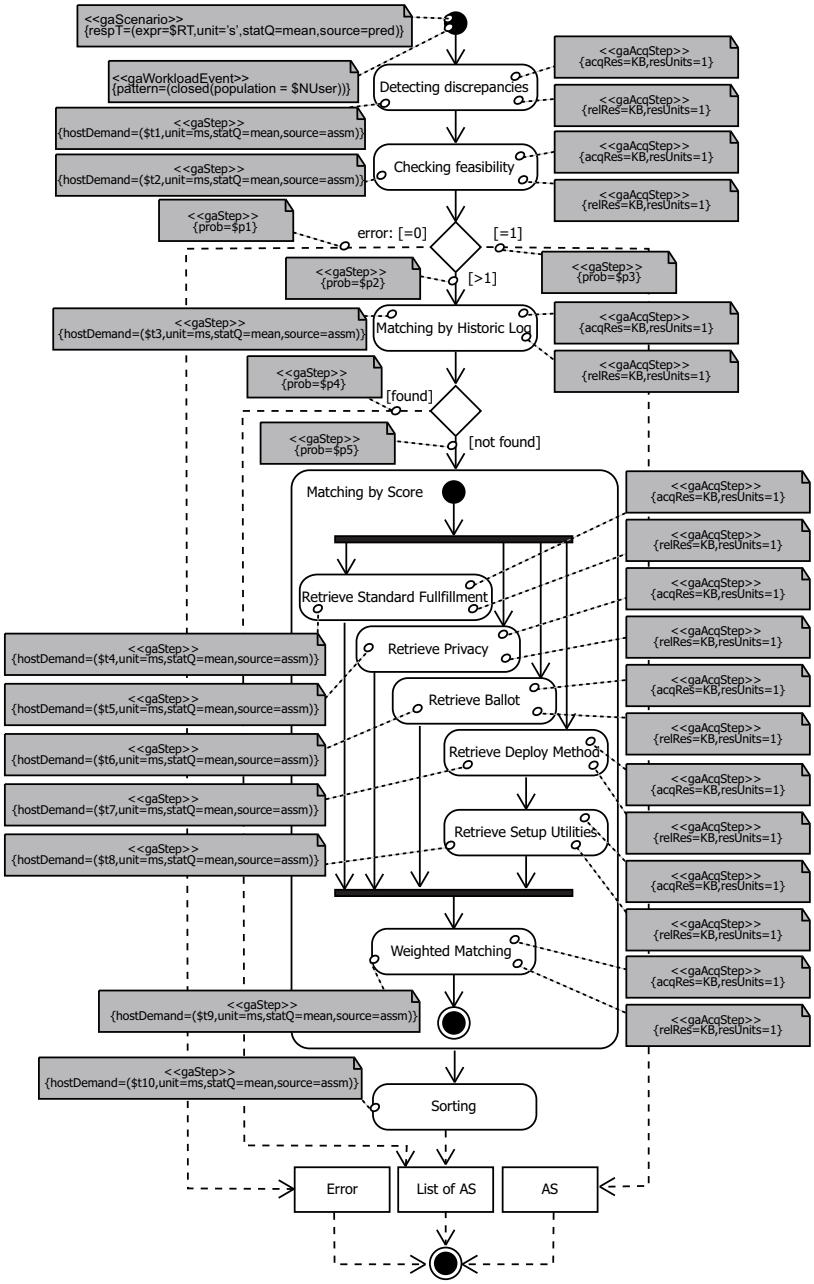

Fig. 26 UML Activity Diagram of the AS selection process

– *Retrieve Ballot.* This activity increases the score for those AS with the best user reviews. These reviews are drawn from all the system's users but they are not linked to any individual user, to maintain privacy about users' functional diversity.

- *Retrieve Deploy Method.* The scoring is the simplest, just to foster the use of *AS* deployed as *SaaS* (*Software as a Service*).
- *Retrieve Setup Utilities.* This activity needs the output of *Retrieve Deploy Method*, so it is not executed in parallel with the others. All of the concepts are scored in this activity to take into account the ease of access and use of the AS.
- *Weighted Matching.* This is the final activity of matching by score. It focuses on adapting the matching to the user's preferences and the Domain Experts' assumptions.
  The user has been previously asked to state its level of importance by means of the user profile stored in the KB. With the weighting system, all the roles involved in the selection are taken into account (i.e., domain experts, the user, and all system users at once using the reviews).

Sorting This is the final activity of the whole process. This activity orders the set of AS products/services of the weighted matching activity in descending order.

B GSPN Overview

A PN system is a tuple $\mathcal{N} = (P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{M_0})$, where P and T are the sets of *places* and *transitions*, **Pre** and **Post** are the $|P|\times|T|$ sized, natural valued, pre- and post- incidence matrices. For instance, $\mathbf{Post}[p,t] = w$ means that there is an *arc* from t to p with *multiplicity* w. When all weights are one, the PN is *ordinary*. Graphically, places and transitions are respectively represented by circles and bars, arcs are shown by arrows.

$\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$ is the *incidence matrix* of the net. For pre- and postsets we use the conventional dot notation, e.g., ${}^\bullet t = \{p \in P : \ \mathbf{Pre}[p,t] \geq 1\}$, that can be extended to sets of nodes. If $\mathcal{N}'$ is the subnet of $\mathcal{N}$, defined by $P' \subseteq P$ and $T' \subseteq T$, then $\mathbf{Pre}' = \mathbf{Pre}[P', T']$, $\mathbf{Post}' = \mathbf{Post}[P', T']$ and $\mathbf{M_0'} = \mathbf{M_0}[P']$. Subnets defined by a subset of places (transitions), with all their adjacent transitions (places), are called P- (T-) subnets.

A *marking* $\mathbf{M}$ is a $|\,P\,|$ sized, natural valued, vector and $\mathbf{M_0}$ is the *initial marking* vector. A transition is *enabled* in $\mathbf{M}$ iff $\mathbf{M} \geq \mathbf{Pre}[P,t]$; its *firing*, denoted by $\mathbf{M} \xrightarrow{t} \mathbf{M'}$, yields a new marking $\mathbf{M'} = \mathbf{M} + \mathbf{C}[P,t]$. The set of all reachable markings is denoted as $RS(\mathcal{N}, \mathbf{M_0})$. An *occurrence sequence* from $\mathbf{M}$ is a sequence of transitions $\sigma = t_1 \dots t_k \dots$ such that $\mathbf{M} \xrightarrow{t_1} \mathbf{M_1} \dots \mathbf{M_{k-1}} \xrightarrow{t_k} \dots$. Given σ such that $\mathbf{M} \xrightarrow{\sigma} \mathbf{M'}$, and denoting by σ the $|T|$ sized firing count vector of σ, then $\mathbf{M'} = \mathbf{M} + \mathbf{C} \cdot \sigma$ is known as the *state equation* of $\mathcal{N}$.

A GSPN is a tuple $\mathcal{G} = (\mathcal{N}, \mathbf{\Pi}, \mathbf{\Lambda}, \mathbf{r})$, where $\mathcal{N}$ is a PN system and the set of transitions T is partitioned in two subsets T_t and T_i of timed and immediate transitions, respectively. Timed transitions are depicted as thick white bars, immediate ones are depicted as thin black bars.

$\mathbf{\Pi}$ is a natural valued, $|T|$ sized, vector that specifies a priority level of each transition; timed transitions have zero priority, immediate transitions have priority greater than zero. A transition $t \in T$, enabled in marking $\mathbf{M}$, can fire if no transition $t' \in T : \mathbf{\Pi}[t'] > \mathbf{\Pi}[t]$ is enabled in $\mathbf{M}$.

Immediate transitions fire in zero time. Instead, the firing of a timed transition is a random variable, distributed according to a negative exponential probability distribution function with rate parameter λ (i.e., mean $\frac{1}{\lambda}$). Then $\mathbf{\Lambda}$ is the non negative real valued, $|T_t|$ sized, vector associated to the transition firing rates (accordingly, the transition firing delay is the inverse of the corresponding firing rate). The positive real valued vector $\mathbf{r}$ is $|T_i|$ sized, and specifies the weights of the immediate transitions for probabilistic conflict resolution.

C List of Performance Patterns and Antipatterns

Table 2 Performance patterns

Pattern	Description	Principle
Fast Path	Identify dominant workload functions and streamline the processing to do only what is necessary.	Centering
First Things First	Focus on the relative importance of processing tasks to ensure that the least important tasks will be the ones omitted if everything cannot be completed within the time available.	Centering
Coupling	Match the interface to objects with their most frequent uses.	Centering, Locality, Processing versus Frequency
Batching	Combine requests into batches so the overhead processing is executed once for the entire batch instead of for each individual item.	Processing versus Frequency
Alternate Routes	Spread the demand for high-usage objects spatially, that is, to different objects or locations.	Spread-the-Load
Flex Time	Spread the demand for high-usage objects temporally, that is, to different periods of time.	Spread-the-Load
Slender Cyclic Functions	Minimize the amount of work that must execute at regular intervals.	Centering

Table 3: Performance antipatterns

Antipattern	Problem	Solution
Circuitous Treasure Hunt	Occurs when an object must look in several places to find the information that it needs.	Refactor the design to provide alternative access paths that do not require a Circuitous Treasure Hunt (or to reduce the cost of each "look")
Blob or "god" Class	Occurs when a single class performs most of the work of the system, relegating others classes to minor, supporting roles.	Refactor the design to distribute intelligence uniformly over the application's top-level classes, and to keep related data and behaviour together.
Concurrent Processing Systems	Occurs when processing cannot make use of available processors.	Restructure software of change scheduling algorithms to enable concurrent execution.
"Pipe and Filter" Architectures	Occurs when the slowest filter in a "pipe and filter" architecture causes the system to have unacceptable throughput.	Break large filters into more stages and combine very small ones to reduce overhead.
Extensive Processing	Occurs when extensive processing in general impedes overall response time.	Move extensive processing so that it does not impede high traffic or more important work.

Continued on next page

Table 3 – continued from previous page

Antipattern	Problem	Solution
Empty Semi Trucks	Occurs when an excessive number of request is required to perform a task. It may be due to inefficient use of available bandwidth, and inefficient interface, or both.	The Batching performance pattern combines items into messages to make better use of available bandwidth.
Roundtripping	Occurs when many fields in a user interface must be retrieved from a remote system.	Buffer all the calls together and make them in one trip. The Facade design pattern and the distributed command bean accomplish this buffering.
Tower of Babel	Occurs when processes excessively convert, parse, and translate internal data into a common exchange format such as XML.	The Fast Path performance pattern identifies paths that should be streamlined. Minimize the conversion, parsing, and translation on those paths.
One-Lane Bridge	Occurs at a point in execution where only one, or a few, processes may continue to execute concurrently. Other processes are delayed while they wait for their turn.	To alleviate the congestion, use the Shared Resources Principle to minimize conflicts.
Excessive Dynamic Allocation	Occurs by the overhead required when an application unnecessarily creates and destroys large number of objects during its execution.	1) "Recycle" objects (via an object "pool") rather than creating new ones each time they needed. 2) Use the Flyweight-pattern to eliminate the need to create new objects.
The Ramp	Occurs when processing time increases as the system is used.	Select algorithms or data structures based on maximum size or use algorithms that adapt to the size.
Traffic Jam	Occurs when one problem causes a backlog of jobs that produces wide variability in response time which persists long after the problem has disappeared.	Begin by eliminating the original cause of the backlog. If this not possible, provide sufficient processing power to handle the worst-case load.
More is Less	Occurs when a system spends more time "thrashing" than accomplishing real work because there are too many processes relative to available resources.	Quantify the thresholds where thrashing occurs (using models or measurements) and determine if the architecture can meet its performance goals while staying below the thresholds.
Sisyphus Database Retrieval	Occurs when performing repeated queries that need only a subset of the results.	Use advanced search techniques that only return the needed subset.

Continued on next page

Table 3 – continued from previous page

Antipattern	Problem	Solution
Falling Dominoes	Occurs when one failure causes performance failures in other components.	Make sure that broken pieces are isolated until they are repaired.
Unnecessary Processing	Occurs when processing is not needed or not needed at that time.	Delete the extra processing steps, reorder steps to detect unnecessary steps earlier, or restructure to delegate those steps to a background task.

D GSPN Models of the System

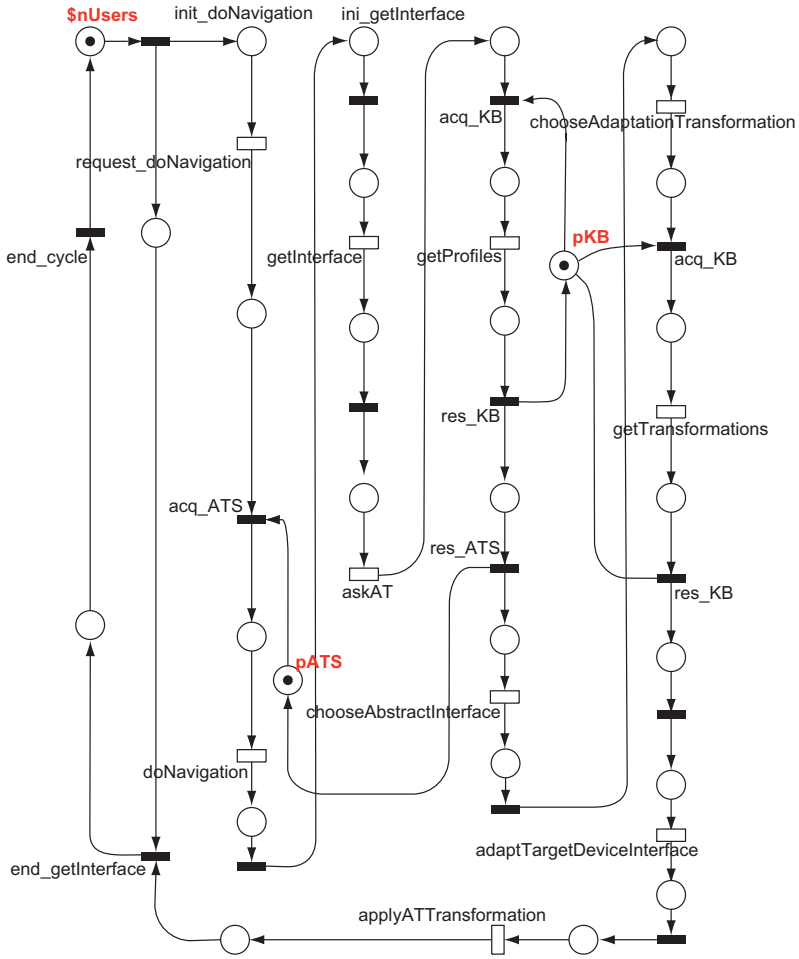

Fig. 27 GSPN representing `Navigation` scenario.

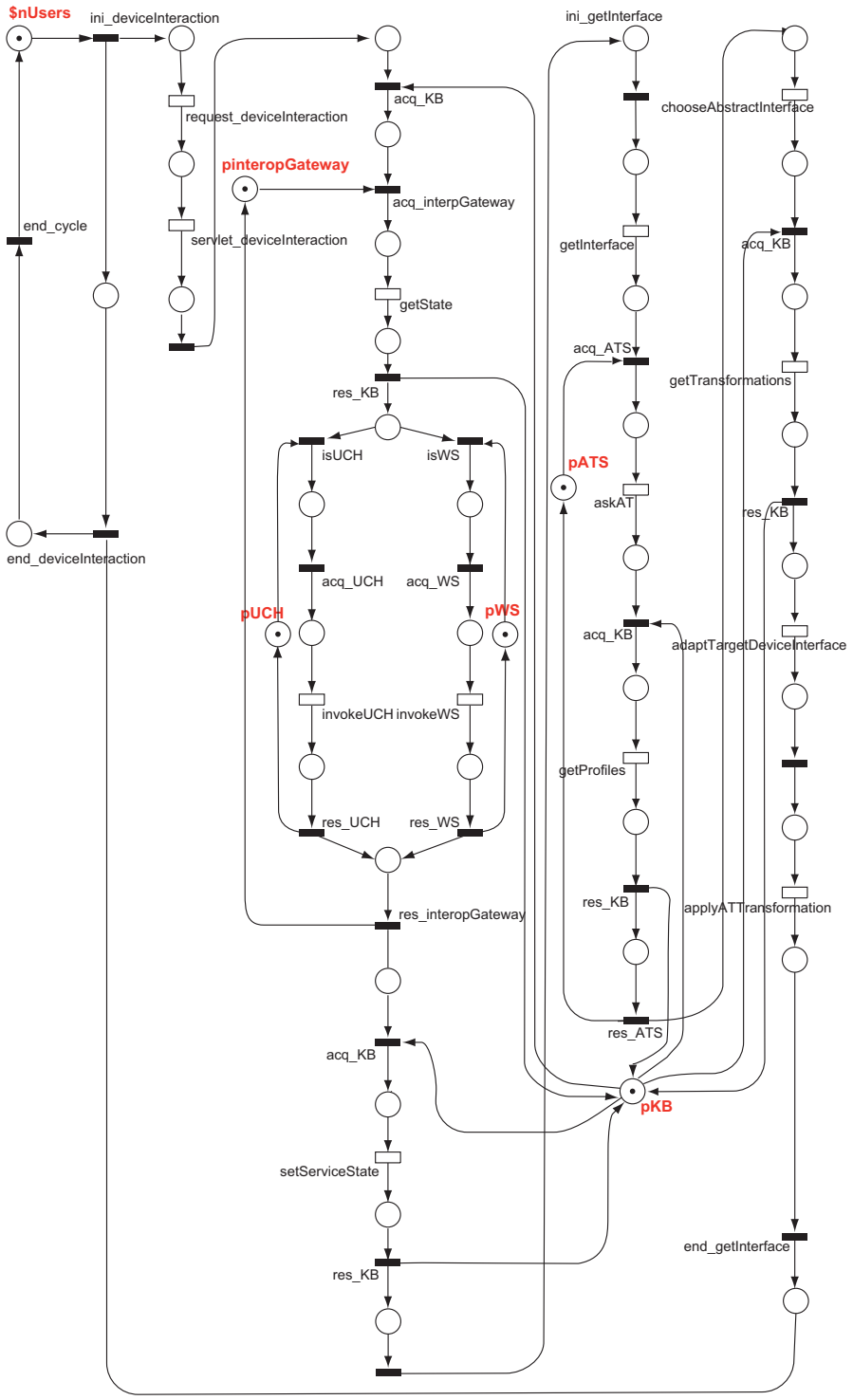

Fig. 28 GSPN representing `Device Interaction` scenario.

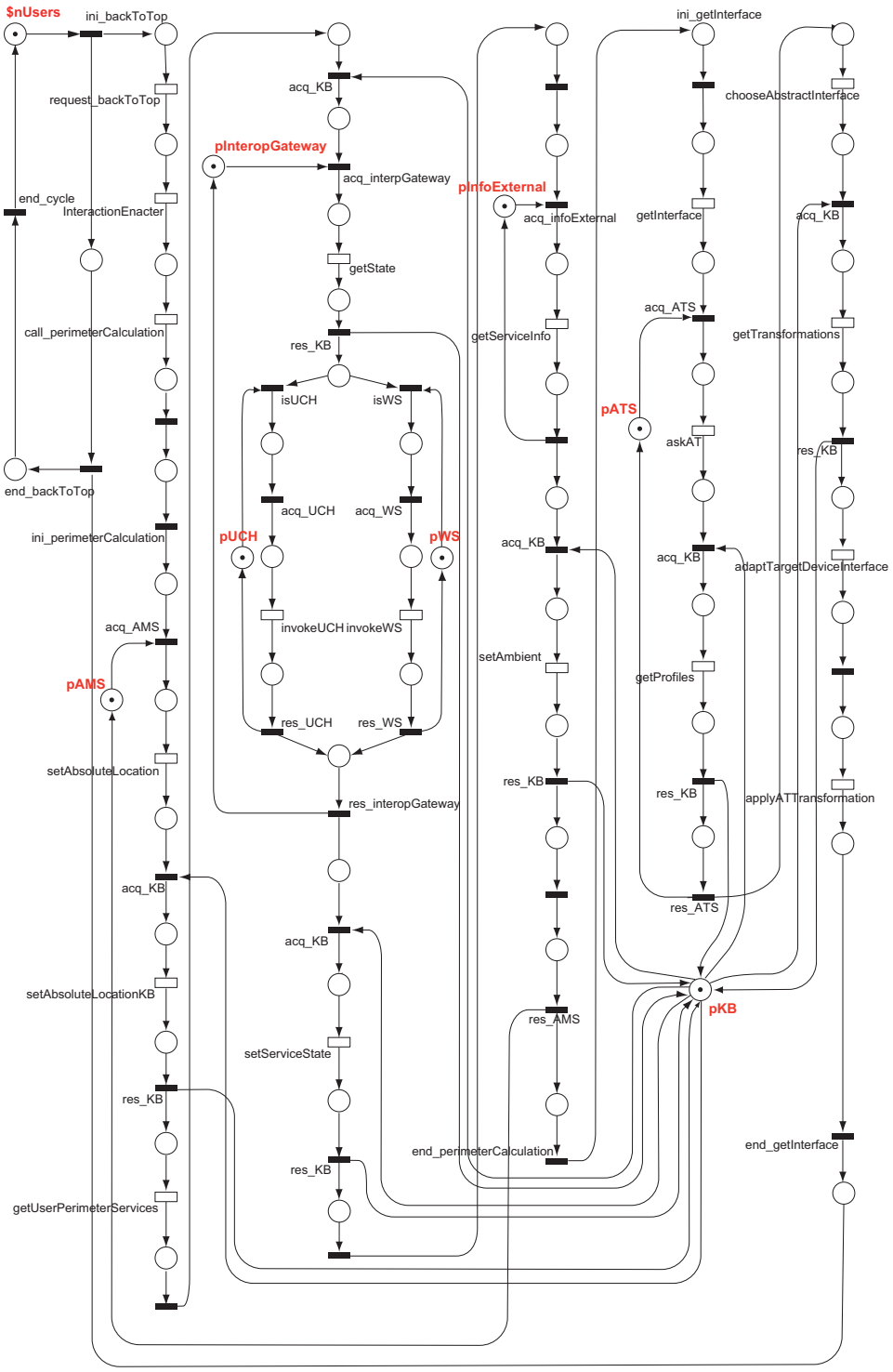

Fig. 29 GSPN representing `Back To Top` scenario.