Toward an Integrated Quality Evaluation of Web Applications with DEVS

Verónica Bogado, Silvio Gonnet, and Horacio Leone

INGAR, Universidad Tecnológica Nacional, CONICET
Avellaneda 3657, 3000 Santa Fe, Argentina
{vbogado,sgonnet,hleone}@santafe-conicet.gov.ar

Abstract. The increasing dynamic and complexity of Web systems turns quality evaluation at any stage of the development into a key issue for the project success in software development areas or organizations. This paper presents a novel approach to evaluate Web applications (WebApps) from their architectures, also considering their functionalities. Discrete EVents System Specification (DEVS) is proposed for behavior and structure analysis based on a set of quality criteria that serve as guidelines for development and evolution of these Web systems. Three quality attributes are considered in this version of the approach: performance, reliability, and availability, but the main advantages are potential scalability and adaptability that respond to the features of these systems.

Keywords: WebApp Quality Evaluation, Software Architecture, DEVS.

1 Introduction

Nowadays, Web has become an indispensable instrument to the most organizations. Thus, developing Web applications (WebApps) becomes a challenge for the software companies. WebApps are software systems with inherent multifaceted functionality and they exhibit sophisticated behavior and structure, which must answer to the demands of high quality and must have the ability to grow and evolve over the time [1]. Visible features at runtime, i.e. during the operation of these systems, such as performance, availability, or security are quality attributes that must to be considered and analyzed during the development and improvement of WebApps. However, software companies still develop this kind of software in an ad-hoc way, increasing problems related to quality. With the aim to assist developers in the design of WebApps, systematic and quantifiable approaches towards high-quality systems are required [2].

Despite the magnitude and impact of WebApps, there are no yet standard quality assessment tools that give support to the software architects/developers during the development of WebApps. However, Software Architecture (SA) is a mean to predict the success or failure of a project providing different views of the system to analysis quality aspects and there are some advances in this issue not only for Web system but for software in general ([3], [4]).

Q.Z. Sheng and J. Kjeldskov (Eds.): ICWE 2013 Workshops, LNCS 8295, pp. 79–91, 2013.

In this work, we integrate structural, functional, and quality aspects in the same analysis to improve quality of Web systems. We propose the construction of an executable model based on Discrete EVent System Specification (DEVS, [5]), where its execution will provide useful information to analyze behavior and quality of complex and sophisticated Web systems applied to several domains. This model is obtained from a Use Case Map (UCM, [6]) model that represents the architecture of the WebApp to be evaluated and the main scenarios. UCMs help architects to understand emergent behavior of complex and dynamic systems. DEVS formalism and its underlying framework for modeling and simulation (M&S) allow us to build an adaptable simulation environment, being an executable model of the WebApp under evaluation. The simulation elements are specified following the principles of modularity and hierarchy. This high level of abstraction enables us to represent suitably the concepts of the SA of WebApps and complex paths to represent complex dynamic scenarios (functionality) under common operation conditions on the Web. Therefore, DEVS allows developers to study potential scenarios and operational profiles of the system by mean of simulation, being a cheap way to prevent problems.

Parallel DEVS is particularly used for the specification of the simulation elements due to it provides a set of features commonly found in WebApps. An Atomic Parallel DEVS has a bag of ports to receive values at the same time with the possible multiple occurrences of its element. It allows all imminent components to be activated simultaneously and to send their outputs to other components having a confluent transition function to solve collisions between internal and external events. Coupled Parallel DEVS specifies components and how they are connected [5].

The rest of paper is organized as follows. Section 2 discusses related work providing an overview. Section 3 summarizes the approach based on DEVS to evaluate quality of Web systems. Section 4 describes a concrete WebApp, simulation outputs, and simple examples of how to use the information to improve the software designs after the evaluation. Section 5 presents conclusions and future work.

2 Related Work: An Overview

Quality evaluation employing architectural designs is becoming a key step in the software development, but it is a growing trend to focus on WebApps because of the demand of this kind of systems. In this context, there are general approaches based on scenarios and qualitative analysis [3]. More formal proposals employ Markov Decision Process with analytical resolution for a quantitative analysis of reliability, performance, or security ([7], [8]). Queueing Theory is useful to measure performance [9], while Petri Nets have been applied to evaluate different quality attributes in an analytical form [10]. These formal techniques have some critical limitations related to the modeling even more in WebApps development [11]. For example, complex systems or software components are reduced to simple states losing important information of the software elements.

Recently, empirical techniques have acquired importance in view that simulation and prototyping provide abstraction level to model real software system. In this way, prototyping has been successfully applied to evaluate SA but with high cost of implementing the prototype [12]. Palladio Component Model (PCM) is a metamodel

to predict quality attributes. It has been focused on performance and reliability prediction transforming the concrete SAs into dynamic models ([13], [14]). Nevertheless, it is still necessary dynamic tools that can be as adaptable as the metamodel in terms of expressiveness of the target model, which results in a Queueing Network or Markov Chain losing features in the transformation.Finally, approaches based on the visual notation UCM allow architects to model complex and dynamic system, where scenarios and structures may change at runtime to study performance using Layered Queueing Network [15].

Although there are several approaches for SA evaluation, there is still need of a formal evaluation model that considers not only SA (structure) and isolated quality attributes but includes functionality in the same analysis. Furthermore, the elements of the evaluation model must represent the software elements and their sophistication. In this way, UCM provides elements to represent graphically system structures (simple and complex SA elements) and scenarios (functionality). On the other hand, DEVS is a formalism to specify complex and dynamic systems with the purpose of simulating them under possible scenarios and conditions. DEVS was successfully applied in other domains ([16], [17]). DEVS provides two basic types of models, atomic and coupled, and extensions like Parallel DEVS, which is used in this work. Parallel DEVS has some advantages to simulate Web background allowing concurrence and distributed components.

3 Quality Evaluation with DEVS

In this section, we summarize the elements considering for the simulation environment with DEVS formalism. Due to the need of considering functional requirements in the SA to have a complete view of the system, we propose UCM models as inputs to evaluate the system quality (Fig. 1(a)). UCM is an informal notation that captures functional requirements in terms of causal scenarios representing behavioral aspects at high level design [6]. UCM does not replace Unified Modeling Language (UML), but complements it being a bridge between requirements and design. Due to it is a graphical notation, it is useful to understand emergent behavior of complex and dynamic systems ([18], [19]). In the last few years, UCM has gained an important place to describe WebApps because it provides tools to treat sophisticated components and complex scenarios and operational uses.

UCM notation provides basic and architectural elements (Fig. 1 (a)): responsibilities, paths and components. A path represents a scenario of the system and it is executed after an external stimulus has happened. A responsibility point is a place where the state of a system is affected or interrogated. Stimulus is a start point in an execution where a pointer starts in this place and then is moved along the path. Thus, the pointer enters and leaves components touching responsibility points inside. Finally, end position is reached and the execution is finished by emitting a response. UCM does not prescribe the number of threads associated with a path. So, concurrency can be modeled with AND-Fork/Join elements generating several concurrent subscenarios. Alternative subscenarios can be represented using OR-Fork/Join ([6],[18],[20]).

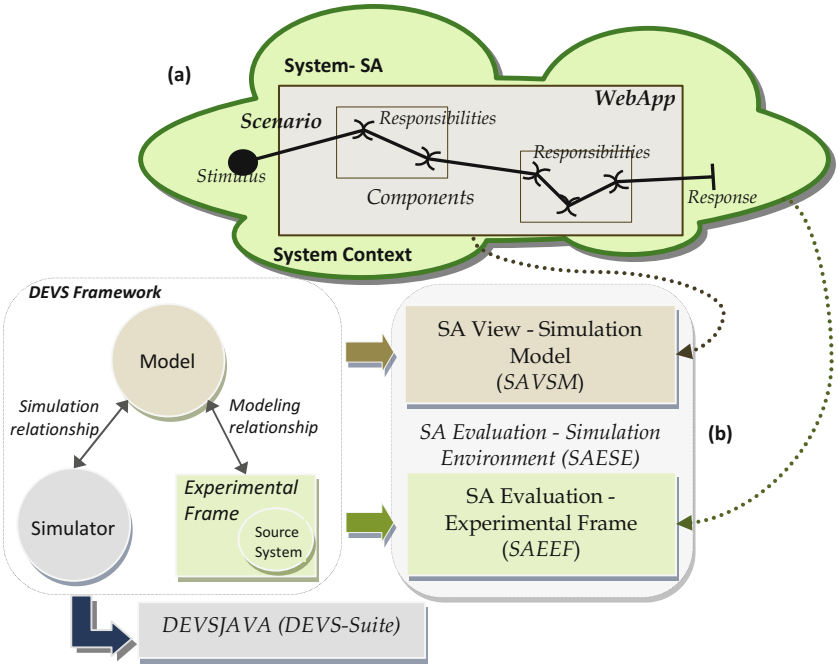

Fig. 1. DEVS Framework for the quality evaluation of WebApps

To complement this flexible but informal notation, we have proposed a DEVS-based simulation environment with the purpose of evaluating software quality. This approach adds semantic to the simulation elements that represent software elements building dynamic models. DEVS formalism specifies the architecture and behavior of Web systems without losing important features of the responsibilities, simple and composite components (hierarchical structures), among others. Furthermore, it integrates several perspectives and quality attributes in the same analysis.

A conceptual framework for modeling and simulation (*DEVS Framework*, Fig. 1(b)) gives support to the formalism. It is composed by three entities [5]: i) *Model*: system specification that defines the structure and behavior to generate data comparable to data from the real world, ii) *Simulator*: computation system that executes the instruction of the model giving life to it, iii) *Experimental Frame*: conditions under which the system is observed for the experimentation and validation of the model. *Modeling* and *Simulation* relationships link these parts. Therefore, the proposed simulation environment (*SAESE*, Fig. 1(b)) has two main conceptual parts: simulation model for SA evaluation (*SAVSM*, Fig. 1(b)) and experimental frame for SA evaluation (*SAEEF*, Fig. 1(b)). *Simulator* is taken from *DEVSJAVA* (*DEVS-Suite* 2.0[1], Fig. 1(b)) implementation encapsulating this part from the other two. Keeping separate theses entities gives some benefits such as the same model can be executed by different simulators or several experiments can be changed for studying different situations.

[1] `http://www.acims.arizona.edu/SOFTWARE/software.shtml`

DEVS is an adaptable and scalable approach to tackle the SA evaluation problem in the context of Web systems. It provides elements to build simple and complex dynamic systems keeping the semantic of the Web software elements. Elements of the core of UCM notation are specified as models of DEVS formalism.

Fig. 2 summarizes DEVS-based evaluation process, which has four main stages: specify UCM for the WebApp, generate the DEVS-based simulation environment, configure and execute the simulation, and analyses the results to make decisions, where each one involves a set of activities detailed in the diagram. First, architects (or developers) have to specify the SA using UCM notation (*SA UCM*), which can be obtained from the user requirements, if it is an early evaluation, or from the implementation, if it is a late evaluation (activity 1). So functional requirements define the scenarios of the Web system and non-functional aspects provide information to build the SA. This input model is translated into a DEVS hierarchy (activity 2), where each element is translated into simulation elements specified in DEVS, major details can be found in a previous work [21]. So, simple elements of UCM such as OR-Fork/Join, AND-Fork/Join, stimulus (start point) are specified as Atomic Parallel models, being the basis to build more complex structures. In this way, responsibility, simple/composite components, system (SA view), and the whole UCM are specified as Coupled Parallel models.

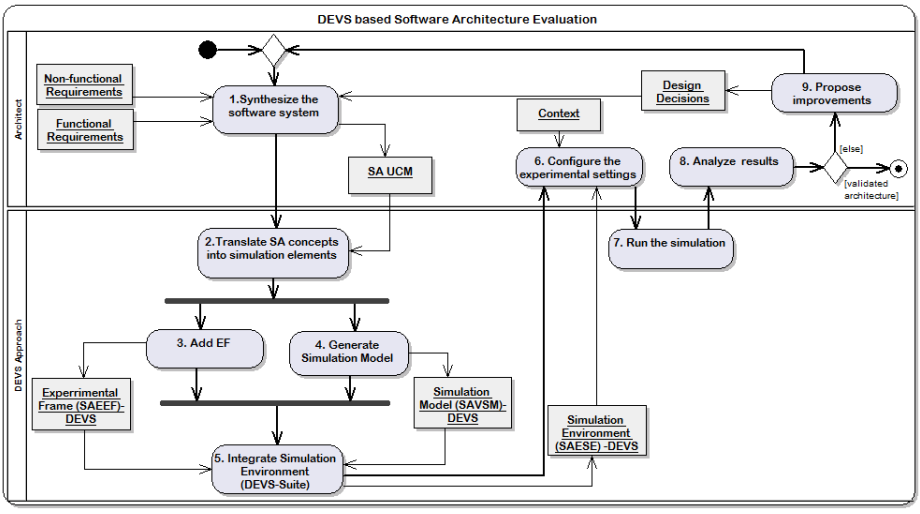

Fig. 2. DEVS-based quality evaluation process

Simulation elements generate the simulation model while configure the EF (activities 3 and 4, Fig. 2). Simulation model represents the WebApp and it is a Coupled Parallel DEVS called *SAVSM*. The background under the Web system operates is represented by the EF, which is also a coupled model (*SAEEF*). These two parts build the whole simulation environment for the WebApp by implementing them in *DEVS-Suite* (activity 5, Fig. 2). Architect/developer has to configure the parameters to run the simulation (evaluation) using information to adjust the probability

distributions (activity 6). After the evaluation (activity 7, Fig. 2), a set of measures and quality indicators (explained in the next subsection) are obtained. This information allows architects/developers to analyze the system (activity 8) and can make design decisions if the quality requirements were not achieved (activity 9).

As we have mentioned, simulation elements build a hierarchical structure of DEVS models that represents the whole simulation environment for the UCM of the WebApp. It has two main parts that work together to simulate the dynamic of the system. In this way, the simulation model (*SAVSM*, Fig. 3) represents the WebApp architecture and the main scenarios. An input port is defined (*erip*) to receive requests from external sources (*SAEG* from *SAEEF*, Fig. 3) and a set of output ports to emit the responses, processed requests (*esop*, Fig. 3), and measures (described in Table 1 in the following section) taken from internal components (*rtaop*, *rdtop*, *rrtop*, and *rfailop*, Fig. 3) that are sent to the EF (*SAEEF*, Fig. 3).

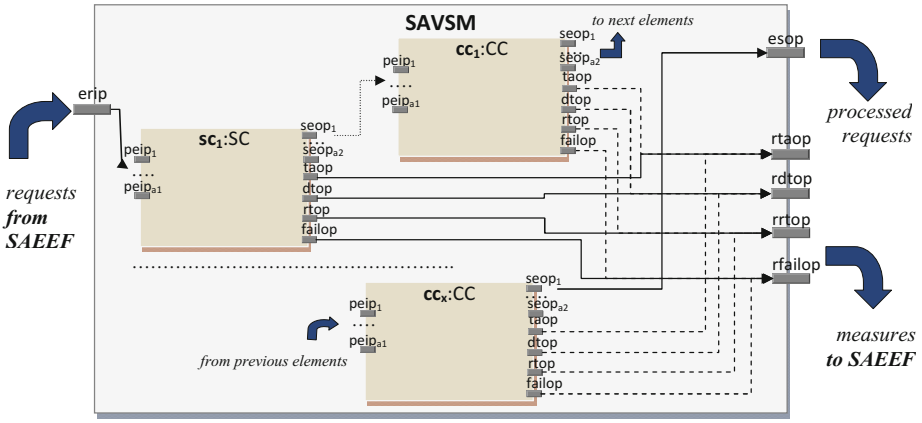

Fig. 3. *SAVSM:* Simulation Model

Responsibilities are the smallest software unit (*CPXRES*) that together scenarios elements such OR/AND define couplings to build complex DEVS that represent simple components (*SC*), which can be coupled to obtain composite components (*CC*) and the view of the architecture (*SAVSM*). *CPXRES*, *SC,* and *CC* define sets of input/output ports for the causal flows ($peip_1 \ldots peip_{a1}$, $seop_1 \ldots seop_{a2}$, Fig. 3) and a set of output ports to propagate taken measures (*taop*, *dtop*, *rtop*, and *failop*, Fig. 3).

The experimental frame, *SAEEF*, has a simulation element that represents the stimulus of the system (*SAEG*, Fig. 4), which "gives life" to the WebApp. It emits a request as output using the port *rop*. Other component defines the start and end of the quality evaluation (*SAEA*, Fig. 4) sending signals using the port *ssop*. Finally, there is a set of elements called "stat" that are responsible for the calculation of quality indicators, one per quality attribute considered in the evaluation. So, we have defined a DEVS model for the followings quality attributes visible at run time (Fig. 4): performance (*SAEPS*), availability (*SAEAS*), and reliability (*SAERS*). These elements have a set of input ports that receive measures or messages from other simulation

elements, which are used to compute quality indicators (described in the next section). The input ports allow *SAEEF* to propagate the measures sent from the *SAVSM* (*rtaip*, *procreqip*, *rfailip*, *rdowntimeip*, and *rrecovtimeip*, Fig. 4) and the signal to start the evaluation (*sip*, Fig. 4). The output interface emits the system indicators (explained in Table 2, next section) for being used in future simulation of more complex environment in the Web context.

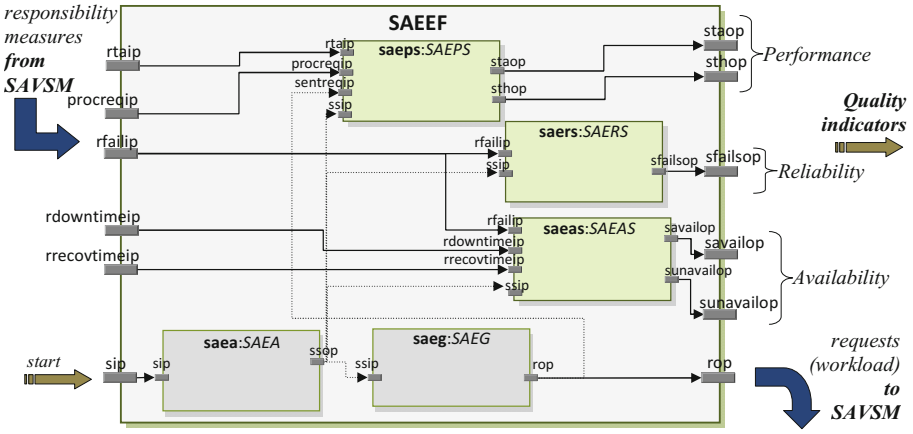

Fig. 4. *SAEEF:* Experimental Frame

3.1 Software Quality Attributes and Related Measures

In this work, metrics related to performance, availability, and reliability are considered to take direct and indirect measures [22]. Responsibilities are the main providers of quantitative information so a set of basic metrics are defined (Table 1).

Table 1. Measures taken from each responsibility

ID	Metric	Description
rta	Turnaround time per request	Time that a responsibility requires to answer to a request.
fdt	Downtime per failure	Time that responsibility is "failed" due to a failure.
frt	Recovery time per failure	Time that a responsibility needs to return to a normal operation after a failure has occurred.
fn	Number of Failures	Failures that have happened in a responsibility.

Measures are propagated through the hierarchy from simple components to the top, *SAVSM* (e.g. *taop*, *dtop*, *rtop*, and *failop* respectively, Fig. 3), sending theses values to the *SAEEF*. Specific simulation elements have a set of specific domain operations that return a more complex value (Table 2). These measures can be quality indicators of the system, being outputs of the simulation environment (Table 2), or responsibilities (e.g. turnaround time or downtime per responsibility, omitted here).

Table 2. Quality Indicators: Simulation Outputs

Metric	Description	Attribute	Sim. Element
Average turnaround time of the system	Average time that the system requires to answer to a request.	Performance	*SAEPS (port staop)*
Average throughput of the system	Average number of request served per time unit in the system.	Performance	*SAEPS (port sthop)*
Total unavailable time of the system	Total time that the system is offline (downtimes and recovery times).	Availability	*SAEAS (port sunavailop)*
Total available time of the system	Total time that the system has been online.	Availability	*SAEAS (port savailop)*
Total number of failures of the system	Total amount of failures occurred in the system.	Reliability	*SAERS (port sfailsop)*

4 Evaluation, Results, and Design Decisions

Digital electoral register (DER) is a WebApp that keeps information about all people registered to vote in a particular city including information of the polling place locations. Furthermore, it has a specific module for geographical information employing maps, which requires to access to an external server that returns a coordinate with the latitude and longitude of a given location.

This WebApp has two kinds of users: elector and admin. They imply different scenarios and operational uses of the system. The first one is a role defined for anonymous people that ask for the polling locations. Each query generates a request to the system and this workload grows in the last month before the Election Day. This WebApp manages information about 60000 persons in condition to vote. On the other hand, admin users are related to other scenario that implies the refinement of the electoral register and the update of the information (electors and schools). In this process of refinement, the coordinates of each elector address and polling place is updated by submitting a query to the external server *Gmaps*. In the last scenario, a higher load is generated 45 days before the closure of the electoral register (a month before the Election Day).

Following the process presented in Section 3, we first specifies the Web system using UCM notation considering the SA and the main scenarios, then we translate this models into DEVS models, adjust parameters and run the simulation obtaining indicators to analyze the system behavior and validate the quality requirements.

SA was rebuilt from the current implementation applying reverse engineering. We have looked at the structure, functions, and operation of the WebApp to obtain a technology independent architecture to study several scenarios and validate the simulation environment. In this paper, we analyze the server and its behavior under several conditions of uses due to it is the main part of this system and it must to be evolved to manage not only information of people in a city but in a province or state. Consequently, a traditional SA for WebApps is obtained, Client-Server pattern structured in three levels with three main parts related to: presentation, business, and data. The view of the server (*DERSystem-Server*) in particular has a composite component (*WebServer*) and a simple component (*DBServer*), where the composite

component has two simple components inside, *BusinessProcessor* and *GMapsLocator*. The first one executes the functionality related to the domain and the other one interacts with *GMaps*. All these components embody fundamental units at runtime for this WebApp. In conclusion, this architecture involves three levels of complexity, where a client requires services to the server specified in Fig. 4, and this server becomes a client to another external server (*GMaps*).

Each element of the architecture takes part according to the scenario. The first scenario is focused on the elector query involving user requests (electors) as stimuli and a set of responsibilities (Fig. 5). So, the scenario starts when a client connects to the server (*r1*), enters the required data and an alphanumeric key that appears as an image (*r2*). User data is validated (*r3*) while Captcha test is executed (*r4*) producing concurrent subscenarios. Finally, the information required in the query is retrieved from the DB, emitting a response with the information or an error message. Here the performance is a critical issue due to the big workload generated by the potential voters near to the Election Day (previous month to this crucial day).

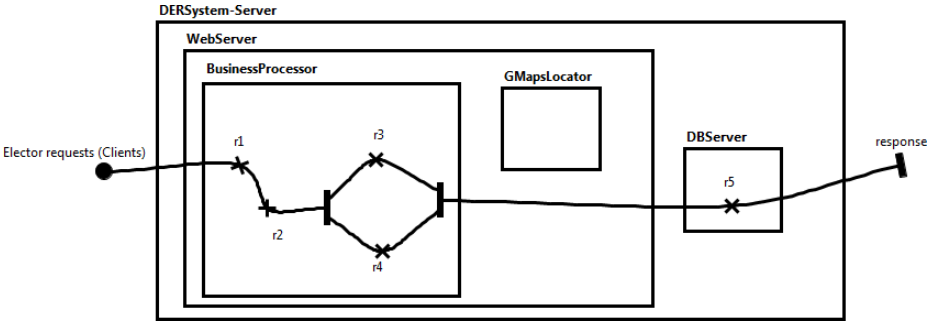

Fig. 5. UCM of the Server: Scenario 1- Elector Query

The second scenario defines a more complex path involving several alternatives in the causal flow (Fig. 6). The stimuli are given by the user requests (admin profile) through the client, which connects to the server (*r1*). Once it is connected, the user can chose between three options that produce three possible subscenarios: elector registration (*r2*), schools registration -polling places- (*r3*), and geographical coordinates updating of registered electors (*r4*). The first two options cause retrieving the neighborhood associated to the given address (*r5*). The alternative paths are joined to require the calculation of the coordinate (*r6*), where this responsibility involves a query to *GMaps*. Lastly, once the coordinates are available, the information is updated in the DB (*r7*) finishing the causal flow.

Regarding operation, this scenario has minor load due to the number of administrators even so during the critical period. Here availability takes an important place.

We describe two quality attribute scenarios as examples. The first requirement is related to system performance specifying that the turnaround time has to be less than 2000 ms under normal operation of *DERSystem-Server* to respond to the user requests (electors). An availability scenario specifies that the unavailable time has to be less than 60 min per month under normal operation of *DERSystem-Server*.

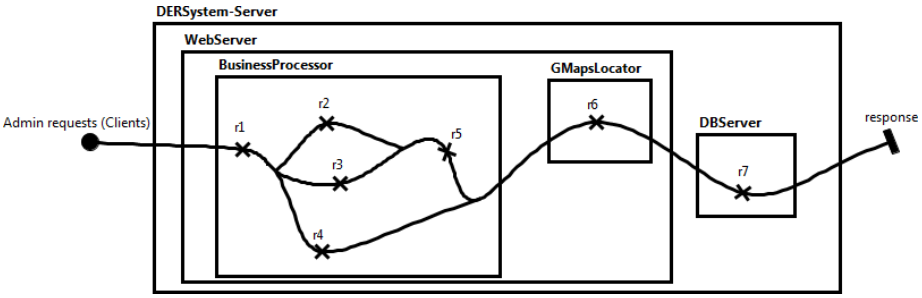

Fig. 6. UCM of the Server: Scenario 2- Elector Register Refinement (admin)

Fig. 7 illustrates the simulation environment for the second scenario and SA of the WebApp in *DEVS-Suite*. Atomic models (grey nodes) compose coupled models that represent responsibilities, simple and composite components, and view. Parameters are configured using information from reports of failures and turnaround times.

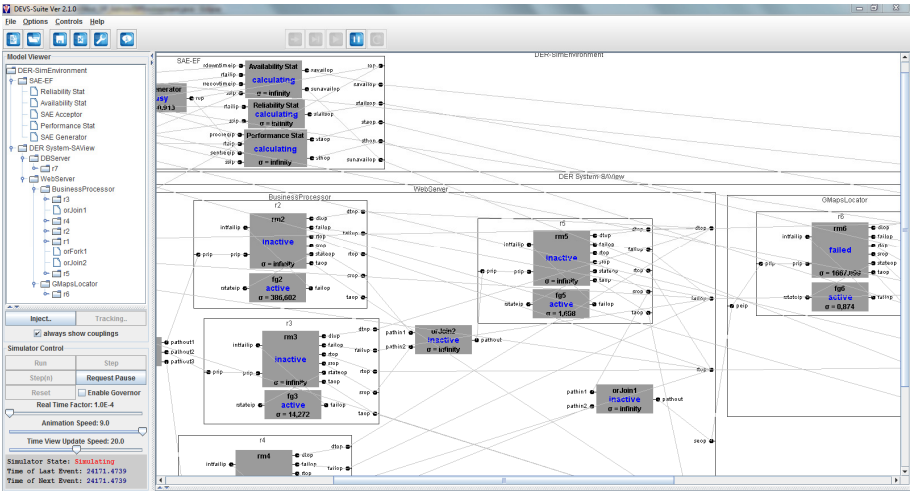

Fig. 7. DEVS Simulation Environment: Scenario 2- Elector Register Refinement (admin)

A late SA evaluation takes place to understand the WebApp (DER System) in an improvement process. In both scenarios, the simulation was run under conditions of each critical period, where the request arrivals are defined by Poisson distribution. Table 3 summarizes these conditions in columns two (period under evaluation) and three (workload). The following columns present the system results (average for ten simulations runs): number of requests sent to WebApp, turnaround time, unavailable time, and number of failures. The real time employed by the simulator to execute each scenario under the defined conditions is detailed in the last column.

Table 3. Summary of results after simulations runs

	Operation time	*Requests*	*Turnaround time*	*Unavailable time*	*Failures*	*Simulation time*
Scenario 1	30 days	11514	**1029.63 ms**	**42.96 min**	25.6	21 min
Scenario 2	45 days	3033	**2643.11 ms**	**106.69 min**	47.8	13.5 min

a. PC: Intel Core i7 860 2.80Ghz, RAM 4GB

Highlighted data can be directly used to validate the quality requirements specified previously. The performance requirement is achieved by the first scenario but not by the second one. Regarding availability requirement, the unavailable time has to be less than 60 min per month, so the first scenario is fulfilled but the second one not (should be less than 90 min in 45 days). These examples and other information that describes each responsibility (omitted here, [21]) can be used to make design decisions. It could help architects to find strengths and weaknesses in the complex structure and behavior of WebApps.

Additional information taken from the implementation was used to validate the outputs of the simulation environment detailed in Table 3. Now the simulation environment can be used to evaluate the WebApp under new operational conditions.

5 Conclusions and Future Work

The main contribution of this work consists in evaluating Web systems employing DEVS. A behavioral and structural analysis driven by quality attributes can be done using a high level design specified with UCM notation. This integrated analysis considers the main perspectives of the system: SA (structure), functionality (scenarios), and quality (measures). A DEVS environment is briefly described: simulation model represents the Web system and the experimental frame implements quality goals and the environment that interacts with the system. Despite of this work presented general designs, DEVS approach can be adjusted to specific Web technologies (Web services, presentation design, frameworks as .Net, JAVA EE).

This approach provides several advantages to evaluate quality of WebApps that could improve this kind of complex and dynamic systems. Firstly, high level abstraction, a modular and hierarchical way to build domain-specific simulation elements. Secondly, model decoupled from the simulator. Thirdly, the experimental frame decoupled from the other two parts specifies the conditions under which the system is observed, and the operational formulation of quality goals including one element for each attribute that will be analyzed. Finally, homogenous representation of the simulation elements allows the interchange of them building a simulation environment based on interface and encapsulation of the internal mechanisms.

Regarding the case study, we have implemented the proposed simulation environment for a WebApp, which has a traditional Web architecture client-server. Two relevant scenarios were validated, analyzing quality requirements and obtaining information to make design decisions that improve the current design or to make changes that address new requirements.

Several issues remain open. Other quality attributes that are visible at runtime will be studied, adding components to the experimental frame. In this way, new quality aspects could be considered in the analysis to make design decisions that improve the quality of WebApps. Moreover, it is interesting to include particularities of Web systems in the model, behavioral or structural patterns, which are domain-specific to resolve the inherent complexity of these systems for better results after the simulation.

Acknowledgements. Authors thank the financial support from Universidad Tecnológica Nacional, CONICET, and Agencia Nacional de Promoción Científica y Tecnológica (PAE-PICT 02315).

References

1. Pressman, R.: What a Tangled Web We Weave. IEEE Software 18(1), 18–21 (2001)
2. Casteleyn, S., Florian, D., Dolog, P., Matera, M.: Engineering Web Applications. Springer (2009)
3. Clements, P., Kazman, R., Klein, M.: Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley (2002)
4. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley (2012)
5. Zeigler, B., Praehofer, H., Kim, T.: Theory of Modeling and Simulation–Integrating Discrete Event and Continuous Complex Dynamic Systems. Academic Press (2000)
6. Amyot, D.: Introduction to the User Requirement Notation: Learning by Example. Computer Networks 42(3), 285–301 (2003)
7. Wang, W., Pan, D., Chen, M.H.: Architecture-based Software Reliability Modeling. Journal of Systems and Software 79(1), 132–146 (2006)
8. Sharma, V., Trivedi, K.: Quantifying Software Performance, Reliability and Security: An architecture-based Approach. Journal of Systems and Software 80(4), 493–509 (2007)
9. Spitznagel, B., Garlan, D.: Architecture-based Performance Analysis. In: Proc. 1998 Conference on Software Engineering and Knowledge Engineering, pp. 146–151 (1998)
10. Fukuzawa, K., Saeki, M.: Evaluating Software Architecture by Coloured Petri Nets. In: Proc. 14th International Conference on Software Engineering and Knowledge Engineering, pp. 263–270 (2002)
11. Singh, L.K., Tripathi, A.K., Vinod, G.: Software Reliability Early Prediction in Architectural Design Phase: Overview and Limitations. Journal of Software Engineering and Applications 4(3), 181–186 (2011)
12. Christensen, H., Hansen, K.: An Empirical Investigation of Architectural Prototyping. Journal of Systems and Software 83(1), 133–142 (2010)
13. Becker, S., Koziolek, H., Reussner, R.: The Palladio Component Model for Model-driven Performance Prediction. Journal of Systems and Software 82(1), 3–22 (2009)
14. Brosch, F., Koziolek, H., Buhnova, B., Reussner, R.: Architecture-based reliability prediction with the Palladio Component Model. IEEE Transactions on Software Engineering (2011)
15. Petriu, D.B., Woodside, M.: Software Performance Models from System Scenarios in Use Case Maps. In: Field, T., Harrison, P.G., Bradley, J., Harder, U. (eds.) TOOLS 2002. LNCS, vol. 2324, pp. 141–158. Springer, Heidelberg (2002)
16. Byon, E., Pérez, E., Ding, Y., Ntaimo, L.: Simulation of Wind Farm Maintenance Operations using DEVS. Simulation 87(12), 1091–1115 (2011)

17. Ferayorni, A.E., Sarjoughian, H.S.: Domain driven Simulation Modeling for Software Design. In: Proc. of the 2007 Summer Computer Simulation Conference (SCSC 2007), pp. 297–304 (2007)
18. Buhr, R.: Use Case Maps as Architectural Entities for Complex Systems. IEEE Transactions on Software Engineering 24(12), 1131–1155 (1998)
19. Amyot, D., Mussbacher, G.: User Requirements Notation: The First Ten Years The Next Ten Years. Journal of Software 6(5), 747–768 (2011)
20. de Bruin, H., van Vliet, H.: Quality-driven Software Architecture Composition. The Journal of Systems and Software 66(3), 269–284 (2003)
21. Bogado, V., Gonnet, S., Leone, H.: A Discrete Event Simulation Model for the Analysis of Software Quality Attributes. CLEI Electronic Journal 14(3), Paper 3 (2011)
22. ISO/IEC 9126-1: Software Engineering – Product Quality – Part 1: Quality Model, Number 1 (2001)