A Research Agenda Concerning Dependability of Web-Based Systems

Position paper

Arndt von Staa [1]

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro, PUC-Rio
arndt@inf.puc-rio.br

Abstract. *Some research issues relative to dependability of web-based systems are identified. A model-based approach to solve these issues is proposed. This approach requires a software engineering meta-environment that operates on a data dictionary. The main macro-functionalities of the meta-environment are described. This approach has been experimentally validated with respect to conventional systems and must now be extended for web-based systems.*

1. Problem statement

As systems migrate from "all components are known" to a service oriented architecture assuring dependability becomes much harder. For one, it is not known a priori who will be the clients of the offered services of the components [Robinson, 2008]. Furthermore, if clients may choose among service providers, it is not always clear if the choice is adequate or not. I will call "component" any closed piece of software that interacts over the web with other components. Components are usually unable to provide meaningful results just by themselves; they must interact with other components to provide such results. The collection of interacting components forms a (web-based) system. Components may be composed by several artifacts, such as models, documentation and modules (code).

Obviously any web-based system must be reliable, more so due to being much more automated than traditional (non web-based) systems [Reason, 1990]. Failures in web-based systems may pose great threats or inflict great losses.

But how can we acquire confidence about the reliability of such systems? There are at least following avenues that should be explored:

- **defect prevention** – defective components may add a very high cost of detection and removal, both at development time as well as at maintenance time [Westland, 2002]. Thus we should construct all components in accordance to clear, complete and published requirements and interface specifications. A nice analogy, although not from the software domain, are DIN (*Deutsche Industrie Norm*) standards. What must be specified considering software components? Are we already capable of specifying with similar rigor as in established engineering fields?

[1] This paper has been supported by Programa INC&T - Projeto: Instituto Brasileiro de Pesquisa em Ciência da Web. CNPq grant 557.128/2009-9 and FAPERJ grant E-26/170028/2008

- **interface contracts** – establish clear and complete interface specifications. Interface data in open systems tend to be XML files. Although a very effective way of identifying names and values, it usually lacks sufficient detail. For example, what units are used with regard to transferred values? What standards are applied? What are the numerical error ranges? Hence, interface contracts must be far more detailed than they are nowadays. They must and also be evolvable [Robinson, 2008] since seldom components are static over time. However, evolution must not compromise already existing and possibly unknown clients. What should interface standards look like? Are anthologies an effective and run-time efficient way of specifying interface contracts? How could we verify whether the contracts are being broken or not? [Carvalho et al, 2006] Are technologies such as agent based systems a good means for implementing controlled interfaces?
- **self-monitoring** – develop sufficient redundancy that allows verifying whether the components are operating in the expected way. Humans err and combinations of errors may lead to disasters [Reason, 1990; Brown and Patterson, 2001]. How can we handle failure exceptions in such a way as not to compromise overall reliability? [Guerra et al, 2003] Another approach could be to develop recovery oriented components [Magalhães et al, 2009]. Such components may fail, possibly due to human error, however they must recover very fast and must not lose the context information upon which they were operating at the moment of failure detection.
- **quality control** – It is known that non-deterministic systems pose a major obstacle considering quality control. How should web-based component and system quality be controlled? How much confidence can we obtain using conventional methods?
- **evolution** – systems that are relevant to users tend to be long lived. Hence they evolve due to changing requirements and due to adapting to new platforms. How can this be achieved without disrupting interactions among collaborating components? How can all interdependent representations be co-evolved assuring continuous coherence among all documents?

2. Proposal

Figure 1 presents an outline of the proposed solution. The main idea is to develop a meta-environment that is capable of editing, transforming and exploring a data dictionary. The contents of this data dictionary should be exportable as an ontology.

Inputs are specifications of a component and of its interface. These documents should be written in a computationally-lay-reader friendly way and, hence, usually do not necessarily convey adequate information to enable the development. For this reason it is necessary to insert a step that transforms these specifications into adequate software specifications. While performing this transformation the data dictionary is populated with a network of text fragments that correspond to the specifications. To increase interoperability it might be necessary to interact with data of other components.

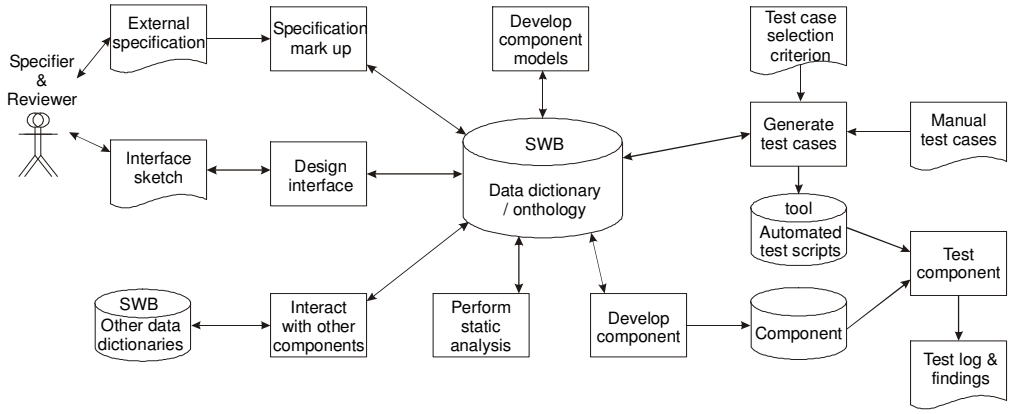

Figure 1. Outline of the proposed solution

The data dictionary is in fact a network of text and data fragments. Given a description of a representation language and a focal element contained in the dictionary, the network is explored in order to renderize representations for human reading or to be used by some tool. The system that operates on the data dictionary is a meta-system than can be instantiated to support a variety of representation language descriptions [Staa, 1993].

Once the specifications have been accepted, models must be developed. Several possible modeling languages may be used, for example UML or OOHDML. Since many new representation languages will be tried, it is necessary that the tools are implemented as meta-tools, allowing adaptation to the needs of these new languages.

The models must be checked for structural (syntactical) correctness, and should also be checked for design anti-patterns (*bad smells*)[Macía et al, 2010]. This is performed by static analysis applied to the models. While performing these checks, it might be necessary to complement or modify specifications, as well as several representations of the models. The tool must be capable of supporting these actions without destroying already done work.

Once some of the models have been accepted, it should be possible to develop test suites that will steer test driven development of the component. Due to the complexity of the test cases it may be necessary to generate them based on transformations of the models.

In parallel with the development of the test suites, artifacts of the component may be developed. These artifacts should also be verified using static analysis tools. Once passed the static analysis, these artifacts will be tested using automatic testing tools. This should assure that each artifact has a sufficiently high quality to be integrated with others to form the desired component.

To be able to aid the maintenance of components, it is of utmost importance that the tools adequately support the co-evolution of the several representations that might be extracted from the data dictionary. It is also important that data dictionaries may be split into several dictionaries, one for each component. When constructing a system the several dictionaries could be explored to assure composability correctness. Finally, the

data dictionaries and the tools that manipulate them constitute the environment necessary to properly maintain each component.

Together with the development of the tools, programming techniques and design patterns should be developed, aiming at controlling the correctness of component interaction, as well as self-monitoring its operations. Since these program elements must stay in the deployed code, it is necessary that they do not impose a too heavy burden on required computational resources, especially considering execution time.

Many of the proposed ideas have already been tried and have shown that they are both feasible and effective considering traditional systems [Staa, 1993]. However, due to several restrictions, not all of the features could be tried in industrial environments. Furthermore, they must be adapted to adequately and efficiently support the development and maintenance of web-based components and systems.

References

Brown, A.B.; Patterson, D.A.; (2001) "To Err is Human"; Proc. of the First Workshop on Evaluating and Architecting System Dependability, Goteborg, Sweden, 2001; URL: http://roc.cs.berkeley.edu/papers/easy01.pdf

Carvalho, G.R.; Brandão, A.A.F.; Paes, R.B.; Lucena, C.J.P.; (2006) Interaction Laws Verification Using Knowledge-based Reasoning; AOIS.06@AAMAS workshop; pp 33-40

Guerra, P.A. de C.; Rubira, C.; Romanovsky, A. and Lemos, R. de; (2003) "Integrating COTS Software Components into Dependable Software Architectures", Proc. of the 6-th IEEE Int'l Symp. on Object-Oriented Real-Time Distributed Computing (ISORC'03). pp 139-142.

Macía, I.B.; Garcia, A.; Staa, A.v.; (2010) "Applying and Detecting Code Smells in Aspect-Oriented Software Systems"; CBSoft/SBES 2010; accepted for publication september 2010.

Magalhães, J.A.P.; Staa, A.v.; Lucena, C.J.P.; "Evaluating the Recovery Oriented Approach through the Systematic Development of Real Complex Applications"; Software Practice and Experience 39(3); New York: Wiley Periodicals; 2009; pp 315-330

Reason, J. (1990) Human Error; Cambridge University Press; 1990

Robinson, I. (2008) Consumer-Driven Contracts: A Service Evolution Pattern; in The ThoughtWorks Anthology; Raleigh, North Carolina: The Pragmatic Bookshelf; pp 93-112

Staa, A.v.; Ambiente de Engenharia de Software Assistido por Computador - TALISMAN; versão 4.3; Rio de Janeiro, RJ: Staa Informática Ltda.; 1993 (in Portuguese).

Westland, J.C.; (2002) "The Cost of Errors in Software Development: Evidence from Industry"; The Journal of Systems and Software 62; New York, NY: Elsevier; pp 1-9.