Chapter 2
Verification of Non-Functional Requirements by Abstract Interpretation

Agostino Cortesi
Università Ca' Foscari, Italy

Francesco Logozzo
Microsoft Research, USA

ABSTRACT

This chapter investigates a formal approach to the verification of non-functional software requirements that are crucial in Service-oriented Systems, like portability, time and space efficiency, and dependability/robustness. The key-idea is the notion of observable, i.e., an abstraction of the concrete semantics when focusing on a behavioral property of interest. By applying an abstract interpretation-based static analysis of the source program, and by a suitable choice of abstract domains, it is possible to design formal and effective tools for non-functional requirements validation.

INTRODUCTION

Effective and efficient management of customer and user requirements is one of the most crucial, but unfortunately also least understood issues (Karlsson, 1997), in particular for Service Oriented Systems. In Service Oriented Architectures the non-functional aspects of services and connections should be defined separately from their functional aspects because different applications use the services and connections in different non-functional contexts. The separation between functional and non-functional aspects improves the reusability of services and connections. It also enables the two different aspects to evolve independently, and improves the ease of understanding application architectures. This contributes to increase the maintainability of applications (Wada, Suzuki, & Oba, 2006 and O'Brien, Merson, & Bass, 2007).

Problems in the non-functional requirements are typically not recognized until late in the development process, where negative impacts are substantial and cost for correction has grown large. Even worse, problems in the requirements may

DOI: 10.4018/978-1-61350-432-1.ch002

go undetected through the development process, resulting in software systems not meeting customers and users expectations, especially when the coordination with other components is an issue. Therefore, methods and frameworks helping software developers to better manage software requirements are of great interest for component based software.

Abstract interpretation (Cousot & Cousot, 1977) is a theory of semantics approximation for computing conservative over-approximations of dynamic properties of programs. It has been successfully applied to infer run-time properties useful for debugging (e.g., type inference (Cousot, 1997 and Kozen, Palsberg, & Schwartzbach, 1994)), code optimization (e.g., compile-time garbage collection (Hughes, 1992)), program transformation (e.g., partial evaluation (Jones, 1997), parallelization (Traub, Culler & Schauser, 1992)), and program correctness proofs (e.g., safety (Halbwachs, 1998), termination (Brauburger, 1997), cryptographic protocol analysis (Monniaux, 2003), proof of absence of run-time errors (Blanchet, Cousot, Cousot, et al., 2003), semantic tattooing/watermarking (Cousot & Cousot, 2004)). As pointed out in (Le Métayer, 1996), there is still a large variety of tasks in the software engineering process that could greatly benefit from techniques akin to static program analysis, because of their firm theoretical foundations and mechanical nature.

In this chapter we investigate the impact of Abstract Interpretation theory in the formalization and automatic verification of Non-Functional Software Requirements, as they seem not adequately covered by most requirements engineering methods ((Kotonya, & Sommerville, 1998), pag. 194). Non functional requirements can be defined as restrictions or constraints on the behavior of a system service (Sommerville, 2000). Different classifications have been proposed in the literature (Boehm, 1976, Davis, 1992, and Deutsch & Willis, 1988)), though their specification may give rise to troubles both in their elicitation and management, and in the validation process.

Let us start from a quite naive question: "*what do we mean when we say that a program is portable on a different architecture?*". In (Ghezzi, Jazayeri & Mandrioli, 2003) a software is said portable if it can run in different environments. It is clear that it is assumed not only that it runs, but that it runs the same way. And it is also clear that if we require that the behavior is exactly the same, portability to different systems (e.g., from a PC to a PDA, or from an OS to another) can almost never be reached. This means that implicit assumptions are obviously made about the properties to be preserved, and about the ones that might be simply disregarded. In other words, portability needs to be parameterized on some specific properties of interest, i.e. it assumes a suitable abstraction of the software behavior. The same holds also for other product non-functional requirements, like space and time efficiency, dependability, robustness, usability, etc. It is clear that, in this context, the main features of abstract interpretation theory, namely modularity, modulability, and effectiveness may then become very valuable.

The main concepts introduced in this chapter can be summarized as follows:

- We extend the usual abstract interpretation notions to the deal with systems, i.e. programs + architectures.
- We show that a significant set of product qualities (non-functional requirements) can be formally expressed in terms of abstraction of the concrete semantics when focusing on a behavioral property of interest. This yields an unifying view of product non-functional requirements.
- We show how existing tools for automatic verification can be re-used in this setting to support requirements validation; their practicality directly depends on the complexity of the abstract domains.