Understanding and Representing Deployment Requirements for Achieving Non-Functional System Properties

Moran Kupfer[1] and Irit Hadar[1]

[1] Department of Management Information System, University of Haifa
Haifa, Israel
{msrugo, hadari}@mis.haifa.ac.il

Abstract. Deployment requirements describe the precise, desired configuration of a software system. They relate the system's non functional requirements to its architecture, providing a basis for making decisions about design trade-offs in terms of the resulting system's non functional properties. The purpose of this position paper is to propose a research direction towards developing an approach for reasoning about deployment decisions. Its main objective is to quantitatively evaluate and select between different potential architecture solutions in order to shorten customer time-to-value and increase satisfaction. In this paper we analyze the relationship between deployment requirements and non-functional properties, and discuss work in progress of developing a deployment-based methodology for evaluating software architecture.

Keywords: Non-functional, properties, requirements, deployment, model-driven software engineering, ontology.

1 Introduction

Requirement Engineering (RE) is the part of software engineering that is concerned with the definition, formalization and analysis of the requirements that a potential system must have to accomplish specific organizational needs [7]. Requirements are divided into two types: Functional Requirements (FR) and Non-Functional Requirements (NFR). In order to meet these requirements, the resulting system, installed in the customer's site, is expected to encompass two types of properties: functional and non-functional respectively. However, some of the non-functional properties required from the system emerge during the design phase, taking into account the new constrains the software system's design poses, in addition to the customer's original requirements.

NFR in Software Engineering (SE) present a systematic and pragmatic approach for building quality into software systems. Systems must exhibit software quality attributes, such as accuracy, performance, security and modifiability [5]. The specific

non-functional deployment requirements (NFDR), which are all NFR related to deployment, are often not included in the original software requirements; part of the reason is that these requirements are not always known until the developers start designing the system [2]. The importance of NFR, and specifically of NFDR, stems from their contribution to the overall quality of the resulting system. When defined, they are usually detailed at the system level; when not defined, they are derived from other requirements, thus it is hard to formally represent them at all levels and determine whether they can be met.

This position paper presents an ongoing research, aimed at exploring the field of software systems deployment requirements and the resulting non-functional properties. The research main goal is to find ways to formally represent and validate NFDR. More specifically, the objectives of this investigation include: (1) developing ontology and a symbolic representation for formally representing and managing deployment requirements; (2) deriving component deployment requirements from the system's requirement; (3) understanding and capturing designers' reasoning in deployment decision making; and, (4) defining a set of measures adequate for supporting deployment decision making.

This paper is organized as follows: Section 2 presents the background including literature review with regard to the issues of NFR and deployment; Section 3 focuses on the research objectives and questions; Section 4 presents the study design; the preliminary results are presented in Section 5; and finally, Section 6 summarizes the expected contribution of this study.

2 Theoretical Background

2.1 Non-Functional Requirements

NFR are typically described as attributes of a system that contribute to the overall quality of the product [14]. These attributes include quality of service parameters and system properties However, NFR are usually hard to unambiguously define, represent and manage.

Once NFR are defined, they serve as selection criteria for choosing among different design choices. Errors of NFR omission are acknowledged to be among the most expensive and difficult to correct [14]. Poorly defining NFR leads to low quality solution and thus user low satisfaction and interest in the resulting system. In order to overcome this problem, several formalizations to NFR have been made over the years. For example, many researchers agree that developing formal definitions of NFR using the Goal Concept helps to evaluate whether they can be met (e.g. [6], [7] and [14]). The Goal Concept [14] is relevant for the elicitation, elaboration, structuring, specification, analysis, negotiation, documentation and modification of stakeholder's requirements. It is divided into 4 types of goals:

1. Functional Goals - services that the software is expected to deliver.
2. Non-Functional Goals - quality requirements that the software should satisfy.
3. Hard Goals – objective satisfaction verification, established using formal verification techniques.

4. Soft Goals – similar to hard goals, except that the criteria for whether it is achieved depend on customer satisfaction rate.

Knowing which qualities are needed to fulfill each goal is essential for developing correct and adequate systems [1]. There are many types of NFR frameworks within the Goal Concept approach, such as: A Process-Oriented Framework where goals represent NFR, design decisions and arguments in support or against other goals [14]; An Agent-Oriented Framework where goals are used to model agents' relationships and eventually link organizational needs to system requirements [7]; and, Goal-Directed Framework where a goal is a non-operational objective to be achieved by the composite system [6]. The goal of all types of framework is to help integrating NFR into the software development process. The framework itself is a tool designed to help evaluating, during system development, whether the NFR can be met.

In summary, NFR have limitations, many of which stem from their definition problem. One way to overcome the definition limitation was introduced by the literature using different formalization methods based, for example, on the Goal Concept presented above. Such formalization frameworks may enable to unambiguously define and evaluate: (1) whether a specific set of NFR can be met; and, (2) the effect they are expected to have on the resulting system. One of the objectives of our research is to study this and other frameworks and, based on them, find a specific solution for representing and validating deployment requirements.

2.2 Deployment Requirements

After developing a software system, all the activities that make it available for use are called software deployment [4]. Researchers refer to software deployment as a process which consists of several interrelated activities with possible transitions between them (e.g., [4] and [17]). Deployment is also defined as the processes between acquisition of software and execution of software [17], and can be characterized as the transformation of one software system configuration to another, based on the set of property values [9]. Deployment is the first step of component management life cycle, after development is completed. It is made up of component publishing, discovery, dependency resolution, downloading, installation, configuration and launching [9]. A deployment process covers post-development activities such as release, install, configure, plan, launch, de-install and de-release [12]. We will focus at the deployment requirements for the release activity, which is the interface of the deployment process with the development process.

The release activity [4] must determine the resources required by a software system to correctly operate at the target site. It must also collect the information that is necessary for carrying out subsequent activities of the deployment process; this information may be derived from a variety of sources including the developers' knowledge about the system structure and operation, the customer's knowledge about the current deployed applications and computational environment at the customer site, etc.

Deployment requirements are a type of NFR, thus face similar challenges. For example, some researches (e.g. [3]) suggest that NFR do not relate to a specific component and cannot be evaluated without observing the system as a whole.

Moreover, NFR are defined in general and cannot be viewed as a single requirement but as a collection of requirements. Nevertheless, in the context of NFDR it is important to be able to define deployment requirements for a single component, for purposes such as deploying alternative combinations of components [12], or adding/removing a specific component to/from an existing deployed system, etc.

The Software Deployment Descriptor (SDD) project [16] of the Organization for the Advancement of Structured Information Standards (OASIS) defines the attributes of a deployable component and its infrastructure, providing a model driven approach for installation, configuration and change management of applications in order to make good deployment decisions regarding a specific component. Lau and Ukis [13] suggest augmenting component's interface with metadata describing the component's behavior in order to understand its run-time behavior and thus help to deploy the component. According to them, deciding how to deploy a component depends much on its behavior, which is usually unknown, thus they emphasize the crucially of the designer's experience for making such decisions.

In our work we aim at capturing expert designers' reasoning in this context, in a way that will support others in making good deployment decision. For this aim a representational framework and relevant ontology will be developed, based on existing modeling frameworks. Ontology describes the concepts and relationships that are important in a particular domain, providing a vocabulary for that domain as well as a computerized specification of the meaning of terms used in the vocabulary [15]. Basically, ontology provides shared software engineering concepts- what they are, how they are related and can be related to one another, for representing and communicating software engineering knowledge [8]. To the best of our knowledge, no current ontology and no formal definition for deployment requirements exist.

2.3 Model-Based Representation

Abstraction, which is one of the most basic principles of software engineering, can be supported by visual models. Models provide an abstract representation of the developed system and can be iteratively refined and finally transformed into a technical implementation, i.e., a software system.

Model-Driven Software Development (MDSD) is an SE approach, consisting of the application of models and model technologies for elevating the level of abstraction at which developers create and evolve software. This approach is aimed at both simplifying (making it easier for the developer) and formalizing (standardizing, so that automation is possible) the various activities and tasks that comprise the software life cycle. MDSD imposes structure and common vocabularies so that its artifacts are useful for their main purpose in their particular stage in the life cycle, for the underlying need to: 1) link between related artifacts and 2) serve as a communication medium between participants in the project [10].

In MDSD, models are used for many purposes, including reasoning about problem and solution domains and documenting the stages of the software life cycle; the result is improved software quality and time-to-value, as well as reduced costs.

Component models technologies have been developed for supporting the processes of development and handling of complex software systems, allowing the

decomposition of systems and the use of software components in a distributed processing environment [11]. In this research we aim to embed formal representation of deployment required properties within the architectural artifact of component models.

Deployment of systems' component is one of the most burning problems for the majority of component models (e.g., [3], [11] and [12]). To deploy a component-based system, each component must first be instantiated, then interconnected and configured. Defining the requirements for deployment involves identifying the components' non-functional requirements for achieving the systems' non-functional properties.

3 Objectives and Questions

The main objective of this research is developing a formal framework for model-based representation of requirements related to deployment, which are NFR in nature. It will be based on frameworks from literature, using existing visual representations, expanding and adapting them to this use and basing them on proper ontology.

In addition, this research aims at understanding deployment decisions reasoning as a basis for supporting deployment requirements management and usage. For this aim, software designers are observed and questioned when making deployment related decisions. Furthermore, a full literature survey is conducted for learning about suggested deployment strategies. Based on observations and literature, the research objective is to understand and model deployment reasoning and develop a corresponding formal representation, which will be easy to use and comprehend by the relevant stakeholders.

Deployment decisions should be made in light of a target function, aiming for high quality of the system deployed under given constraints. However, in order to support deployment decisions it is essential to identify concrete measures as a basis for decision making and evaluation of the proposed solutions. Such measures can be static (e.g. component coexistence and dependencies) and dynamic (e.g. load, performance, volume). This research aims at identifying the relevant measures that influence and are influenced by deployment decisions, in order to provide guidance in deployment decision making.

Another objective is to examine the possibility of deriving deployment requirements to individual components. For this aim we empirically explore whether such derivation of NFDR from system to component is possible, and if so – how.

The research focuses on the following questions:

(1) What are NFDR?
(2) How can NFDR be formally represented within a model-driven software development framework?
(3) What are the considerations and reasoning underlying deployment decisions?
(4) What measures influence and are influenced by deployment decisions?

4 Research Method and Settings

This ongoing research explores current practices in research and industry for supporting requirements and representing information in architecture solutions. A field study is being performed for observing and interviewing designers during their deployment decision making. The case study is conducted in a large IT management software provider company which applies development methodologies emphasizing architecture. The study participants include R&D, design and implementation groups, all involved in the deployment of a security product. The data collection is conducted using interviews and observations.

In order to understand deployment, a better understanding of the policy and coordination modeling needs to be achieved. Qualitative research methods and tools are used to uncover and understand what lies behind the deployment decision making process. Specifically, the grounded theory approach used in this study is a qualitative research method that collects data about a phenomenon from its natural settings, in order to ground the findings or formulated theories in the field, with the researcher serving as the main research instrument. The purpose of the grounded theory method is to build a theory that is faithful to and illuminates the area under study [18].

5 Preliminary Results

The first stage for representing NFDR is to understand their relation to the systems' NFP. This enables representing them in architectural terms (i.e. components and connectors) as well as identifying a process for evaluating and choosing between different deployments choices (i.e. reflecting the reasoning behind deployment decisions).

Figure 1 presents a conceptual model for understanding the relations between requirements and architecture in the Component-Based Software Engineering approach as extracted from the field. A specific NFR, provided by a stakeholder, may be used for deriving NFDR, which are realized into component NFP, and thus can be assessed.

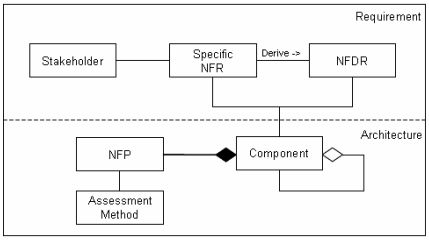

Fig. 1. From requirement to architecture: The deployment perspective

For example, a customer requests the following NFR: sending a message through the system will occur in less than a defined time unit. This specific NFR constrains the system NFP SEND_MESSEGE to the requested time unit. The NFDR derived

from this NFR define the required system's NFP configuration setting (such as memory usage, disk usage, load balance etc.), for achieving the NFP constraint. Since there can be more than one configuration, it is required to perform evaluation analysis for choosing the most suitable NFDR.

The evaluation analysis uses the assessment method, which takes into consideration the most influential measurement (as perceived by the designers) and the component's NFP involved, in order to evaluate the required satisfaction level.

Figure 2 presents a conceptual model for transforming NFR into a software solution. The NFDR are derived from the customer's NFR, and define configuration setting for system's NFP. This leads to possible architecture models. Then, an evaluation analysis is applied to choose between them. Finally, the most satisfying architecture model is selected.

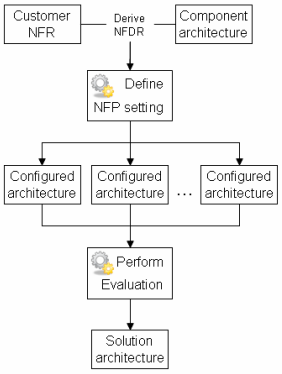

Fig. 2. The transformation process from requirements to solution architecture

The conceptual models presented above enable understanding NFDR and their relation with system NFP, as well as representing them in architectural terms. Further exploration of the transformation process will enable to develop an evaluation process, encompassing relevant measures for choosing between different deployment choices. This phase is currently taking place within our research work.

5 Conclusion

In this paper we presented an ongoing research work for understanding and representing deployment requirements in order to achieve predefined non-functional properties, as well as some preliminary results obtained so far. The final results of this study will offer a methodology for defining, representing and validating NFDR in model-driven software development projects. This methodology will enable to evaluate different architecture solutions and deployment configurations prior to the installation of the system at the customer's site. Such methodology is expected to reduce time investment in deployment decision making and increase the quality of the deployment process and the deployed systems, both in terms of customer's experience and deployment team support.

Acknowledgement: This work is supported by funding from CA Labs, CA Inc.
We thank Amir Jerbi, Ethan Hadar and Andreas Seibel for their helpful cooperation.

References

1. Adam, S. and Doerr, J. (2007) "On the Notion of Determining System Adequacy by Analyzing the Traceability of Quality", Advanced Information System Engineering, 19th International Conference, CAiSE 2007, pp. 325-329
2. Apte, A. (2002) "Java™ Connector Architecture: Building Custom Connectors and Adapters", Sams Publishing
3. Burge, J.E. and Brown, D.C. (2003) "NFR: Fact or Fiction?", Computer Science Technical Report, Worcester Polytechnic University, Link: http://web.cs.wpi.edu/~dcb/Papers.
4. Carzaniga, A., Fuggetta, Hall, R.S., Heimbigner, D., van der Hoek, A. and Wolf, A. (1998) "A Characterization Framework for Software Deployment Technologies", Technical Report CU-CS-857-98, Department of Computer Science, University of Colorado, Link: http://serl.cs.colorado.edu/~carzanig/papers/CU-CS-857-98.pdf
5. Chung, L., Nixon, B.A. and et, al. (2000) "Non-Functional Requirement in Software Engineering", Kluwer Academic Publisher
6. Dardenne, A., van Lamsweerde, A. and Fickas, S. (1993) "Goal-directed requirements acquisition", Science of Computer Programming, Vol. 20, Issue 1-2, pp. 3-50
7. Donzelli, P. and Bresciani, P. (2004) "Improving Requirements Engineering by quality Modeling A Quality-based Requirements Engineering Framework", Journal of Research and Practice in Information Technology, Vol. 36 Issue 4, pp. 277-294
8. Gruber, T. (1995) "Towards principles for the design of ontologies used for knowledge sharing", International Journal of Human-Computer Studies, 43(5/6), 907-928.
9. Hall, R.S., Heimbigner, D, and Wolf, A.L. (1999) "A Cooperative Approach to Support Software Deployment Using the Software Dock", ISCE.
10. Hailer, B. and Tarr, P. (2006) "Model-Driven development: The good, the bad, and the ugly", IBM Systems Journal, Vol. 45, November 3, 2006.
11. Hofman, A. and Neubauer, B. (2005) "Deployment and Configuration of Distributed Systems", System Analysis and Modeling, Lecture Notes in Computer Science, Vol. 3319, pp. 1-16
12. Hnětynka, P. (2005) "Making deployment process of distributed component-based software unified", Ph.D. Thesis, Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague. Link: http://dsrg.mff.cuni.cz/publications/Hnetynka-PhD-thesis.pdf
13. Lau, K.K. and Ukis, V. (2006) "Deployment Contracts for Software Components", Pre-print CSPP-36, School of Computer Science, the University of Manchester
14. Mylopoulos, J., Chung, L. and Nixon, B. (1992) "Representing and Using Nonfunctional Requirements: A Process-Oriented Approach", IEEE Transactions on Software Engineering, Vol. 18 Issue 6, pp. 483-497
15. Noy, N. F. and McGuinness, D.L. (2000) "Ontology Development 101: A Guide to Creating Your First Ontology", Stanford University, Link: http://protege.stanford.edu/overview/
16. Organization for the Advancement of Structured Information Standards (2007). "Solution Deployment Descriptor Specification 1.0", OASIS working draft
17. Object Management Group (2004), "Deployment and Configuration of Component-based Distributed Applications Specification", OMG working draft, Link: http://www.omg.org/docs/ptc/04-08-02.pdf
18. Strauss, A. and Corbin, J. (1990) Basics of Qualitative Research Grounded Theory Procedures and Techniques, Sage Publications, Inc