Software Architecture for Systems of Software Intensive Systems (S3): The Concepts and Detection of Inter-System Relationships

John Brøndum
University of New South Wales and NICTA
13 Garden Street
Eveleigh, Sydney, Australia
johnb@cse.unsw.edu.au, john.brondum@nicta.com.au

ABSTRACT

Key to software architecture is the description of relationships between software components [10] supported by commonly understood semantic definitions [9][8]. However, the definitions do not adequately capture the inter-system level software relationships. This leaves software architects either unaware of critical relationships or, to ‘roll their own’ based on aggregations of code-level call/use structures. This leads to critical gaps in the architectural description and communication problems within distributed development environments - as poorly understood relationships can inadvertently propagate changes and break system interoperability [2]. The solution requires a description of new system level relationships and a new systematic, repeatable technique to detect both immediate and linked system level relationships. The solution will be developed through the mining of existing software ecosystems and industry systems of software intensive systems (S3) architectures. Validation will be performed through case studies from industry collaborations.

Categories and Subject Descriptors

H.1.0 [Information Systems]: Models and Principles—General; D.2.11 [Software Engineering]: Software Architecture—Languages; D.2.12 [Software Engineering]: Interoperability—Distributed objects

General Terms

Design

Keywords

Software Architecture, Systems-of-Systems

1. INTRODUCTION

Software architecture is defined as the combination of elements, form and rationale [10]. Elements are of three main types: ‘processing’, ‘data’ and ‘connectivity’. Form captures the constraints imposed on the properties of the elements and relationships between the elements. The rationale is the explicit architectural reasoning guiding subsequent realization and modification.

Figure 1: The three levels of Software Relationships

As illustrated in the above figure, a modular structure between systems leads to a degree of coupling at various stages of the software lifecycle. A coupling may have different dimensions including but not limited to technology, protocol, location or identity[5]. However, the relationship between two systems is potentially more than ‘just’ the aggregation of (2) and (3). In 2009, Twitter [14] sought to limit the permitted number of Twitter client authentications without requiring changes to the Twitter API. The change did introduce new time and state based couplings breaking the existing interoperability, as it required Twitter clients to track the number of authentications per hour. In 2007, Skype suffered a two-day global outage apparently triggered by Microsoft distributing a software update[13]. The update caused a significant number of computers to restart and send authentication requests beyond Skype’s network capacity.

2. CURRENT RESEARCH

Current literature reveals research of relationships within architectural decisions[15], amongst architectural models[11], and dependency analysis[6]. Operational relationships of (failure) cause and impact [1] have been studied through statistical analysis. Other efforts have looked at management aspects of relationships between architecture and quality attributes [3], as well as architecture and work dependencies[4]. But neither of these enhance our understanding of software relationships beyond the traditional modular structures. The proposed research differs to software contracts[16], as it seeks to capture implicit relationships (e.g., not directly expressed in an interface) and support the architectural process prior to the detailed interface specification.
3. PROPOSED RESEARCH

The proposed research is focused on two key questions: 1) What relations can exist between software systems that impact one or more coupling aspects between two or more applications? Sub-questions include: What is the conceptual meaning of inter-system level relations? What are the attributes of the relations? How do they relate to 'coupling dimensions'?; and 2) How can 'linked relationships' (two or more degrees of separation) causing change propagation be detected? Sub-questions include: What kind of changes will be propagated? Will the change propagation always happen or only under certain conditions?

3.1 Solution & Contribution

The proposed solution consists of two parts: 1) a description of new system level relationships, and 2) a new systematic, repeatable technique to detect both immediate, and linked system level relationships, causing change propagation - impacting interoperability. It’s anticipated that new relationships will cover aspects such as timing, implied NFR or business rules, and shared resources. The detection technique is aimed at developing a novel algorithm to identify linked relationships utilising Design Structure Matrices.

3.2 Methodology

The solution development will be based on an iterative approach leveraging: 1) Existing research literature, and 2) exploratory case studies of open source communities using software architecture reconstruction methods. Suitable case studies must display all of the characteristics of S3 [7] and maintain a repository of defect and/or change history logs detailing how changes in one system affected another system.

3.3 Validation Plan

The validation plan is based on case studies [12] utilising four high level steps: 1) Fit for purpose assessment of the software system level relationship semantics, using competency questions, 2) Apply detection technique to detect new relationships, 3) Identification of change propagation affecting interoperability, and 4) Evaluation of the technique’s ability to repeatably, and systematically, detect change propagation affecting interoperability - benchmarked against expert opinion and system change history list.

3.4 Progress

The proposed research is entering the detailed exploratory case-study phase.

4. CONCLUSIONS

Key to software architecture is the description of relationships between software components [10] supported by commonly understood semantic definitions [9][8]. However, the definitions do not adequately capture the inter-system level software relationships. The solution requires a description of new system level relationships, and a new technique to detect inter-system software level relationships, developed through the mining of existing software ecosystems and industry S3 architectures investigating the three levels of software relationships. Validation will be performed through case studies from industry collaborations.

5. ACKNOWLEDGMENTS

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

6. REFERENCES