Towards an Architectural Viewpoint for Systems of Software Intensive Systems

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

Liming Zhu
University of New South Wales and NICTA
13 Garden Street
Eveleigh, Sydney, Australia
liming.zhu@nicta.com.au

ABSTRACT

An important aspect of architectural knowledge is the capture of software relationships[25]. But current definitions[25] [21][23] do not adequately capture external system relationships[5], and offer no guidance on *implicit* relationships[29]. This leaves architects either unaware of critical relationships or, to 'roll their own' based on aggregations of code-level call structures, resulting in critical architectural gaps and communication problems within Systems of Software intensive Systems (S3) environments[2]. These environments may also restrict the sharing of architectural knowledge due to either legal, or contractual constraints, or overwhelm due to the size and number of involved systems adding to the challenges of identifying and describing the relationships.

This paper presents a novel S3 Architectural Viewpoint consisting of; 1) an extensible taxonomy of relationships (building on existing relationship concepts), 2) a systematic, repeatable technique to detect both immediate and composite relationships, and 3) proposes the Annotated Design Structure Matrix to link S3 views, with existing dependency analysis technique. The goal is an architectural approach for sharing and analysis of architectural knowledge relating to relationships, in an S3 environment. The research is ongoing and 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*

General Terms

Design

Keywords

Software Architecture, Systems-of-Systems

SHARK '10, May 2-8 2010, Cape Town, South Africa

1. INTRODUCTION

Software intensive systems are defined as a collection of components organized to accomplish a set of functions[25] with the combined ability to meet the concerns of designated stakeholders[14]. The challenge of an S3 environment is multiple stakeholder groups and their ability to operate and manage their systems independently[19]. Each group is identified through their separate goals and motivations[15] and their ability to enforce (governance) control upon the system. The research will initially focus on S3 environments with the existence of these independent groups.

Software architecture plays a key role as a stakeholder communication tool to relay the rationale[10] behind not only the final architecture, but also important intermediate and alternative solutions. It captures the reasoning and assumptions for the choice in components, their permitted properties and the overall relationships[28], as well as external dependencies[29].

Each S3 participating system is managed by an independent group of *stakeholders* . From an S3 viewpoint, this results in a system level selection of the S3 components (i.e., 'a system in its own right'[19]) and permitted properties, yet the relationships are determined through the interoperability of the S3 systems[32]. The aim is to develop an Architectural Viewpoint that supports the sharing and analysis of architectural knowledge relating to software system relationships in an S3 environment.

The remainder of the paper is organised into the following sections: Section 2 describes related work. Section 3 describes the proposed S3 Architectural Viewpoint, and Section 4 provides the conclusion.

2. RELATED WORK

Perry and Wolf[25] stated that "*Relationships are used to constrain the 'placement' of architectural elements - that is, they constrain how the different elements may interact and how they are organized with respect to each other in the architecture*".

A number of software design 'orientations' have provided a set of relationship concepts to aid designers and architects with the placement, including object oriented design[23], aspect oriented programming[17], 'design by contracts'[22], and 'service oriented computing'[24]. Common to all are a focus on direct code-to-code interfaces. This increases the risk of making architectural decisions too early in the decision process[33], as it distracts from the appropriate decision context[20]. This adds risks of additional expense if the deci-

sion is proven inappropriate - both in terms of lost credibility and project development cost.

Current literature covers research of relationships within architectural decisions[33], amongst architectural models[26], and dependency analysis[13]. Operational relationships of cause and impact[1] have been studied through statistical analysis. Other efforts have looked at management aspects of relationships between architecture and quality attributes[4], as well as architecture and work dependencies[6]. But neither of these enhance our understanding of software relationships beyond the traditional modular structures.

The environmental and external system context through *contextual viewpoints*[29], or as context diagrams[7], have been noted as an important way to define the boundary of a system, but both approaches are only concerned with connector style relationships. The contextual viewpoint description warns of the common pitfall of *implicit* relationships between systems[29] without attempting to identify or describe these 'intangibles'.

Enterprise Architecture represents means of capturing the relationships between business operations and the supporting systems. Gaps have been identified in the support for S3 contexts due the underlying assumption of a single, ultimate source of control[3]. The proposed solution offers a modelling technique to support the proposed collaborative model enhancements[3] to the Zachman framework[30].

3. S3 ARCHITECTURAL VIEWPOINT

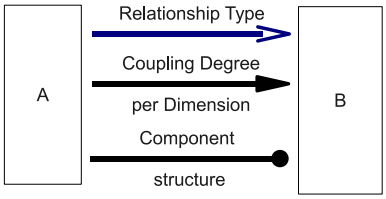

Figure 1: The three levels of Software Relationships

Our concern is the relationships between systems. As illustrated above, the inter-system relationships can be viewed as three levels of abstractions; 1) relationship types (section 3.2.1) coupling degree per dimension (section 3.2.3), and 3) if present, the component / modular structure to support the relevant coupling dimension (e.g., the UML view). The objective of the S3 Architectural Viewpoint is focused on enable answers to the following questions[5]:

Types of Relationships: What relations can exist between software systems that impact one or more coupling aspects between two or more applications? What is the conceptual meaning of inter-system level relations?

Change propagation: How can 'composite relationships' (two or more degrees of separation) causing change impact propagation be detected? What kind of changes will be propagated? Will the change propagation always happen or only under certain conditions?

3.1 Viewpoint Modelling

3.1.1 Taxonomy of Relationship Types

Viewing an S3 as a network of relationship, there is at least the following basic types of relationships: 1) 'relates to' (base abstract type), 2) 'depends on', 3) 'constrains', and 4) 'refers to', and 5) 'connects to'. These initial relationship types have been compiled based SOA[9] and Enterprise Integration Patterns[12].

R : A 'relates to' B: This is the most abstract form, where not a lot is known about the relationship between A and B. This relationship forms the base type for the other subsequent types, with four attributes: 1) 'Attribute' - an abstract type to capture unique characteristics about the relationship, 2) 'Dimension' paired with a 'coupling degree', 3) 'Style' - Architectural Style utilised to realise the relationship, and 4) Weight - an abstract sub-type of 'Attribute', to capture unique environmental characteristics (e.g., sub-type may be created to capture 'number of permitted connections' for a 'connect with' relationship).

RD : A 'depends on' B: Specialisation of 'relate to', where A is functionally dependent on B. This can be broken into *functional* (logical, implicit) or *resource* (structural, explicit) dependencies – yet not the same as the connect relationship below. Example: A creates purchase orders, but cannot complete the creation without a successful credit check performed by B.

RC : A 'constrains' B: A acts as a constraint to B. Example: A is a proxy for access to B (e.g., single sign on, media streaming, reverse proxy), or A limits B's ability to perform it's function (e.g., A may be a currency trading platform with limited trading windows per currency, yet B needs to be able to accept and process trades for all currencies 24x7). Note that in these scenarios, A and B do *not* require direct interfaces to exist (e.g., relationship type 'connects with').

RR : A 'refers to' B: A's data, function or otherwise is mastered by B. Example: A shopping cart system provides account balance information for authenticated customers calculated by the account system. The relationship between the two systems can either be modelled as a) 'depends on', if the custom is not allowed to progress without, or b) 'refers to', if the customer can check out using only a credit card. The 'refers to' relationship may be specialised into a 'data', 'function', 'process' or other subtypes.

RCw : A 'connects with' B: A is structurally connected to B via a *connector*[21].

Using the above taxonomy, we can develop definitions for the *implict* relationships[29]: if a relationship is detected in one or more dimensions, but the associated attributes are not captured as part of the software contract or service interface. A relationship is *indirect* if System A has a 'decoupled' relationship with System B, and C has a relationship with both A and B other than 'decoupled'.

Similarly, we can define *composite relationships* as a chain of relationships where all of the relationships share one or more attribute types.

3.1.2 Example

The below diagrams contain an example from a fictitious company running an online retail store. The connect view of the end-to-end solution architecture is 'classic boxology'[27] illustrating a number of systems and their connections. In the alternative 'refers to (data)' view, the lines have been redrawn to reflect relationships in terms of who 'masters' the data utilised by other systems. E.g., 'Proxy' connects with 'LDAP', and 'Content' refers to data mastered by 'Shopping Engine'.

Even simple diagrams utilising the proposed taxonomy

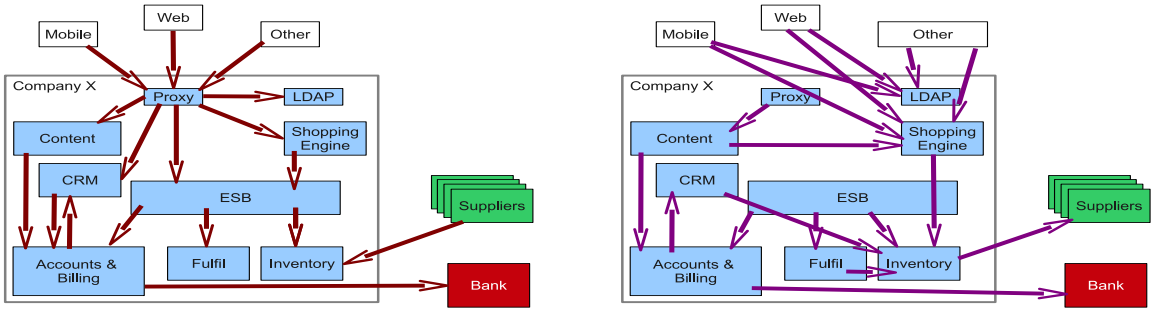

Figure 2: The 'connects with' (left) and 'refers to' views

draws out a few points regarding connection focused architecture documentation: a) It distorts the relative importance of proxy or integration systems. From a data management point of view, company X might find the Shopping Engine and Inventory systems to be more important, and b) it can hide major gaps, e.g., figure 2 (left) gives the impression of a well integrated company, but figure 2 (right) highlights a interoperability gaps between the Billing and Inventory systems.

3.1.3 *Coupling Degrees per Dimension*

The concept of 'loosely coupled' can be interpreted as the degree of change propagation between two components, or how much A is able to tolerate B changing[11]. Changes propagate along one or more *dimensions* (also know as 'sources' [11] or 'aspects'[16]) including; messages or otherwise shared data, interface parameters, (technical) environment, technology, protocol including control, language, location, time, security including identity, quality attributes. The definition of coupling degrees is an enhancement to the 'service relationship styles'[16] and proposed as follows:

Coupled: The relationship between S1 and S2 is *coupled* in dimension D, if changes to S1 affecting D also require changes to S2 and/or any intermediate systems.

Declared: The relationship between S1 and S2 is *declared* in dimension D, if changes to S1 affecting D only require changes to intermediate systems, and S2 does not require any changes.

Transformed: The relationship between S1 and S2 is *transformed* in dimension D, if changes to S1 affecting D can be transformed by intermediate systems, such that S2 does not require any changes.

Negotiated: The relationship between S1 and S2 is *negotiated* in dimension D, if changes to S1 affecting D can be negotiated by S2 or intermediate systems, such that S2 does not require any changes.

Decoupled: The relationship between S1 and S2 is *decoupled* in dimension D, if changes to S1 affecting D do not require any changes to S2 or intermediate systems.

Good system to system interoperability is then interpreted as low 'coupling degree' per 'dimension' for each of the identified relationships.

3.1.4 *Identification and Composition of Relationships*

Identification: The identification process is broken into three high-level activities: a) *'External'*: Identification of relationships from the perspective of the system to other S3 systems. For each relationship, establish the dimensions of the relationship and the desired coupling degree as an extention existing architectural evaluation methods[31] [8], b) *'Internal'*: For each dimension, reference architecture or architectural style is selected to meet the requirements. The aim is to ensure accurate relationship information to be communicated to the stakeholders of System B, but without exposing the detailed structural and behavioural views of A to B, and c) *'View synchronisation'* assesses the following questions: Do the views align, e.g., do the architect for system A agree with the architect for system B, about the set and nature of relationships between the two systems?

Composition: The assessment of relationships follow a pattern similar to standard clustering analysis techniques developed utilising *design structure matrix* ('DSM')[18]. The proposed approach is an extension utilising the proposed taxonomy to create an *Annotated DSM*. As illustrated below (read per row, left to right), relationship types are marked between systems, e.g., row 1 reads: A 'connects with' C and D).

	A	B	D
A	o	RCw	RCw
B	RCw	o	RR
D	RCw, RC	RC	o

The analysis activities enabled by the ADSM, cover a) Identification of system clusters (A, B, and C) or proxy (C), including an explicit representation of implicit relationships, b) Identification of composite relationships through the matching of common attributes, e.g., C -RC-> B -RCw-> A is a composite relationship, if the two relationship types have common attributes, and c) validation of the identified coupling dimensions and architectural styles to detect architectural mismatch potentials.

4. CONCLUSIONS

The semantic definitions of *software relationships*[25] do not adequately capture inter-system level relationships[5] [21] [23], and offers no guidance on *implicit* relationships[29]. This leads to critical architectural gaps and communication problems within S3 environments [2]. This paper presents a novel and light-weight S3 Architectural Viewpoint consisting of; 1) an extensible taxonomy of system level relationships, 2) a systematic, repeatable technique to detect system level relationships, and 3) proposes the Annotated DSM as the basis for dependency analysis. The continued effort into the exploration of relationships and the validation of the viewpoint is in progress.

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

[1] A. Adi, D. Gilat, R. Ronen, R. Rothblum, G. Sharon, and I. Skarbovsky. Modeling and monitoring dynamic dependency environments. In *ICSC 2005, IEEE Int. Conf. on Services Computing,*, pages 208–214, 2005.

[2] M. Bass, V. Mikulovic, L. Bass, H. James, and C. Marcelo. Architectural misalignment: An experience report. In *WICSA '07, 6th Working IEEE/IFIP Conf. on Soft. Arch.*, pages 17–17, 2007.

[3] P. J. Boxer and S. Garcia. Enterprise architecture for complex system-of-systems contexts. In *Proceedings of IEEE International Systems Conference 2009*, pages 253–256, Vancouver, BC, Canada, 2009.

[4] H. P. Breivold, I. Crnkovic, R. Land, and S. Larsson. Using dependency model to support software architecture evolution. In *ASE '08, 23rd IEEE/ACM Int. Conf. on Automated Software Engineering - Workshops.*, pages 82–91, Sept. 2008.

[5] J. Brondum. Software architecture for systems of software intensive systems (s3): The concepts and detection of inter-system relationships. In *ICSE '10, 32nd ACM/IEEE int. Conf. on Soft. Eng. - Doc. Symposium*, May 2010.

[6] M. Cataldo, R. B. LLC, J. A. Roberts, and J. D. Herbsleb. Software dependencies, work dependencies, and their impact on failures. *IEEE Transactions on Software Engineering*, 2009.

[7] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little. *Documenting software architectures:views and beyond*. Addison-Wesley, 2002.

[8] P. Clements, R. Kazman, and M. Klein. *Evaluating software architectures: methods and case studies*. Addison-Wesley, 2006.

[9] T. Erl. Soa patterns, 2009. http://www.soapatterns.org - Acc. 31 Jan 2010.

[10] R. Farenhorst, J. F. Hoorn, P. Lago, and H. van Vliet. The lonesome architect. In *WICSA/ECSA '09, Joint Working IEEE/IFIP Conf. on Soft. Arch. & European Conf. on Soft. Arch.*, pages 61–70, 2009.

[11] R. High Jr, G. Krishnan, and M. Sanchez. Creating and maintaining coherency in loosely coupled systems. *IBM Systems Journal*, 47(3):358, 2008.

[12] G. Hohpe. Patterns and best practices for enterprise integration. http://www.enterprise integrationpatterns.com - Acc. 31 Jan 2010.

[13] S. Huynh, Y. Cai, Y. Song, and K. Sullivan. Automatic modularity conformance checking. In *ICSE '08: 30th Int. Conf. on Software Engineering*, pages 411–420, 2008.

[14] IEEE. Recommended practice for architectural description of software-intensive systems. 2007.

[15] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *WICSA '05, 5th Working IEEE/IFIP Conf. on Soft. Arch.*, pages 109–120, 2005.

[16] M. Keen, A. Acharya, S. Bishop, A. Hopkins, S. Milinski, C. Nott, R. Robinson, J. Adams, and P. Verschueren. *Patterns: Implementing an SOA Using an Enterprise Service Bus*. IBM Redbooks. 2004.

[17] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proc. of ECOOP*, volume 1241, pages 220–220, Jyvaskyla, Finland, 1998.

[18] U. Lindemann. The design structure matrix (dsm) - dsmweb.org, 2005. Accessed 31st Jan. 2010.

[19] M. W. Maier. Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284, 1998.

[20] R. Malan and D. Bredemeyer. Less is more with minimalist architecture. *IEEE IT Pro*, page 48, 2002.

[21] N. R. Mehta, N. Medvidovic, and S. Phadke. Towards a taxonomy of software connectors. In *ICSE '00, 22nd Int. Conf. on Soft. Engineering*, pages 178–187, 2000.

[22] B. Meyer. Applying 'design by contract'. *IEEE Computer*, 25(10):40–51, 1992.

[23] OMG. Unified modelling language v2.2, Feb 2009.

[24] M. P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *WISE '03: 4th Int. Conf. on Web Info. Systems Eng.*, 2003.

[25] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *SIGSOFT Soft. Eng. Notes*, 17(4):40–52, 1992.

[26] A. Radjenovic and R. F. Paige. The role of dependency links in ensuring architectural view consistency. In *WICSA 2008. 7th Working IEEE/IFIP Conf. on Soft. Architecture*, pages 199–208, 2008.

[27] M. Shaw and P. Clements. A field guide to boxology: preliminary classification of architectural styles for software systems. In *COMPSAC '97. 21st Int. Comp. Soft. and App. Conf.*, pages 6–13, 1997.

[28] J. Tyree and A. Akerman. Architecture decisions: Demystifying architecture. *IEEE Software*, 22(2):19–27, 2005.

[29] E. Woods and N. Rozanski. The system context architectural viewpoint. In *WICSA/ECSA '09, Joint Working IEEE/IFIP Conf. on Soft. Arch. & European Conf. on Soft. Arch.*, pages 333–336, 2009.

[30] J. A. Zachman. A framework for information systems architecture. *IBM systems journal*, 26(3):276, 1987.

[31] L. Zhu, M. Staples, and R. Jeffery. Scaling up software architecture evaluation processes. In *ICSP '08, Proc. on International Conference on Software Process*, volume 5007, page 112. Springer, 2008.

[32] L. Zhu, M. Staples, and V. Tosic. On creating industry-wide reference architectures. *EDOC '08. 12th Int. IEEE Enterprise Dist. Object Computing Conf.*, pages 24–30, 2008.

[33] O. Zimmermann, U. Zdun, T. Gschwind, and F. Leymann. Combining pattern languages and reusable architectural decision models into a comprehensive and comprehensible design method. In *WICSA '08, 7th Working IEEE/IFIP Conf. on Soft. Architecture*, pages 157–166, 2008.