e-Informatica Software Engineering Journal, Volume 7, Issue 1, 2013, pages: 45–52, DOI 10.5277/e-Inf130105

# From Principles to Details: Integrated Framework for Architecture Modelling of Large Scale Software Systems

Andrzej Zalewski[*], Szymon Kijas[*]
[*]*Institute of Automatic Control and Computational Engineering, Warsaw University of Technology*
a.zalewski@elka.pw.edu.pl, s.kijas@elka.pw.edu.pl

**Abstract**

There exist numerous models of software architecture (box models, ADL's, UML, architectural decisions), architecture modelling frameworks (views, enterprise architecture frameworks) and even standards recommending practice for the architectural description. We show in this paper, that there is still a gap between these rather abstract frameworks/standards and existing architecture models. Frameworks and standards define what should be modelled rather than which models should be used and how these models are related to each other. We intend to prove that a less abstract modelling framework is needed for the effective modelling of large scale software intensive systems. It should provide a more precise guidance kinds of models to be employed and how they should relate to each other. The paper defines principles that can serve as base for an integrated model. Finally, structure of such a model has been proposed. It comprises three layers: the upper one – architectural policy – reflects corporate policy and strategies in architectural terms, the middle one –system organisation pattern – represents the core structural concepts and their rationale at a given level of scope, the lower one contains detailed architecture models. Architectural decisions play an important role here: they model the core architectural concepts explaining detailed models as well as organise the entire integrated model and the relations between its submodels.

## 1. Introduction

Large scale software intensive systems are built to serve country- or worldwide organisations employing thousands of users. They span across the organisation's entities and locations often needing to cope with distributed data storage management and processing. The research presented in this paper has been motivated by the lack of effective approaches to the modelling of architecture of such systems. This is caused by the gap existing between the modelling frameworks and software architecture models.

The modelling frameworks classify information describing system structure rather than indicate precisely how this information should be represented with appropriate architecture models. On the other hand, existing models and modelling approaches usually represent selected viewpoint on system architecture, however it is not clear how these different models should be integrated to create an effective architecture modelling engine.

The research envisaged in this paper is aimed at integrating various models and modelling approaches into a uniform, integrated framework providing a precise guidance on employed models and structure of their relations. The rest of the paper is organised as follows: state-of-the-art in architecture modelling is analysed in section 2, section 3 presents the basic rulesguiding the design of an integrated architecture model, section 4 contains a proposition of such an integrated architecture model illustrated with a real world example, concept summary and range of further research comprise section 5.

## 2. State-of-the-art in Architecture Modelling

Architectural description of software-intensive systems seems to be a well established practice. It is defined by IEEE Std 1471-2000 and draft standard ISO/IEC 42010:2007 [1]. These standards comprise the most important concepts of the architecture genre as: stakeholders' viewpoints, stakeholders' concerns, architecture views [2] and even architecture rationale [3]. However, these standards do not indicate how architecture modelling should be done in practice, i.e. what kind of models shall be used, how such a suite of architecture models should be organised, verified etc.

We argue that this challenge have not been met yet at least in case of the modelling of large scale software systems. The above standards are rather of a declarative than of an imperative style.

Enterprise architecture frameworks as Zachman Framework [4, 5] or The Open Group Architecture Framework (TOGAF) [6] belong to the same declarative genre: they classify information describing architecture neither indicating how this information should be represented and what models shall be applied nor how different classes of information are interwoven and interact with each other.

On the other hand, there is a large variety of architecture description languages (ADL). ADL's as ACME, Wright, Aesop, UniCona and xADL (for full reference see [7]) have not reached the level of industrial application maturity. In contrast: Unified Modeling Language (UML) plays a role of an industrial standard, however, because of its origin, it is rather recognized as a set of models of software than system architecture. No wonder, UML is mainly applied in the context of 4+1 views [2] of software architecture (logical view, process view, physical view, development view, scenarios).

Architectural decisions [8, 9] are often perceived as another wave in architecture modelling – compare "third epiphany" in [3]. The idea that systems architecture, as every design, results from a set of decisions seems to be strongly appealing to both engineers and scientists. Architectural decisions can potentially represent any architectural concept belonging to any architectural view [3]. On the contrary to the other architecture models as UML or ADL's architectural decisions help to capture design rationale, which is a part of tacit knowledge which usually evaporates as soon as design is ready or as architect is gone. This ability to capture design intent is perceived as the most important advantage that architectural decisions provide.

However, hopes that architectural decisions alone will become an effective model of software architecture seem to be unfounded, especially in case of large scale software systems. The fundamental limitations of this modelling approach have been investigated in our former paper [10]. Architectural decisions are represented as text records, sometimes accompanied with illustrating diagrams [11]. The limitations of textual models are well-known in the genre of software engineering. Therefore, sets of hundreds of architectural decisions necessary to sufficiently represent architecture of a large system, are difficult to comprehend, analyse, verify and ensure completeness and consistency or even just to navigate through them.

This creates a substantial risk that modelling approaches based only on architectural decisions will collapse under their own weight as they create complexity of their own rather than helping to control complexity of system architecture. The effort and cost necessary to create and maintain such a large set of architectural decisions can discourage engineers and managers from using them at all. The existence of standards (e.g. IEEE-1471) and commercial modelling frameworks (e.g. TOGAF) should indicate that architecture modelling have matured to the industrial application. However, the gap between the declarative standards/frameworks and concrete architecture models still exists. For this reason, newer frameworks and models still emerge – compare recent developments: architectural decisions [12], recent versions of TOGAF [6] or Archimate notation and modelling approach [13] both promoted by The Open Group.

The challenge is to make an efficient modelling framework out of the existing models and frameworks (at least parts of them), in accordance with existing standards.

## 3. The Basic Rules for the Design of an Integrated Model of System Architecture

The observations presented below are supposed to exploit the advantages of the models and approaches presented above, while trying to minimise their drawbacks. They are aimed at providing foundation for integrated models of software architecture.

1. **Design rationale should be captured only for the most important design elements. Hence, architectural decisions should express only the core design concepts that are necessary to comprehend the structure of a given design component. They should by no means express design details.**
   The value of good system architecture is that it defines a kind of a skeleton defining basic organisation of every system's entity. This skeleton should remain almost unchanged throughout the life time of a given system entity. Here are just two examples of such skeleton-decisions:
   - design pattern (e.g. broker, model-view-controller) defines fundamental design structure of a given software component usually remaining unchanged as long as the component exists;
   - decision that the corporate systems will be integrated at two levels: domain and enterprise: there will be systems (e.g. Enterprise Service Bus – ESB , Business Process Management – BPM) integrating systems belonging to certain domains (e.g. sales, financial management) and a separate system integrating domains at corporate level – provides a structure, to which future system developments have to be tailored.

   Representing these basic structural concepts does not require modelling of all the details with architectural decisions as diagrammatic models are usually more efficient. This should help to overcome the intrinsic limitations of architectural decisions simultaneously making ADL/UML models easier to comprehend.
2. **Models representing the details of systems architecture should be chosen adequately to the class and properties of the modelled system as well as its stakeholders' concerns.**
   This observation is a consequence of the former one: core concepts can effectively be modelled with architectural decisions, while efficient modelling of design details can only be done with appropriately chosen models. Efficient architecture models will be different for different classes of systems – compare e.g. Service Oriented Architecture (SOA) and real-time systems. It is also worth noting that different models can be useful for different stakeholders' concerns – e.g. performance, security, reliability. Hence, the contents of integrated model should be chosen with respect to both the selected class of systems and concerns of architecture stakeholders.
3. **Architectural decisions should explain detailed models**
   Models of systems architecture like ADLs or UML are usually easier to comprehend when their underlying concepts are clearly stated – e.g. it is much easier to analyse a class diagram for model-view-controller component if you know in advance that this pattern has been followed. It is often difficult to deduce such an intent straight from the class diagram itself. Linking architecture decisions with architecture models can make ADLs more ef-

ficient. This is quite an opposite approach to [11], where architectural decisions are illustrated with diagrams. In fact, both possibilities are included in the proposed model.

4. **Architecture should be represented at different levels of detail/scope/view (scope [14]), being useful for different stakeholders**
   The complexity of large scale software system architecture results from the fact that organisation of software systems can be perceived at different levels of details and from the viewpoints of different stakeholders. This is both virtue (helps to cope with systems complexity) and vice (the relations between models at different levels of scope and/or belonging to different views are by no means clear increasing the overall architecture model complexity).
5. **Architectural decisions related to the detailed models can become a kind of an index helping to navigate through them**
   The architectural decisions are a versatile model of system architecture modelling efficiently only its core concepts. As such they can be used to integrate all the models across all the levels of scope, detail and views. If detailed models are appropriately linked to the core architectural decisions they can be used as a kind of an index to the detailed models.
6. **Integrated model should support systems evolution**
   Almost all the software systems are subject to changes throughout their lifetime. Hence, architecture models representing current-state system architecture only are of a limited use nowadays. Mechanisms of capturing changes, presenting model snap-shots for a selected moment of time should accompany an integrated architecture model.
7. **Support alignment of systems architecture with business strategy/policies**
   The need for the alignment with business strategy and policies does not have to be explained. However, it is usually difficult to assess whether architecture of systems or IT products really supports business strategy. Therefore, integrated architecture model shall make such an assessment easier.
8. **Promote and enable validation/verification of one model against other connected ones (especially higher level models).**
   As architectures are modelled at different levels of scope, detail and from different points of view these models can potentially be assessed/verified/validated one against another. This requires that architecture models are appropriately organised and interrelated with each other.

## 4. The Integrated Model of Large Scale Software System Architecture

The concepts of integrated model of large scale software systems will be illustrated with an example of a real system used in the banking sector. The system presented in fig. 1 has been developed to support the exchange of various kinds of information and documents (claims, direct debits, information concerning accounts moved from one bank to another, etc.) between banks and other institutions (e.g. bailiffs' offices, social security agencies). Additionally, it is used as a fail-over communication channel in case of a failure of the main clearing system. The system generally follows the service oriented architecture scheme providing both www and web services interfaces to its functionality.

The overall structure of the proposed integrated model of software architecture has been presented in fig. 2. It comprises the following tiers:

– **Architectural Policy**: comprises a set of clauses that translate the enterprise strategy and relevant policies to a set of principles shaping the architecture of corporate IT systems. Architectural Policy is represented as a set of clauses being simply text records consisting of the following fields (similar to the architecture principles of TOGAF [6]): a.p. policy rule name, a.p. policy rule, a.p. rule explanation. As Architectural Policy reflects

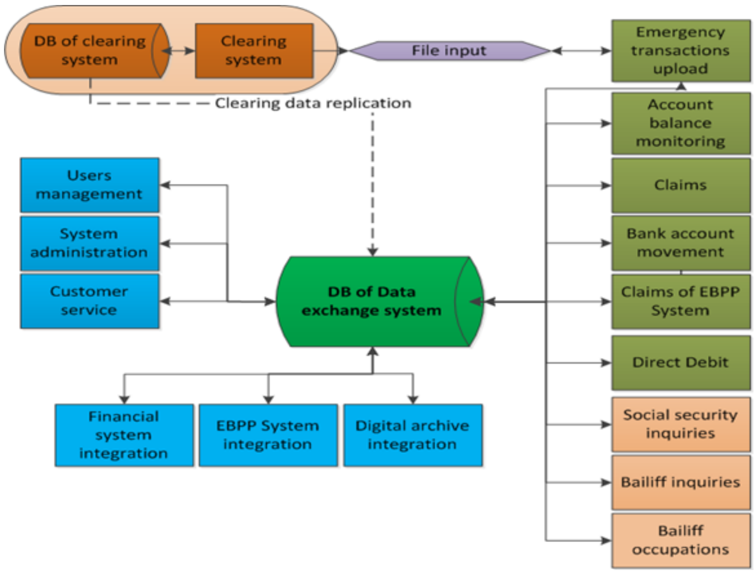

Figure 1. Banking data and document exchange system

enterprise strategy and relevant policies in the architectural categories it can provide criteria for the assessment of the alignment between system architecture and business strategy or enterprise policies.
Our example: if the corporate strategy says that the mission of the company is to provide fully secure services to banking sector, it can be translated into a number of architectural policy clauses – e.g. "all in-coming and out-going data is securely transferred", "all the processing of clients data can be back-traced", "all the users have to be securely authorised before accessing its services". These rules can be used to verify/assess the rules of system organisation pattern (in our example: the first architecture policy clause applies to the rules concerning communication framework selection and data input organisation (see below).

– **System Organisation Pattern**: is set of architectural decisions (called System Organisation Pattern Rules) representing core concepts organising system (this extends the idea presented in [15]) at different levels of scope [14]: enterprise, domain, system/application, component. These basic decisions include but are not limited to:
  – *Decomposition into set of domains/subsystems/applications*: defines organisation of system functionality – from conceptual or business (domain) to technical level (applications/subsystems).
  Our example: the following domains have been defined: Human Resources (HR) Management, Finance Management, Clearing Systems, Digital Signature and Auxiliary Services. The „Data exchange system" belongs to Clearing Systems Domain. It has been divided into 15 subsystems/modules (comp fig. 1)
  – *Geographical and organizational allocation of system entities*: defines both the deployment of system entities in terms of organization's geography and organization structure.
  Our example: The components of „Data exchange system" are supposed to be deployed in central company's data centre, client applications will be provided throughout the company and its clients.
  – *Organisation of data input*: indicates where and how the data is fed into the system.
  Our example: The data will enter/leave the system via central interfaces only: WWW interface, web service interface and file interface.

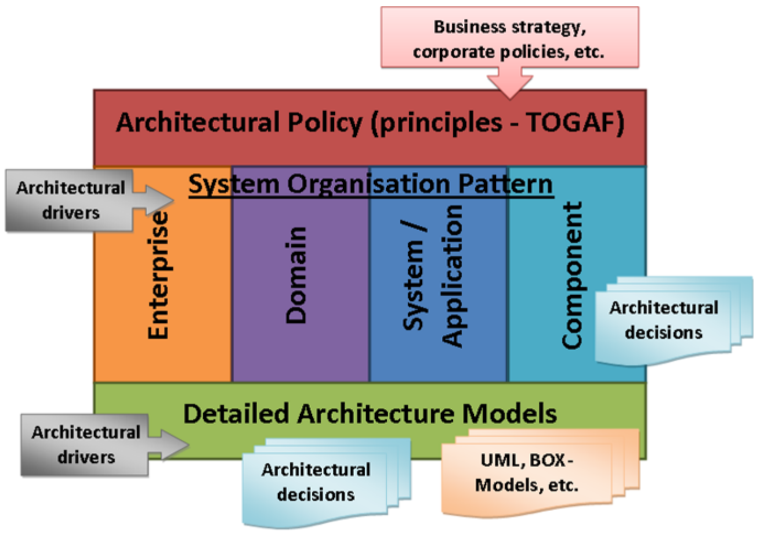

Figure 2. The overall structure of the integrated architecture model for large scale software systems

- *Data storage distribution*: defines how permanent data is distributed among databases;
  Our example: Lack of distributed data (all of data are allocated in one central database) eliminates the need for distributed data storage management and allows for short transactions only.
- *Distributed data storage management*: defines the means of database synchronization, data transmission between remote locations etc.
- *Transaction management*: concerns the selection of mechanisms and/or solutions that are used to ensure transactional processing.
- *Communication framework*: defines communication mechanisms between equivalent system entities (e.g. queuing solutions).
  Our example: subsystems exchange data only via database; clients communicate with the system via SSL channel.
- *Core pattern/style selection*: the selection of design patterns or architectural styles that define the overall structure of a given architecture entity.
  Our example: all the subsystems are supposed to follow three-tier architecture.

These decisions are supposed to be represented as text records as in [9, 16]. Other models can be employed here as they emerge.

- **Detailed Architecture Models**: represent details of system architecture. Obviously the suite of models used at this level has to be tailored to the class and properties of the modelled system. We plan to develop such a model suite for SOA systems as an integral part of our research and a mean of validation of our concepts. Obviously, other suites should also be developed for other domains or classes of systems.

The detailed structure of the contents of the integrated architecture model has been presented in fig. 3. The following properties could be observed from the model presented in fig. 3:

- Rules of system organisation pattern can be related to a number of architectural policy clauses, which should support the assessment of strategic alignment of system architecture;
- A number of rules of system organisation pattern can be applied to a given detailed model, sometimes detailed model can be used to illustrate a given rule;
- System organisation pattern rules belong to one of the levels of scope and describe architectural entities defined at this level;

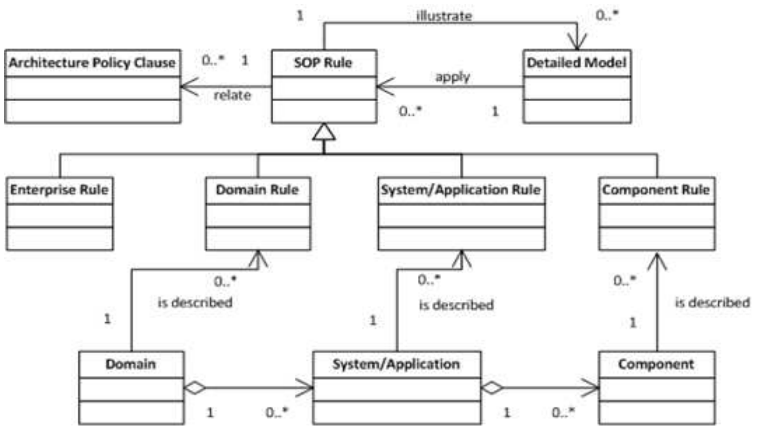

Figure 3. Class diagram illustrating dependencies between model entities

- Association between system organisation pattern rules and detailed models indicates models of a given system entity.

The dependencies that may exist between system organisation pattern rules have not been shown to make the class diagram more legible, however, enterprise level rules, can be applicable to some lower level decisions, especially domain ones, the latter ones to the system/application rules etc. etc. These dependencies can be used to verify whether lower level models are compliant with higher level ones.

## 5. Summary. Further Research Prospects

The integrated system architecture model presented in section 4 fulfils most of the premises enumerated in section 3. Architectural decisions model only core architectural concepts at a given levels of scope (rule 1, 3, 4), relations established between them and detailed models provide for the indexing of detailed models (rule 5). Alignment with business strategy (rule 7) is ensured with Architectural policy being an integral component of the model.

Further research is needed to:

- Define model suites for different classes of systems and their stakeholders (rule 2) – our research is heading towards identification of such a suite for SOA systems – these are supposed to include at least: BPMN as business process models (at different levels of detail) and LOTOS as a formal model of concurrent processing in Business Processes providing for extended verification capability;
- Enhance integrated model with mechanisms supporting system evolution (rule 6);
- Although model is structurally ready for the assessment of one model against another one, effective analysis/assessment techniques can be developed when full suite of models is defined as well as verified properties are know. This can only be done in the context of a certain genre of software-intensive systems as e.g. service-oriented systems. (rule 8).
- Develop tool support the integrated architecture model.

## Acknowledgment

This work was sponsored by the Polish Ministry of Science and Higher Education under grant number 5321/B/T02/2010/39.

## References

[1] *Recommended Practice for Architectural Description of Software-Intensive Systems*, http://standards.ieee.org/findstds/standard/1471-2000.html, IEEE Std., 2000.

[2] P. Kruchten, "The 4+1 view model of architecture," *IEEE Software*, Vol. Volume 12, No. 6, 1995, pp. 45–50.

[3] R. C. Philippe Kruchten and J. C. Dueñas, "The decision view's role in software architecture prac-

tice," *IEEE Software*, Vol. Volume 26, No. 2, Mar/Apr 2009, pp. 36–42.

[4] J. Zachman, "Framework for information systems architecture," *IBM Systems Journal*, Vol. Volume 26, No. 3, 1987, pp. 276–292.

[5] ——. (2010) Zachman framework. http://www.address.org/.

[6] The Open Group Architecture Framework (TOGAF). http://www.zifa.com. The Open Group.

[7] A. W. Kamal and P. Avgeriou, "An evaluation of ADLs on modeling patterns for software architecture," *LNCS Springer*, Nov 2007, 3rd International Workshop on Rapid Integration of Software Engineering techniques (RISE).

[8] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," *IEEE Computer Society*, 2005, pp. 109–120, 5thWorking IEEE/IFIP Conference on Software Architecture (WICSA'05).

[9] J. Tyree and A. Akerman, "Architecture decisions: Demystifying architecture," *IEEE Software*, Vol. Volume 22, No. 2, Mar 2005, pp. 19–27.

[10] A. Zalewski and S. Kijas, "Architecture decision-making in support of complexity control," *LNCS Springer*, Vol. Volume 6285, 2010, pp. 501–504.

[11] F. N. Rafael Capilla and J. C. Dueñas, "Modeling and documenting the evolution of architectural design decisions," *IEEE CS Press*, 2007, proc. 2nd Workshop Sharing and Reusing Architectural Knowledge Architecture, Rationale and Design Intent.

[12] P. Kruchten, "An ontology of architectural design decisions," *Rijksuniversiteit Groningen*, Oct 2004, pp. 54–61, in 2nd Groningen Workshop on Software Variability Management.

[13] ArchiMate® 1.0 specification. http://www.opengroup.org/archimate/doc/ts_archimate/.

[14] R. Malan and D. Bredemeyer, "Less is more with minimalist architecture," *IEEE's IT Professional*, Sep/Oct 2002.

[15] A. Zalewski, "Beyond ATAM: Architecture analysis in the development of large scale software systems," *LNCS Springer*, Vol. Volume 4758, Sep 2007, pp. 92–105, first European Conference, ECSA 2007 Aranjuez, Spain.

[16] P. A. Neil B. Harrison and U. Zdun, "Using patterns to capture architectural decisions," *IEEE Software*, Vol. Volume 24, No. 4, Jul/Aug 2007, pp. 38–45.