Explaining Architectural Choices to Non-architects

Diego Bernini and Francesco Tisato

D.I.S.Co., University of Milano-Bicocca, Viale Sarca, 336
20126 Milano, Italy
{bernini,tisato}@disco.unimib.it

Abstract. Explaining and motivating architectural choices are crucial points both in real system development and in computer scientists education. Stakeholders and students should fully understand from a high level perspective the rationale behind basic architectural choices. The paper proposes a communication approach that is complementary to established design processes and can be exploited in workshops that involve the "non-architects" at the end of each phase of an iterative development process. Starting from a problem analysis focused on the significant aspects of data, activities and information flows, a logical architecture is defined by grouping activities into logical components. Different logical architectures are rated according to several conceptual dimensions, in order to highlight their specific rationale and benefits. Finally, deployment solutions are considered to weight the ratings according to costs and constraints of different deployment architectures and of the underlying technologies.

Keywords: requirements analysis; architectural design; components; architect-ture teaching.

1 Introduction

Many methods and approaches have been proposed to drive the architectural design [1]: among them Rational Unified Process methods [2] and Kruchten's 4+1 views [3], the Siemens Four Views model [4] and the Architecture Tradeoff Analysis Method [5]. Most methods rely on an iterative approach and highlight the relevance of frequent workshops [6] to assess the achievements of each iteration and to draw the guidelines for the next ones. Workshops involve "non-architects", be they stakeholders in a business context or undergraduate students in an educational context; they play a crucial role but are often flooded with trendy buzzwords including, during the last decades, client-server, three-tier, grid, SOA, cloud computing and so on. Though these terms denote significant technological opportunities, they are often misused and presented as silver bullets in a marketing perspective.

The risk is the premature elaboration [6] of key architectural aspects (e.g., distribution issues) that are improperly biased by scarcely motivated technological choices and do not rely on a clear understanding and assurance that the architecture meets the business needs [7]. This risk is especially high at early stages of the process, (e.g., at the end of the inception and of the initial elaboration phases in RUP), both because key choices about coarse-grained architectural aspects can be hardly modified later and because non-architects playing strategic roles are involved in these stages.

The aim of the paper is to suggest a *communication* process that is complementary to the overall design process and can support the explanation of architectural choices to non-architects when they are involved in critical decisions. The explanation should present in understandable and linear way the rationale of the choices, not the history of the underpinning process; this can be subsumed by the sentence "fountain process, waterfall explanation".

First, the *problem architecture* is introduced. It includes, as one can expect, actors, use cases and domain model. It provides insights about the conceptual activities the system must perform by sketching the major information flows in a Data Flow Diagram style [8]. The problem architecture also summarizes the major Non-Functional Requirements (NFRs), whereas it carefully excludes technological issues.

Then the *logical architecture* is defined by grouping activities into *logical components* according to two well-established criteria, Low Coupling and High Cohesion [9]. Coupling and cohesion are rated according to dimensions that correspond not only to functional, but mainly to non-functional requirements. The ratings are synthesized by Kiviat charts allowing alternative architectural solutions to be roughly compared "at-a-glance". Technological issues are still kept out of scope.

Finally the *deployment architecture* shows how the components of a logical architecture can be deployed into a distributed system. Technological platforms enter the scene. The ratings are weighted by considering costs and criticalities in different deployment scenarios.

Section 2 introduces a simple case study and the problem architecture. Section 3 presents two logical architectures and discusses how they can be compared. Section 4 sketches some deployment architectures and exemplifies the impact of technological constraints. Finally, Section 5 highlights lessons learned from the application of the approach.

2 Problem Architecture

A simple example will be used as reference in the following. A shop chain has several shops and one central warehouse. Each shop has a local warehouse. The goal is to manage the demand chain. Sold goods are recognized at each POS via RFID (or via code bar or keyboard). Product stocks are managed at three levels: shelf (to notify an operator about the need for replenishment), local shop inventory (to require the delivery of goods from central to local warehouse) and global inventory (to plan purchases or production).

The aim of the Problem Architecture is to communicate the key elements of the problem by answering a few basic questions: *Who* actors are, *Where* they are located, *What* the interesting information is, *How* it is generated and processed, *When* activities have to be performed and *Why* they are triggered.

Every question leads to the identification of properties of information and activities on the basis of both functional and non-functional requirements. Properties correspond to *problem dimensions* that will provide a conceptual framework for the comparison of alternative logical architectures, as we shall discuss later.

Answering the *"Who" and "Where"* questions means to identify actors and where they are physically located. In the example, there are four actors: POS, located at the shop; Operator, mobile inside the shop; Shop manager, located at the shop and Purchase manager, located at the central site.

"What" we are talking about is answered by a domain model in terms of class diagrams. In the example the basic concept is that of good, characterized by an identifier and by a number of items (sold or on a shelf or in the shop warehouse or in the central warehouse). This can be modeled in a straightforward way by an abstract Goods class with several subclasses corresponding to different views of the general concept of Goods. Subclasses may exhibit significant NFRs: for example, the required precision, which is different for different subclasses.

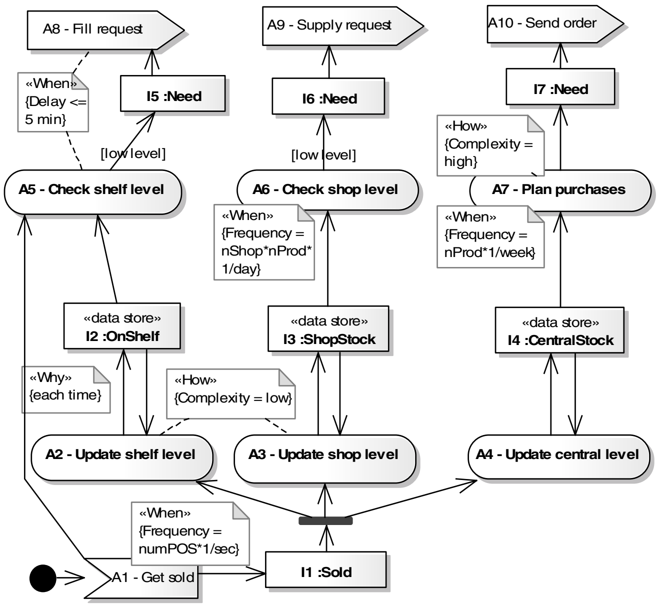


Fig. 1. "How", "When" and "Why"

"How" information are generated and flow among computational activities is sketched by the data flows in the activity diagram of Fig. 1. A relevant intrinsic property of the activities is their computational complexity. Adornments dealing with "When" and "Why" issues would be better presented in separate diagrams to highlight that "How" just defines necessary conditions (i.e., the availability of information) for the execution of the activities. Defining *"When"* they are performed implies to identify frequencies and timing constraints. Finally, answering the *"Why"* question implies to identify those, and only those, control constraints that are explicitly stated by the specifications.

3 Logical Architectures

The next step is to describe the *Logical Architecture*, i.e. how to group activities into *logical components*. A logical component conceptually identifies a coarse-grained software entity which encapsulates computations and state in a self-contained whole that can be utilized through well-defined interfaces [10][11].

Activities can be grouped into logical components in many ways. Architects often choose a dominant dimension as driver and check the resulting architecture against other dimensions. Choosing different dimensions as drivers may lead to dramatically different logical architectures. For example, Fig. 2 a) sketches a grouping driven by the “What” question i.e., the goods a component manages. Fig. 2 b) sketches a grouping driven by the “Where” question i.e., the location of devices and actors.

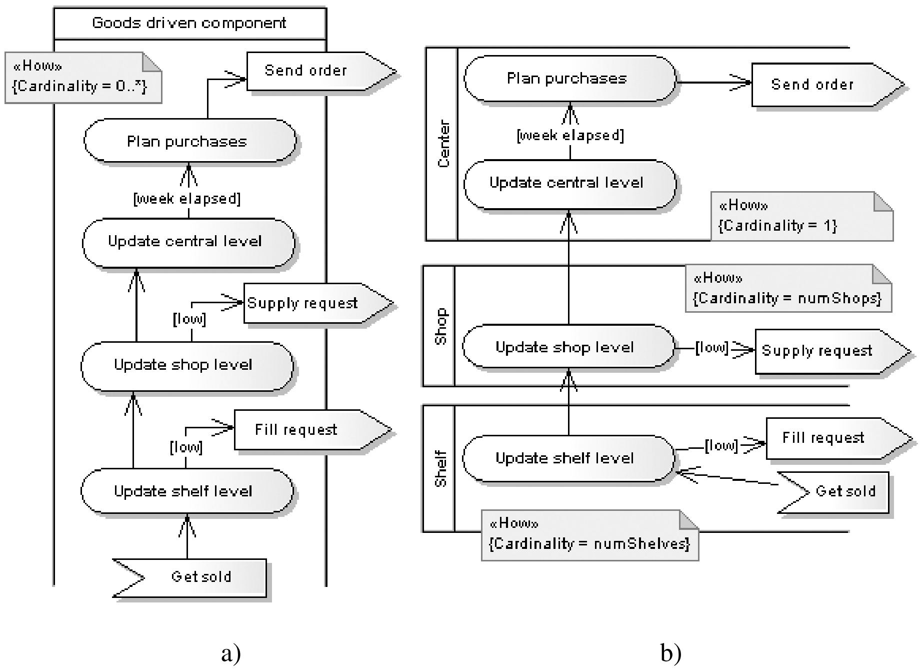


Fig. 2. a) “Goods-driven” grouping; b) “Location-driven” grouping

<<How>> adornments deal with the multiplicity of component instances. The assumption is that the Goods-driven component is dynamically instantiated for each sold item, whereas the Location-driven components are statically instantiated according to the locations they deal with; of course, they have a cyclic behavior, not shown for simplicity.

The presented solutions are naive and extreme, but they are useful to stress how the choice of a driving dimension influences the logical architecture. The solutions are named “Goods-driven” and “Location-driven” respectively to help non-architects identifying the driving problem dimensions, though the solutions correspond to well-known composition criteria (e.g., functional grouping and user/device oriented grouping [12]).

The question is: how to compare the effectiveness of different logical architectures by considering *all* the problem dimensions? Key criteria are *low coupling* and *high cohesion* [9], which should be evaluated over all the dimensions of the problem, including NFRs. This can be done via Kiviat charts (see Fig. 3) where axes correspond to problem dimensions. The "What" issue is refined to consider how many data types are exploited by a component (Types), how may object instances are managed by a component instance (Instances), data flows across components (Flows) and data sharing among components (Sharing). The effectiveness on each dimension is rated from 1 to 10 by assigning crisp "rule-of-thumb" values; this may suffice in a workshop discussion aimed at providing a broad comparison of different solutions.

The footprint of a chart provides a rough but impressive "at-a-glance" feeling about the effectiveness of different logical architectures: wider footprints correspond to better solutions. The charts provide a reference for discussing with non-architects the rationale behind the ratings, as summarized in the rest of this section.

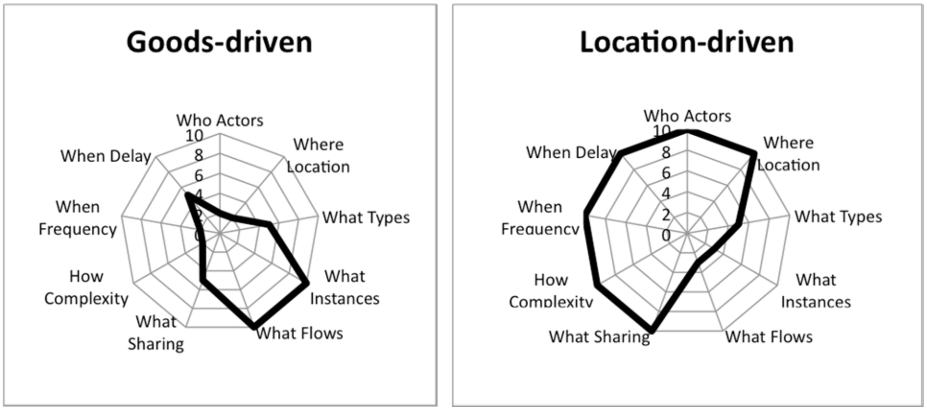


Fig. 3. Ratings for Goods-driven and Location-driven logical architectures

The Goods-driven architecture (see Fig. 3) is poor in terms of cohesion. The component interacts with all the actors ("Who") wherever they are located ("Where"). It also deals with all the Goods types; in this simple example there are few subclasses of Goods that perform similar functions, so that this dimension gets a medium rating, though in more complex situations this aspect might be more critical. The cohesion is also poor regarding "How" and, in particular, the complexity, because the component intermixes very simple and computationally intensive activities (Plan purchases). Finally, the component exhibits scarce cohesion in the "When" dimension, regarding both the frequency of the activities it includes and the timing constraints.

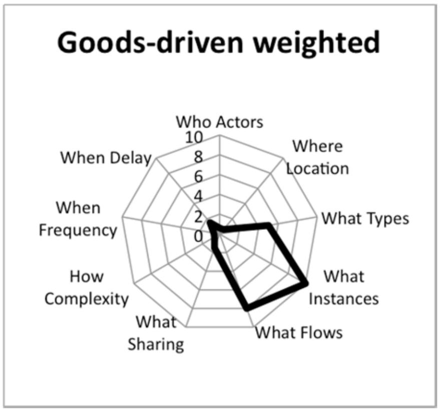
Things are not so bad when looking at coupling. There is only one component, which obviously is fully uncoupled if its static structure is considered. However, the component is multi-instantiated; therefore the coupling among different instances must be rated by considering the dynamic behavior. Each instance of the component manages an individual Goods item, so that different instances do not explicitly communicate and are fully uncoupled in the "What-Flows" dimension. They sometimes interact over the data stores, so that there is a moderate coupling in the "What-Sharing" dimension.

The Location-driven architecture can be rated in a similar way (see Fig. 3). Just note that component manages several instances of Goods ("What-Instances"). There are several information flows among components ("What-Flows"). Information sharing ("What-Sharing") is negligible because the data stores exploited by the components contain disjoint information.

The comparison of the two charts in Fig. 3 shows that the ratings of the two logical architectures are somehow complementary. The Location-driven one looks better, but this preliminary conclusion will be refined in the following.

4 Deployment

At last platforms and technologies enter the stage. The *deployment architecture* defines how logical components can be deployed into computational nodes. A natural deployment for the Goods-driven architecture is to rely on a centralized server farm, where instances of the unique logical component can be dynamically created each time items are sold. On the opposite, a natural deployment for the Location-driven architecture is to rely on a distributed infrastructure where computational nodes are associated with shelves, shops and central site. Again, these are naive solutions, presented here to exemplify how technological issues can be exploited to tune the ratings deriving from the logical architecture.



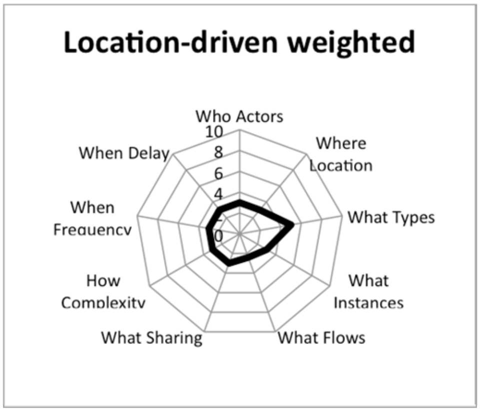


Fig. 4. Weighted ratings for Goods-driven and Location-driven logical architectures

The straightforward idea is that the rating of a logical architecture on each dimension must be weighted according to the advantages it produces in a specific deployment scenario. For example, if a low-cost and reliable broadband network is available, computing power is not a problem and an efficient DBMS is available, the ratings of the cohesion on the "Where", "How-Complexity", "What-sharing" and "When" dimensions get a low weight (say 0.3). On the opposite, the presence of complex inter-component data flows may imply high development and management costs; therefore the rating of the “What-Flows” gets a high weight (say 0.8). The result is shown in Fig. 4, highlighting that the Goods-driven logical architecture might be more cost-effective than the Location-driven one. Of course, the result can be very different under different technological assumptions, for example if connectivity problems are foreseen.

5 Lessons Learned and Conclusions

The proposed approach focuses on “how to communicate to non-architects the criteria underlying architectural choices”. The idea is that the communication process cannot mirror the development process, because the explanation must follow a waterfall pattern even if the real development process is iterative. Therefore the approach should be viewed as complementary, not alternative to established design and development processes.

The approach borrows some central ideas from the Model Driven Architecture (MDA) approach [13]. Moreover it strong related with the Use-Case driven architecture design [14]. The Krutchen 4+1 views [3] and the Rational Unified Process [2] pay particular attention to the identification of use cases, business and problem analysis to validate the final architecture. The term “Logical Architecture” is used there to identify the functionalities that the system has to provide. However, our approach is more focused on how the component organization can be conceptually defined and motivated in term of clusters of functionalities and properties. Similar remarks apply to the Conceptual Architecture view proposed by [4].

The approach stems from experiences in real-life projects and, in particular, from the participation in project reviews involving high-level stakeholders which “want to understand” (and to decide) in half an hour and are often biased by up-to-date buzzwords. The problem here is to focus on key issues and to avoid discussions shifting from vague philosophical principles to technological tricks. The separation between problem, logical and deployment architectures might seem pedantic, but it helps enforcing the attitude towards abstraction and separation of concerns, without neglecting technical aspects and constraints.

The approach has been also successfully tested in several introductory courses on software architecture. What the authors learned is that presenting to students one or more systems is reasonably easy; what is difficult is to communicate the rationale underlying the choices and the conceptual process that led to a specific architecture. Not surprisingly, students with an “algorithm-oriented” culture started with solutions like the “goods-driven” one, whereas students with a “web-oriented” culture started with the “location-driven” one. The comparison of the solutions by relying on the impressive, though naive, presentation of the footprints, together with the “what-if” discussion of what happens if different dimensions have different weights, have been the basis for highly effective classroom discussions.

Ultimately, our goal was to support fruitful discussions, to raise doubts, to stimulate a critical attitude and to warn against the unconscious adoption of a-priori solutions.

Further research will deal with a more precise formalization of the communication process and, in particular, on metrics supporting the rating of the architectures over the problem dimensions.

References

1. Wieringa, R.: A survey of structured and object-oriented software specification methods and techniques. ACM Comput. Surv. 30, 459–527 (1998)
2. Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley Professional, Reading (2000)
3. Kruchten, P.: The 4+1 View Model of Architecture. IEEE Softw. 12, 42–50 (1995)
4. Hofmeister, C., Nord, R., Soni, D.: Applied Software Architecture. Addison-Wesley Professional, Reading (1999)
5. Kazman, R., Barbacci, M., Klein, M., Carrière, S.J., Woods, S.G.: Experience with performing architecture tradeoff analysis. In: Proceedings of the 21st international conference on Software engineering, pp. 54–63. ACM, Los Angeles (1999)
6. Larman, C.: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd edn. Prentice Hall PTR, Englewood Cliffs (2004)
7. Tyree, J., Akerman, A.: Architecture Decisions: Demystifying Architecture. IEEE Softw. 22, 19–27 (2005)
8. DeMarco, T.: Structured Analysis and System Specification. Prentice Hall PTR, Englewood Cliffs (1979)
9. Stevens, W.P., Myers, G.J., Constantine, L.L.: Structured design. IBM Systems Journal 13, 115–139 (1974)
10. Shaw, M., Garlan, D.: Software architecture: perspectives on an emerging discipline. Prentice-Hall, Inc., Englewood Cliffs (1996)
11. Heineman, G.T., cur Councill, W.T.: Component-based software engineering: putting the pieces together. Addison-Wesley Longman Publishing Co., Inc., Boston (2001)
12. Wieringa, R.J.: Design Methods for Reactive Systems: Yourdon, Statemate, and the UML. Morgan Kaufmann, San Francisco (2003)
13. Mellor, S.J., Kendall, S., Uhl, A., Weise, D.: MDA Distilled. Addison Wesley Longman Publishing Co., Inc., Amsterdam (2004)
14. Tekinerdogan, B., sit, M.A.: Classifying and Evaluating Architecture Design Methods. In: sit, M.A. (cur.) Software Architecture and Component Technology, pp. 3–28. Kluwer Academic Publishers, Dordrecht (2001)