

Chapter 2

Software Architecture Today

There is no all-explaining and globally valid definition for either software architecture or system architecture. Many experts have their own approach and way to define their field of action. But all of them have at least one thing in common: They all think experience is a key element for building software and/or system architectures.

In [Sect. 2.1](#) some possible definitions of software architecture and the ideas behind them are pointed out. In [Sect. 2.2](#) definitions of system architectures are defined and analyzed. After defining those two areas nowadays common approaches to find architectures for software and computer systems are shown in [Sect. 2.3](#).

2.1 What is Software Architecture?

Shaw and David define software architecture the following way:

Structural issues include the organization of a system as a composition of components; global control structures; the protocols for communication, synchronization, and data access; the assignment of functionality to design elements; the composition of design elements; physical distribution; scaling and performance, dimensions of evolution; end selection among design alternatives. This is the software architecture level of design.¹

More abstract:

The architecture of a software system defines that system in terms of computational components and interactions among those components.²

¹ Shaw and David 1996, p. 1.

² Shaw and David 1996, p. 3.

In comparison to the definitions mentioned above Vogel has a more general approach to define architecture for software. In his view architectures of any kind always strongly include social and organizational emphases. So it is not only a pure technical discipline. Also an architecture cannot bear any details of the system or software itself. Its intention has to be building fundamental columns on which in a later process of constructing the software the details can be evolved. It has to provide a manageable view over the complexity of the problem.³

Die Software-Architektur eines Systems beschreibt dessen Software-Struktur respektive dessen – Strukturen, dessen Software-Bausteine sowie deren sichtbaren Eigenschaften und Beziehungen zueinander.⁴

Another definition is provided by Bass:

The software architecture of a program or computing system is the structure of structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.⁵

Bass sees software architecture as a result of an ongoing cycle of technical, business and social influences. In turn software architecture also affects the corresponding environments. They call this during a design process existing situation the architecture business cycle (ABC). In this cycle Bass sees the architect and its product to be influenced by a variety of stakeholders, the technical environment and his own experience. Stakeholder would be customers, end users, the developing organization's management, marketing and the maintenance organization. Each of them has other priorities to what the architecture has to feature. For example the customer who has to pay for the product would like to be the costs as low as possible. He'd even accept a loss of usability to lower the costs. This in turn is the opposite of what the end user would like the product to be. He who actually uses the resulting program wants to work with a product which provides a high level of usability.⁶

2.2 What is System Architecture?

Vogel provides a definition for a system in a general aspect:

Ein System ist eine Einheit, die aus miteinander interagierenden Software- und Hardware-Bausteinen besteht sowie zur Erfüllung eines fachlichen Ziels existiert. Es kommuniziert zur Erreichung seines Ziels mit seiner Umwelt und muss den durch die Umwelt vorgegebenen Rahmenbedingungen Rechnung tragen.⁷

³ Cf. Vogel et al. 2005.

⁴ Vogel et al. 2005, S. 46.

⁵ Bass and et al. 1998.

⁶ Cf. Bass et al. 1998, pp. 6–10.

⁷ Vogel et al. 2005, S. 44.

Consequently for Vogel a system is more than just components of software. Software architecture is therefore just a part of the architecture of a whole system. However, they both are connected very tight and therefore affect each other. The system is embedded in an organizational environment with which it interacts.⁸

2.3 Today's Approach

In Vogel's⁹ opinion architectural awareness is what makes an architect either successful or not. The architect has to decide a lot of decisions during the process of design. However, the fundamentals of those decisions are very dynamical. New demands and specification expressed by the stakeholders, arriving of new technology, or even new experiences of the architect himself can influence the making of decisions and so also the development of the architecture.

2.3.1 *The Proceeding*

Further on Vogel states in order to prevent any unpredictable events the architectural awareness should work like a letter case. With its help the architect can deposit experiences and acquired knowledge into mental boxes and also fetch them whenever he needs them. This letter case is spanning a regulation framework within which the architect creates and develops system architecture.

Within this regulation framework Vogel defines six main dimensions¹⁰:

- What – architectures and disciplines in architecture
- Where – perspectives on architecture
- Why – specifications for architecture
- Wherewith – instruments in architecture
- Who – organizations and individuals
- How – approach to architecture

Those dimensions provide the rough fundamentals for the letter case. They assure a meaningful breakup between each other so that they can be expanded independently and still they are extensive enough so that all different approaches to architecture can be conceptualized. With the help of such a regulation framework, in Vogel's opinion, the architect can attend to fundamental questions and therefore systematically orientate himself in practical cases.

⁸ Vogel et al. 2005, S. 44–45.

⁹ Cf. Vogel et al. 2005, S. 24–30.

¹⁰ Cf. Vogel et al. 2005, S. 26–27.

The following section presents the fundamental dimensions of the regulation framework how Vogel defines them¹¹:

– What

The architect should be capable of defining, describing and comparing architecture to other architectures even of different subjects and/or professions. Furthermore due to the high requirements of software architecture distinct disciplines of architecture established themselves. Vogel lists the following disciplines: software architecture, data architecture, architecture of integration, network architecture, security architecture, system management architecture, enterprise architecture. The architect often has to decide which kind of architecture he wants to realize.

– Where

Architectures of any kind are very complex and often unmanageable as one piece. Therefore the architect needs to reduce its complexity by only examine and handle manageable parts of the ensemble. To do so he works on so called architecture layers or levels always knowing what kind of connection the layer has to and how the layer fits to the rest of the architecture. On different layers the architect can use different instruments. He can even adopt different views to the layer. Views should lower the complexity for the architect and should make a systematically approach to the architecture possible.

– Why

Software architectures have numerous specifications to fulfill. Those specifications in turn can be classified by their kinds. For Vogel such classes are: Specifications by an organization, by a system, by a component, by the time of development, by the runtime and organizational conditions. An architect needs to be aware of the different kinds of specifications in order to design a goal-oriented IT-system.

– Wherewith

Every architect gets to know a huge variety of instruments which can be used to create architecture during his professional life. He keeps a classification system for all these instruments regarding their relevance to the architect in mind. Such instruments can range from conceptual principles to concrete technologies. Principles are approved instruments of architectural design. Fundamental concepts use such principles to enable the architect to use them in his architecture. Architectural styles and patterns, again, rely on such concepts and principles. Styles and patterns are successful and approved solutions for a thriving architecture. Patterns often also are used to document the structure of architecture. When a specific solution for an architecture was outstanding the architect can use the whole

¹¹ Cf. Vogel et al. 2005, S. 30–38.

structure of it as a reference to which he can rely in the future. Because it is important to not only create architecture but also vulgarize it to all involved stakeholders, developers and other individuals the architect also needs a collection of documentary instruments. Moreover, the practice of creating architecture for software and IT-systems generated specific architectural structures which provide basic technologies to build such approved structures. And last but not least the architect needs to be aware of updates of such instruments to in turn update his collection so that he can provide the most proven solution for his clients.

– Who

As Vogel mentioned before an architect needs to vulgarize all the working steps, the proceeding and the architecture itself to an amount of individuals. In order to do so most effectually and successfully he needs to possess expertise in social contact.

– How

The goal of an architectural design is to provide a fundamental structure onto which a system can be build. The architect can access a collection of instruments, interchange information with others and take different views on to his work in order to reach the optimal architecture to fulfill all the requirements and specifications. In Vogel's opinion, for an architect to distinguish himself as a successful architect he needs to be capable of working systematically. So he needs to work iteratively through the procedure model provided by Vogel: creating a business case, understanding the specifications, designing the architecture, vulgarizing the architecture, implementing the architecture.

The whole process of development is done through an iterative-incremental process. In those iterations the architect's tasks consist of a combination of above mentioned five actions. Whereas in the beginning of the process the tasks envelop rather the actions of creating a business case and understanding the specification and in the further preceding of the process the tasks consist of communicating the architecture and implementing it.¹²

The following will show the architect's role during the elaboration of the main actions¹³:

- Creating a business case: During this action the characterization of the task and the goal takes place. The architect has the role of a technical adviser to ensure the technical feasibility.
- Understanding the specifications: The architect finds and analyzes the specification and tries to solve inconsistent requirements.
- Designing the architecture: This is the actual task where the architect designs the architecture. To do so he has a variety of instruments as mentioned above. Subsequent those instruments get described more detailed.

¹² Cf. Vogel et al. 2005, S. 274.

¹³ Cf. Vogel et al. 2005, S. 275.

- **Vulgarizing the architecture:** It is of great interest to the architect that all the stakeholders have a good understanding of the architecture. It influences their further work and ability of success. So in this task the architect makes sure that all stakeholders understand the architecture.
- **Implementing the architecture:** In order to make sure the implementation of the system is conforming to the architecture the architect verifies this during this task.

So far the today's approach was roughly outlined. In the next step the architect's instruments get described more detailed.

It was mentioned above that an architect makes use of a variety of instruments. With the help of such resources he designs, concepts, communicates and documents his architectures. Those are¹⁴:

- **Architectural principles:** There are a big number of architectural principles, such as modularity, information hiding, abstraction, separation of concerns, etc., which have proven themselves to be useful and constructive. It doesn't depend on the use of such principles whether the out coming architecture is a good or a bad one. But the lack of fundamental principles within an architecture is a strong sign that it can be a weak model of a specific architecture. There are possibilities that one principle is conflicting with another and thus the two can't be combined in a single architecture but this should always be deliberate and the consciously omitting of an appropriate principle should always be well documented.
- **Architectural concepts:** There are fundamental concepts for how to build an architecture which the architect can access. Those are widely used and highly approved. Such concepts would be procedural approaches, object orientation, component orientation, etc.
- **Architectural styles:** A less abstract instrument for the architect would be architectural styles. Such styles developed from the experience that they have already repeatedly been useful to solve a specific problem in an architecture. They are solutions to recurrent occurring and well known problems. The architectural style of a system describes the founding structure of the system itself and allows categorizing it and comparing it to other systems. Such styles¹⁵ would be pipes and filters, hierarchical layers, rule-based systems, event systems, etc.
- **Architectural patterns:** Patterns are like styles a more concrete support for the architect. They again are a solution possibility for a series of specific problems. Moreover, patterns provide a concrete method of resolution to solve a precise problem. Such patterns are proxies, broker, etc.
- **Architecture as reference:** When an architect uses a whole architecture as a reference he has got a complete solution for a certain problem area. Such a solution combines general architectural knowledge and expertise with specific requirements. The architect can use such a reference whenever he encounters an architectural problem which has been solved similar before.

¹⁴ Cf. Vogel et al. 2005, S. 112–194.

¹⁵ Cf. Shaw and David 1996, p. 20.

2.3.2 *Instruments for Documentation*

So far actions that are taken by the architects have been listed. However, it is not the only problem for them to actually create an architecture but also to write it down and document it. The right choice of instrument to document the built architecture is very important to communicate it to all the stakeholders. It is a great influence to how easy the various stakeholders can understand the architecture. Different instruments can provide a different sort of view onto the architecture which in turn can be more practical in communicating the architecture to a specific stakeholder. That is the end user who actually has to work with the system at the end has a different approach to the whole system than a programmer who will implement the system. Different views provide different types of information about the architecture. So the architecture needs to be presented to each stakeholder as clear and as understandable as possible in order to let them be as most constructive as possible. To do so the architects have a variety of documentation instruments. The following will shortly present the instruments UML (Unified Modeling Language), ADL (Architecture Description Language), DSL (Domain Specific Language) and Acme (an ADL) (Fig. 2.1).

- UML¹⁶: The Unified Modeling Language is the result of the merging of a lot of sublanguages. In its evolution it combines and adapts more and more other languages to result as a single valid one. Today UML is available as version 2.1.1¹⁷ and still evolving. The positive part about UML is that it provides a lot of different views onto one architecture at the same time. But because it is a relative young standard it does not include all kinds of components so that an architect may use another element to substitute the missing one. To provide those views UML comes with a variety of different diagrams. Vogel contrasts the different view onto an architecture with the accordingly UML diagrams.¹⁸
- ADL¹⁹: In Architecture Description Languages a component features a very central role. ADLs approve the precise design of architectures before they are implemented. That makes it possible for the architect to determine whether the architecture has any errors or problems and whether it complies with the requirements which should from it. Moreover, the attributes of the architecture can be analyzed. So an ADL has a good qualification to design, analyze and simulate an architecture before its implementation as a system. Additionally they feature the following²⁰:

¹⁶ Cf. Vogel et al. 2005, S. 197.

¹⁷ OMG – Object Management Group.

¹⁸ Vogel et al. 2005, S. 199–200.

¹⁹ Cf. Vogel et al. 2005, S. 201.

²⁰ Vogel et al. 2005, S. 202.

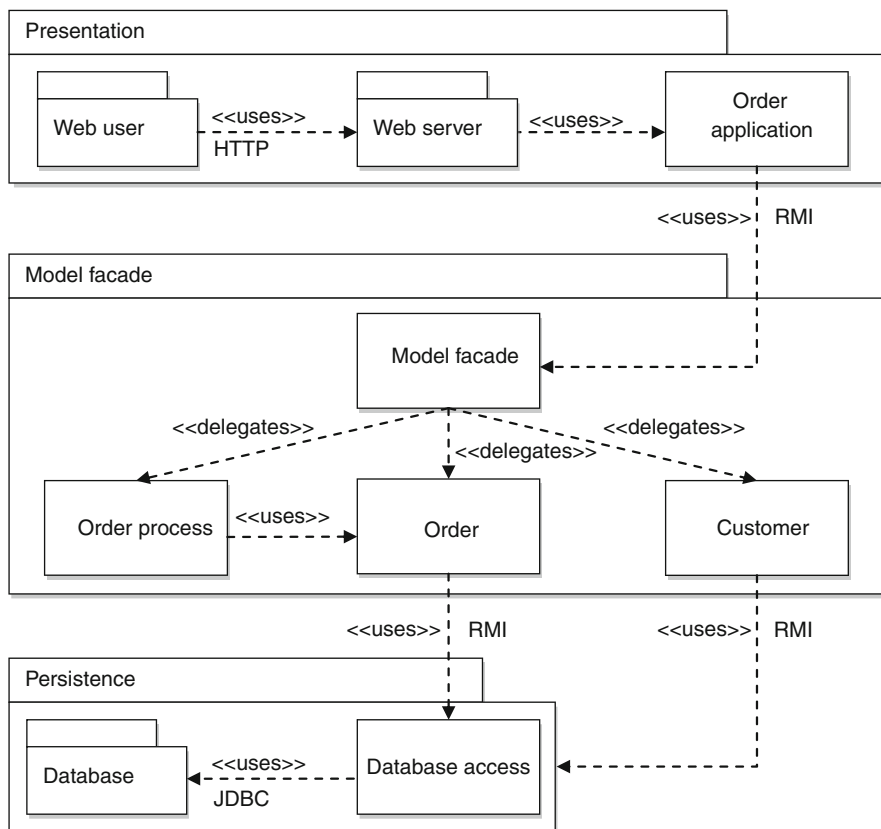


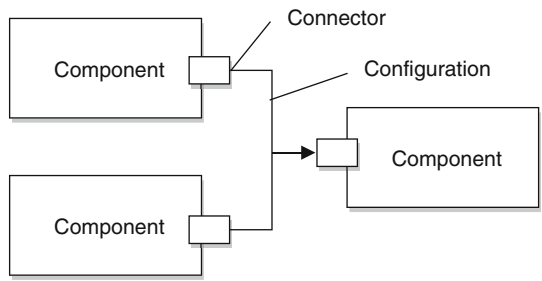
Fig. 2.1 An example for an UML-diagram (Cf. Vogel et al. 2005, S. 201)

- Formal representation of architecture using textual and graphic notations at a very high level of abstraction
- Readability for human and machine
- Possibility for analysis various architectural aspects like completeness, consistency, performance, etc.
- Partly featuring of automated code generation

As mentioned before ADLs concentrate in building an architecture out of components. For those components it describes its data and data integrity as well as their interfaces of any kind. In order to let the components interact among each other ADLs inaugurate connectors. They connect components to each other and determine the rules under which the components can interact. Last but not least ADLs define a architectural configuration which defines the architectural structure by determining which components are connected through which connectors (Fig. 2.2).²¹

²¹ Vogel et al. 2005, S. 204.

Fig. 2.2 ADL core concept
(Cf. Vogel et al. 2005, S. 206)



- DSL²²: DSLs are description languages specific for according domains. The concepts behind it are well formed for the appropriate problem and therefore advantageous over normal ADLs.
- Acme²³: It was developed out of the need to provide interoperability between varieties of other instruments for architectural design. Acme provides three essential capabilities:
 - Architectural interchange: Architects who don't use Acme are capacitated to use their ADL with Acme so that they have a greater choice of what design tools they want to use in order to analyze their architecture. On the other side Acme users can incorporate with other ADLs.
 - Extensible foundation for new architecture design and analysis tools: Because of its fundamental and common approach to architectural design Acme can be used as basis for other more specialized languages still providing the interchangeability. This is reducing costs and time for development of the new language.
 - Architecture description: Actually developed as an interchange tool it evolved as a standalone description language including architectural styles, types, structures plus properties for all elements with which easily software architectures can be created.

2.3.3 Weak Points

Nonetheless above mentioned architect's instruments do have their weak points. As Vogel²⁴ states in a list of handicaps of ADLs their first weak point is that it is not yet clarified which aspects of system architecture an ADL should even cover. There is no distinct determination of what needs to be documented by an ADL. This is so far going that it is not even specified what ADL should be called on what problem. So if

²² Cf. Vogel et al. 2005, S. 206.

²³ ABLE – Architecture Based Languages and Environments 2007.

²⁴ Vogel et al. 2005, S. 202.

an architect needs to document a system architecture of a specific domain he still is spoilt for choice of what kind of ADL he should even use.

That is because there is no single ADL which excels itself as the one standard ADL. They are all specialized to be used for specific domains. And still there is no single ADL for each domain. Each of it is strongly specialized in its own way. Because of their specialization there is no uniform structure amongst the ADLs. Each structure is adapted to their need in the particular domain. The more sophisticated structure supports an ADL's ability to analyze and simulate an architecture of a specific domain.

Hence, a problem of more practical origin arises. Notations of the ADLs also vary due to different structures. But what makes them alike is that their notations are more or less difficult to process and until now there is no support for them by commercial tools.

Another more practical difference between ADLs is the targeted user of an ADL. Some of them support a direct code framework conversion of the created architecture. Those kinds of ADL address architects who are more into the actual implementation of their system whereas other ADLs which do not support such a conversion offer more freedom in the choice of how an architect wishes the architecture to be implemented. It is more of a problem to the actual programmer. ADLs of the second kind keep code away from the architecture and thus make the project more modular.

Moreover, ADLs are still a matter of research. They are often developed by research projects at universities and therefore not really established as a tool for commercial use due to constant development and alteration.

Although ADLs still undergo ambitious and concrete development there is no clear delimitation to other kinds of instruments. For example they still use a broad spectrum of UML-diagrams in order to illustrate the architecture. Whereas it is of interest to use existing technology it is still a problem of independence. So ADLs which rely too much on other instruments are more dependent than ADLs which try to realize the whole architecture with their own methods. On the other hand ADLs which support known technology can be more understandable and more open to others for further development of architectures and development of the ADL itself.

Over all in Vogel's opinion it can be said that ADLs are less general than other instruments. This as a result does not award them with a qualification for the use to describe general aspects of architecture. They can simply not be used for that matter. So ADLs are more and more developed for the needs of a specific domain and therefore become more and more specialized for their area of interest which on the other side deprives them as a candidate to describe more general architectural aspects.

In contrast to ADLs UML as the de-facto standard is more concentrated on classes and objects. Whereas for ADLs components and the connectors between the components are in focus of processing for UML it is more the composition of classes and objects. ADLs do have a more precise syntax adapted for the domain the ADL is in use. They focus on interfaces of the components and the connectors as a central concept. In comparison UML is not specialized in any way.

This enables a user to operate with it in a greater variety for documentation. But this causes the constricted possibilities to analyze and simulate the problem for which it was pulled up. Moreover, it can lead to a misunderstanding by stakeholders due to unclear semantics of notation. To partially avoid this problem UML can be extended with concepts of any kind (like ADL-concepts). This makes it more precise and accurate to a specific problem but again withdraws general aspects of UML.



<http://www.springer.com/978-3-7091-0754-6>

Haptic Systems Architecture Modeling

Weber, A.; Dustdar, S.

2012, IX, 124 p., Hardcover

ISBN: 978-3-7091-0754-6