

## Chapter 2

# Viewpoints for Describing Software Architectures

In this chapter we present the architectural framework provided by SysADL. We define software architecture and the fundamental notion of software architecture description according to the ISO/IEC Standard 42010 “Systems and Software Engineering—Architecture description” [1]. We present, in detail, the concepts underlying the architecture description in terms of *viewpoints* and *views*.

You will learn the following:

- the definition of software architecture;
- the concepts of viewpoints and views for describing a software architecture;
- the architectural framework provided by SysADL in terms of concepts, viewpoints, and views.

### 2.1 The Definition of Software Architecture

The ISO/IEC Standard 42010 “Systems and Software Engineering—Architecture description” defines software architecture as “*The fundamental properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution*”.

As we can see, this definition highlights three matters in an architecture:

- (i) the elements;
  - (ii) the relationships between elements;
  - (iii) the principles in the design and the evolution of these interrelated elements.
- This definition also highlights that the architecture comprises the implied properties emerging from these constituents.

## 2.2 Software Architecture Description

An architecture needs to be described to be used by its different stakeholders. According to the ISO/IEC/IEEE 42010 Standard, an architecture description is: “a work product used to express an architecture.”

Let us now put the notion of architecture description in context, before presenting its characteristics. Figure 2.1 depicts this context.

Let us now present this conceptual model of the context of an architecture description:

- An architecture description expresses, at least, one architecture, but not all architectures are described. It means that an architecture description may describe one or many architectures.
- An architecture may be exhibited by none or several systems. It means that an architecture is a notion that may be realized in different concrete systems.
- A system is situated in an environment. It means that the interaction with the environment needs be taken into account in the architecture description. Note that a system is more than the software part of it. Indeed, it is a software-intensive system.

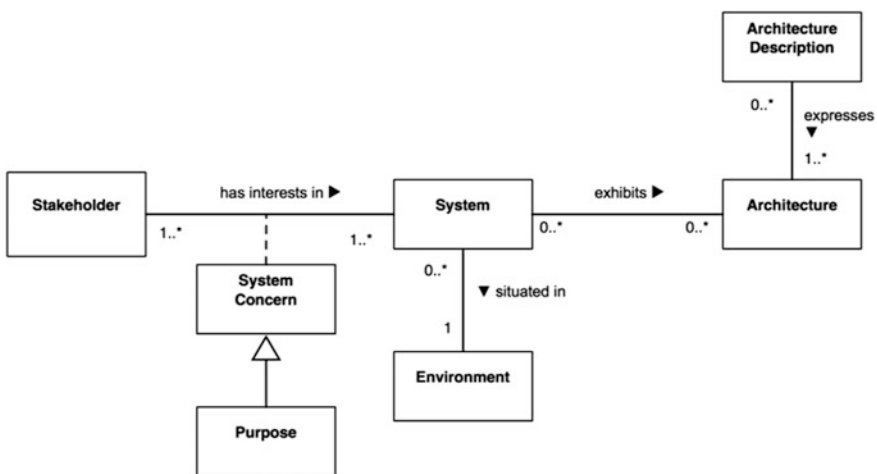


Fig. 2.1 The context of an architecture description [ISO/IEC/IEEE 42010]

Now let us examine the involved stakeholders. They comprise person, group, or organization with an interest in one or several systems. The interests are expressed as *concerns*. Each concern has a *purpose*.

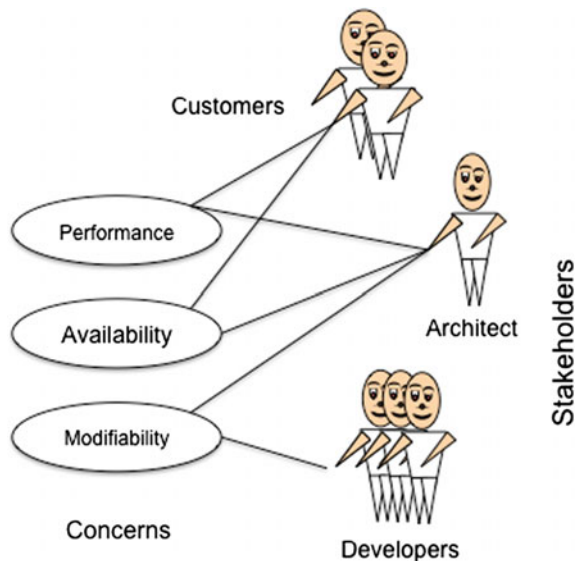
For instance, considering the RTC system, examples of stakeholders are: clients, the architect, and developers. Clients are the users of the system, thereby they are concerned with performance and availability. The associated purpose for availability is to enable the use of the system whenever needed, and the purpose for the performance concern is the effectiveness and efficiency of the system. Developers are mainly concerned with modifiability, in order to facilitate maintenance and also to reduce the costs. All those concerns are important to the architect who is in charge of developing a solution that satisfies client and developers concerns. Figure 2.2 illustrates these stakeholders and concerns.

Let us now explain the elements of an architecture description based on the conceptual model of an architecture descriptions provided by the ISO/IEC/IEEE 42010 Standard, depicted in Fig. 2.3.

According to this conceptual model, an architecture description:

- identifies a system-of-interest (it is the system that, for instance, will be developed based on the described architecture);
- identifies the system stakeholders and their concerns (the stakeholders that will, for instance, use and develop this system);
- defines the architecture viewpoints used to represent the architecture (these viewpoints support the description of architecture in terms of different views);
- supports the correspondences between views in terms of architectural elements, possibly defined by correspondence rules (see Fig. 2.4);
- documents the architectural decisions with their rationale.

**Fig. 2.2** Stakeholders and concerns in the RTC system



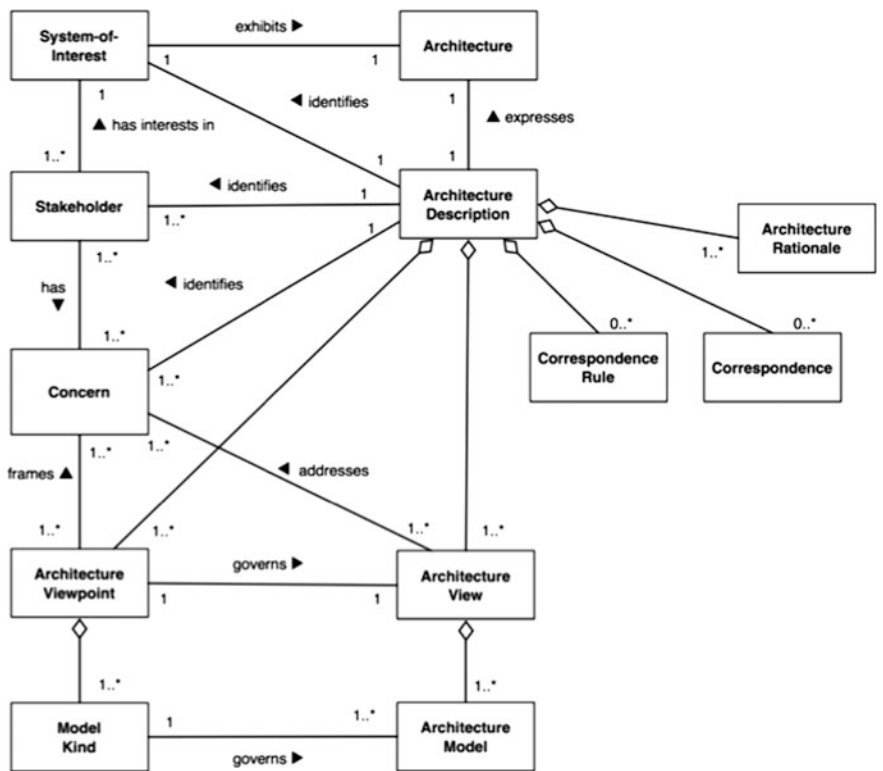
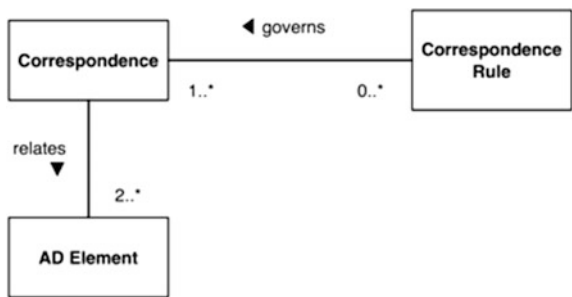


Fig. 2.3 Conceptual model of an architecture description [ISO/IEC/IEEE 42010]

Fig. 2.4 Correspondences relating architectural elements [ISO/IEC/IEEE 42010]



A key concept in architecture description is *viewpoint*. All concerns are mapped onto the viewpoints. Each architectural viewpoint governs one or more architecture *views*. This means that the architecture viewpoint defines the constructs, including rules and conventions, for the definition of the views describing an architecture. An architecture view comprises one or more architecture models. Figure 2.5 illustrates three viewpoints, each one with a view.

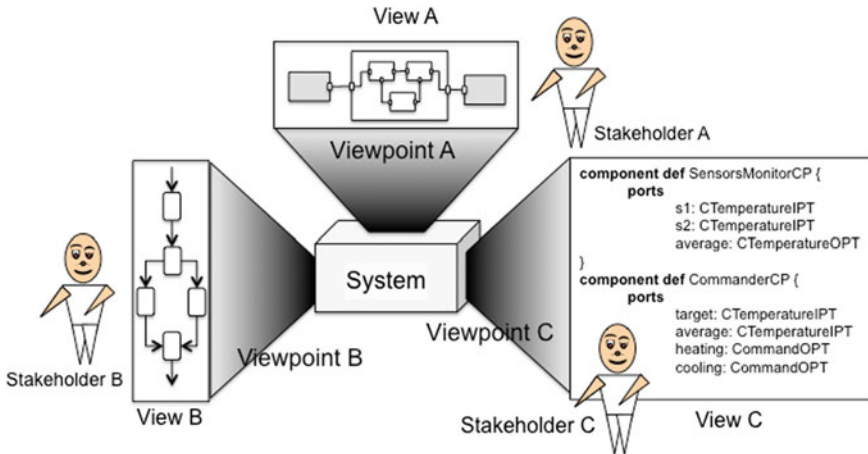


Fig. 2.5 Viewpoints and views

In this context, an architect has the role of describing the architecture of the system-of-interest, considering the environment, the stakeholders, and their concerns. The architect will describe the architecture using different architecture views according to the viewpoints supported by an *architecture framework*. According to the ISO/IEC Standard 42010, an architecture framework provides the “conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders.”

SysADL is an architecture framework for software engineers and developers, providing a set of viewpoints to the stakeholders. For each viewpoint, the SysADL framework defines a language with conventions and principles about the definition and use of the architectural elements of the viewpoint.

## 2.3 Concepts for Describing Software Architecture

The ISO/IEC/IEEE 42010 Standard specifies a conceptual model of architecture description, but it does not define the main building blocks and primary elements of an architecture description. For this purpose, we adopt the well-known abstractions of *components*, *connectors*, and *configuration*. These concepts are independent of any specific notation, concrete syntax, or architecture language.

Figure 2.6 shows a conceptual model relating these three architectural elements. This figure shows that an architecture description element (*AD Element*) may be a component, a connector, or a configuration. We define that a configuration may contain one or more components, and none or several connectors. A connector connects two or more components.

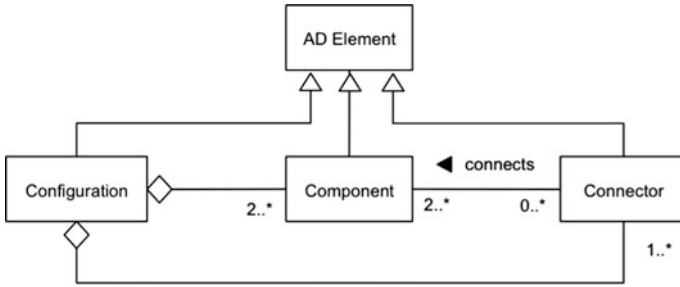


Fig. 2.6 Architectural description elements

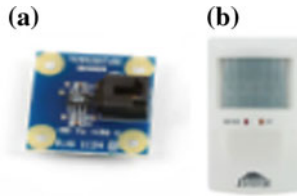


Fig. 2.7 a Temperature sensor component; b Presence sensor component

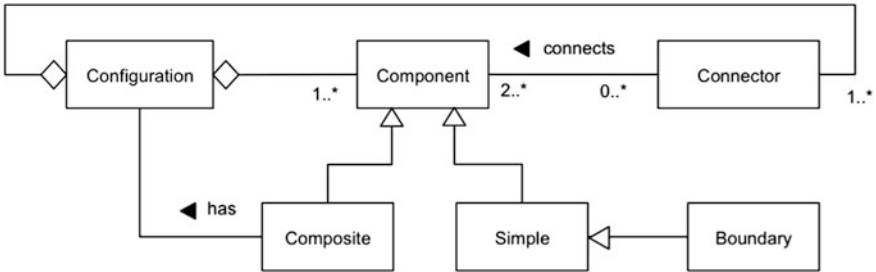
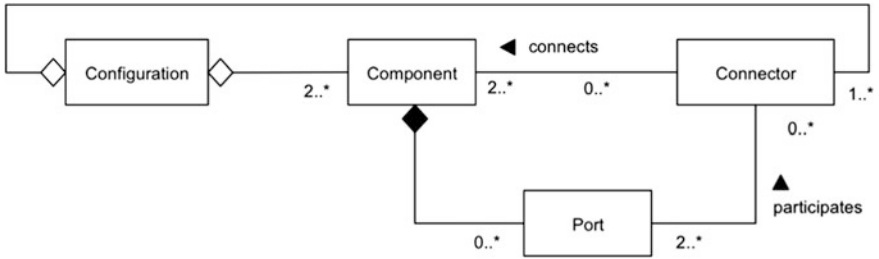


Fig. 2.8 Simple and composite components

Components are architectural elements that provide functionalities in a system. It is the central element, the loci of computation and state. Figure 2.7 illustrates two components: a temperature sensor and a presence sensor.

As depicted in Fig. 2.8, a component can be a simple element providing a simple functionality, or even a composite element representing a system as a whole, which itself is composed of others components too. A *simple component* performs sequential computation using data available in its ports. A *composite component* performs concurrent computation by being composed of components in their internal structure.

A component has clearly defined *ports* that are interaction points between it and the external world, i.e., its environment. Ports specify the data that a component



**Fig. 2.9** Ports

provides or requires from other components in the architecture. The explicit specification of the data that a component provides and requires is essential to support the component’s composability. In fact, the specification of an architecture follows the idea of assembling components together to form a system using information from their interfaces. Figure 2.9 shows that a component has none or several ports. Thus, ports belong to components and participate to connectors (ports do not belong to connectors).

Components can be internally located in a system or in the boundary between the system and the environment. *Boundary components* are thereby placed at the interface with the environment. Internal components are located inside a composite component including the architecture itself, and therefore are not visible outside of it. A component that is located in the boundary represents the system interface with the environment, encapsulating the mechanisms the system uses to interact with the environment. As an example, you can see that the *RTC System* has different boundary components such as a sensor that physically measures the temperature and provides its value in an *out* port. Figure 2.7 illustrates two boundary components.

The second fundamental concept in an architecture description is a *connector*. A connector is an element responsible for mediating the interaction among components, establishing rules that govern those interactions. It provides the glue mechanism to binding components together. Figure 2.10 illustrates a connector. Note that each side of the connector specifies the kind of component’s port that can be bound to it.

Connectors define which ports can be connected and how the interactions between connected components take place. Therefore, they are the locus of communication. As sketched in Fig. 2.9 a connector refers to the ports it connects. It connects at least two ports.

As shown in Fig. 2.11, a connector can be a *simple* element connecting two or more ports, or even a *composite* element which itself is composed of others

**Fig. 2.10** Connector



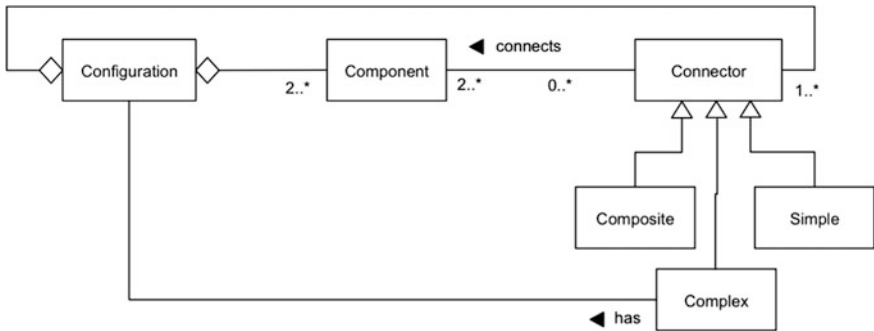
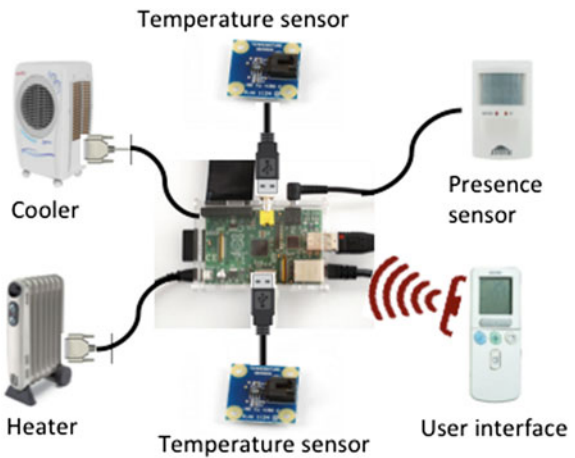


Fig. 2.11 Simple and composite connectors

Fig. 2.12 Configuration of the RTC System



connectors, or a *complex* element that has a configuration. For instance, in the case of the RTC system, a connector (Fig. 2.12) could be used to connect the temperature sensor component with the controller component.

As you can see, components and connectors are complementary concepts and provide a clear separation of concerns between computation and communication.

The third fundamental concept in an architecture description is *configuration*. *Configurations* describe the topology for identifying which components are part of a software architecture and how they are connected together through connectors. In this way, the architecture configuration defines a connected graph of components and connectors that describes the architecture. In the example of the RTC system, as shown in Fig. 2.12, the configuration could define that the controller is connected in a star topology configuration with two temperature sensors, one presence sensor, and two actuators—the cooler and the heater. They use cable connectors to send and receive data to the controller. The user interface is a remote control that sends data via an infrared connector.



These three concepts raise different needs in terms of software architecture description:

- how to describe the ports of components?
- how to describe the ports of connectors?
- how to describe the configuration of components and connectors?
- how to describe the behavior of components?
- how to describe the behavior of connectors?
- how to describe the behavior of the configuration of components and connectors? And also:
- how to validate the designed architecture?
- how to verify the implied properties of the designed architecture?

To address these needs, architecture description languages (ADLs) were defined.

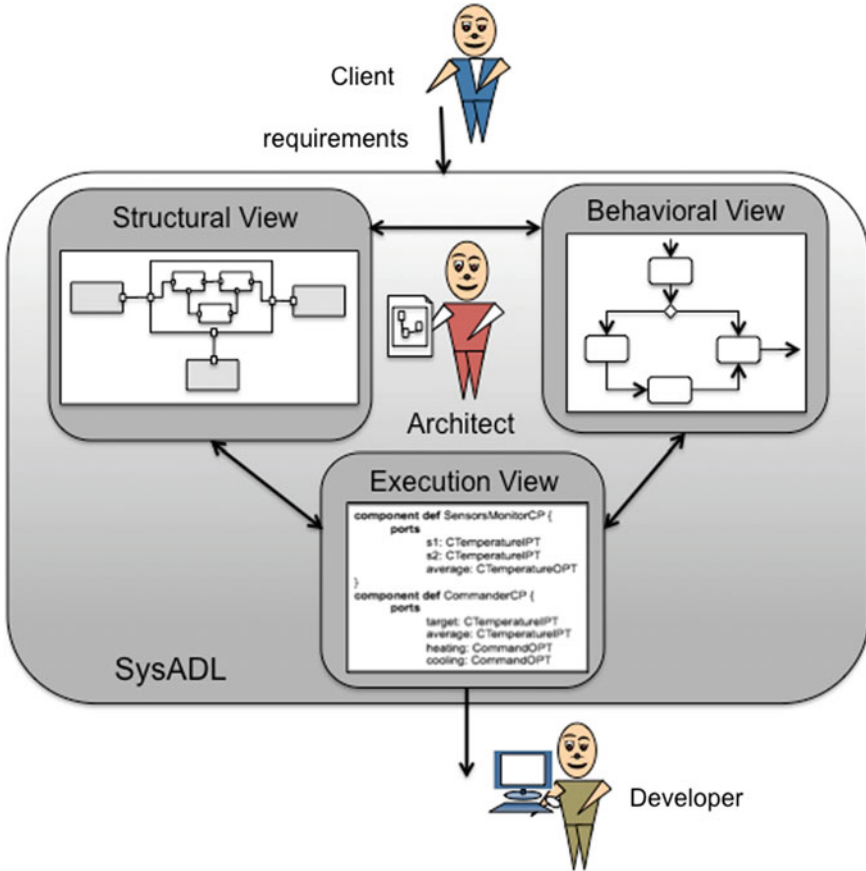
## 2.4 Architectural Viewpoints and Views

As specified in the ISO/IEC/IEEE 42010 Standard, an architecture is described from different viewpoints in terms of *views* represented by *model diagrams*. We will present now these two notions.

### 2.4.1 Architectural Viewpoints

Viewpoints realize the stakeholders concerns by means of *architecture views*. A viewpoint defines the kinds of model that govern the diagrams used to represent its corresponding views. Such models are what a stakeholder “sees” when looking at the system from a specific viewpoint. The ISO/IEC/IEEE 42010 Standard also emphasizes that multiple views are essential to cover all the stakeholders’ concerns, and to detail the architecture from different perspectives.

Let us make an analogy between software architecture and civil architecture, which also represents architectures from different viewpoints. The well-known structural, electrical, and hydraulic plans are some examples of civil architecture views from the structural, electrical, and hydraulic viewpoints. Each one provides a specific view of a building. The plans are used by the stakeholders—engineer, architect, brick worker—to reason about and govern the building process. They help the stakeholders to understand the building concerns. Similarly, software architecture views show different perspectives of the software-intensive system. They communicate the architecture to the different stakeholders.



**Fig. 2.13** Architectural viewpoints

The main stakeholders interested in the software architecture description are clients, architects, and developers. As illustrated in Fig. 2.13, the client specifies the requirements of the system to be realized by the architecture, and is interested in the executable architecture satisfying the requirements. The architect is concerned with the specification of the architecture, reasoning about it to verify properties and detailing its structure, behavior, and execution. Structure and behavior are declarative specifications that can be mapped to a corresponding executable model, characterizing the executable viewpoint. Finally, the developer is interested in the executable architecture for implementing the system.

According to the concerns depicted, these three architectural viewpoints are essential to communicate the architecture to the involved stakeholders.

### 2.4.2 Architectural Views

As discussed, an ADL must support different viewpoints. SysADL defines three architectural viewpoints that are essential to communicate the software architecture to the involved stakeholders:

- (i) the structural viewpoint;
- (ii) the behavioral viewpoint;
- (iii) the executable viewpoint.

Each viewpoint provides the constructs for describing different views of the architecture from these viewpoints. For instance, Fig. 2.14 shows two viewpoints (style level viewpoint and instance level viewpoint) and an architecture description with three views (one for each of the three viewpoints).

The structural view describes the main building blocks of the architecture from a conceptual point of view: components, connectors, and configurations. It makes use of two kinds of diagrams: the block definition diagram (*bdd*) and the internal block diagram (*ibd*). The *bdd* is used to define the components and connectors of the architecture. The *ibd* goes a step further, showing how components and connectors bind each other defining the configuration of the architecture or of the internal structure of composite components and connectors.

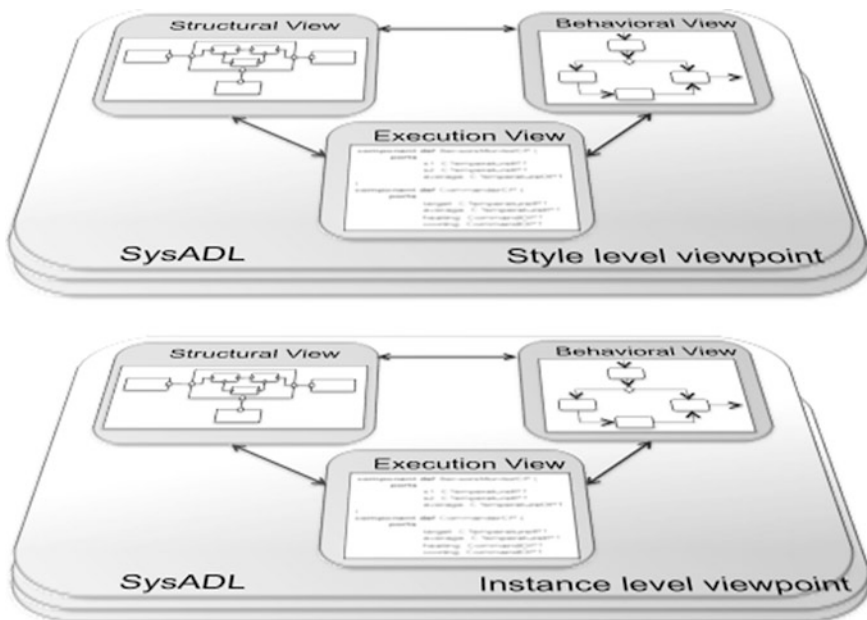


Fig. 2.14 Architectural views from viewpoints

The behavioral view is concerned with the behavior of the architecture, specifying sequential and concurrent activities by making use of the activity diagram (*act*). The behavioral description also includes the specification of protocols supporting the interaction between components through connectors and the specification of actions. In the latter case, parametric diagrams are used (*par*).

The executable view describes the execution details of the architecture. It represents the code view. This viewpoint provides a superset of the OMG Action Language for Foundational UML (ALF) as part of SysADL. The use of ALF allows for a direct mapping onto the programming language level.

Note that, in addition to views, we have two abstraction levels: style level and instance level.

The *style level* is where the architecture style is specified, defining the types of components and connectors that compose the architecture, as well as the set of constraints on how they are combined. The style is used to derive instances of the architecture, following the types and constraints that are defined. For instance, the client–server style defines server and client as component types, a server–client connector type to connect servers and clients, and constraints on systems composition specifying the kinds of allowed compositions. For example, clients must not communicate among them, clients communicate only with servers, servers may not initiate the communication with a client, and may allow no more than 10 clients to be simultaneously connected to them.

The *instance level* is where the instances of the style are defined. For each style, several instances can be defined. An instance is specified by instantiating the elements that compose the style, and by satisfying the constraints. For the client–server style, a possible simple instantiation can be a Web server connected to different clients using one connector for linking each client to the server. Another instantiation can be a banking server with a dedicated connector for each client.

## 2.5 Summary

In this chapter, you have learnt the following:

- how to define a software architecture;
- what are the underlying concepts needed for describing a software architecture: viewpoints and views;
- what is the architectural framework provided by SysADL in terms of viewpoints and abstraction levels.

## Further Reading

1. Bass, L., Clements, P., Kazman, R.: Software Architectures in Practice, 2nd edn. Addison Wesley, Reading (2003)
2. Clements, P., Bachmann, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R.: Documenting Software Architecture: Views and Beyond. SEI Series in Software Engineering (2003)
3. Rozanski, N., Woods, E.: Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley (2012)

## Reference

1. [www.iso-architecture.org/42010/index.html](http://www.iso-architecture.org/42010/index.html)

Software Architecture in Action

Designing and Executing Architectural Models with  
SysADL Grounded on the OMG SysML Standard

Oquendo, F.; Leite, J.; Batista, T.

2016, XVII, 236 p. 249 illus., 46 illus. in color., Softcover

ISBN: 978-3-319-44337-9