Fachhochschule Nordwestschweiz
Hochschule für Technik

Software Architecture

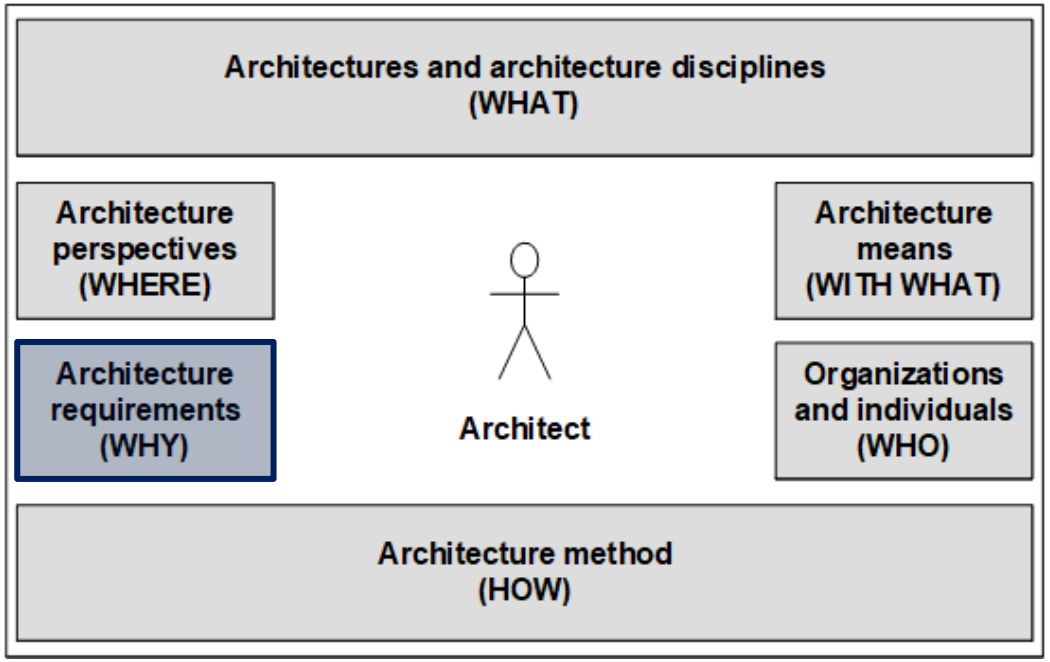
Architecture Conditions

BSc

Ingo Arnold

Lecture Opening
Architecture Orientation Framework

Architecture WHY deals with the **drivers and forces architecture has to cope with** – from constraints and assumptions to requirements [Vogel, Arnold et al 2011].



Lecture Opening
Motivation

In this unit we consider the **drivers of architectural action** (i.e., architecture conditions) – from constraints to assumptions to requirements.

We will present heuristics along which given architecture conditions can be assessed for their **architectural significance**. This is an important step to assess architecture relevance.

Finally, we will look at how to ensure **traceability of conditions in architecture approaches** that address them.

Lecture Opening
Learning Objectives

You …

- understand the role of conditions in the development process.
- know the importance of different types of conditions and how they are adequately captured.
- are able to assess architecture-significance and thus determine architecture conditions.
- understand the role of architecture approaches and how conditions are tracked in approaches.

Lecture Agenda

- Architecture Condition
- Architecture-Significance
- Architecture Approach

Architecture Condition
Conditions meta model

Conditions as motivators. The architecture of a system does not arise randomly. Requirements, constraints, and assumptions are conditions under which a system and its architecture emerges. While architecture conditions constitute a system of forces the architecture of a solution is designed to respond and balance these forces [Arnold 2021].

Architecture conditions must be …

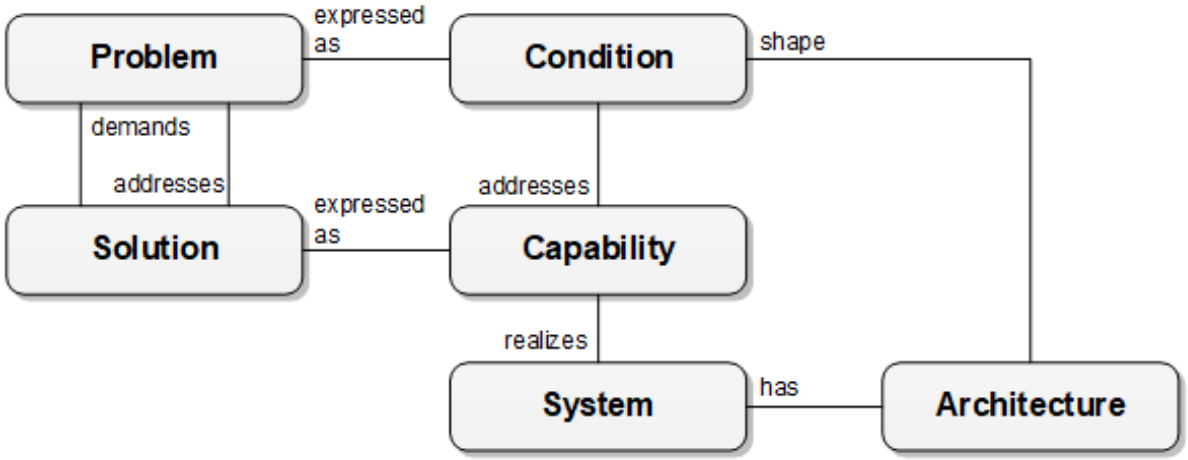
- **Correct**. However, only stakeholders can assess whether this is the case.
- **Feasible**. It must be possible to realize the condition under the given circumstances and with the means available
- **Unambiguous**. A condition must be formulated in such a way that the reader can only draw one conclusion from it.
- **Verifiable**. Even the best condition is useless if you cannot verify it reliably.

Architecture Condition
Conditions meta model

A **solution addresses a problem** by absorbing the conditions representing the problem.

Conditions constrain the space to manoeuvre in solution endeavours. They can be divided into those that express …

- a desire in an organization (e.g., requirements, principles)
- uncertainty that you cannot ignore (e.g., assumptions)
- limiting factors constraining the constitution of a solution (e.g., constraints).



Architecture Condition
Types of conditions

Conditions are differentiated as …

- **Requirements** represent needs an organization wish to addresses when it responds to a problem.
  - **Functional requirements** are existential needs. They are the reason for a system's existence and express what your organization expects a solution to contribute.
  - **Non-functional requirements** are not existential, such that they are no initial reasons for a system's existence. However, they are mostly still essential, since they express *how* a solution needs to acceptably make its crucial contributions.
- **Constraints** are assertions of a fact which cannot be ignored. Where requirements express a desire, constraints express facts one cannot circumvent (e.g., time, skills, or budgetary constraints).
- **Assumptions** express relevant areas of uncertainty – known unknowns if you wish. They are conditions which likely (but not necessarily) materialize in future. Assumptions can be made about requirements and constraints. For those decisions you based on assumptions, it is more likely that you need to revert them.

Architecture Condition
Capturing conditions

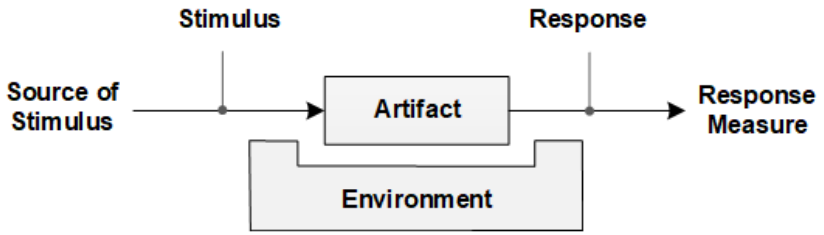
Architecture Conditions can be captured via …

- **Use Cases** (functional requirements) represent needs an organization wish to addresses when it responds to a problem.
- **Quality Attribute Scenarios** (non-functional requirements, constraints, and assumptions) are not existential, such that they are no initial reasons for a system's existence.

Use Case Template

Name	
Description	
Actor Description	
Pre-conditions	- <precondition$_1$> - <precondition$_2$>
Behavior	Step / NFR
Alternative Behavior	Step / NFR
Post-conditions	- <postcondition$_1$> - <postcondition$_2$>
Special requirements and invariants	- <special requirement$_1$> - <special requirement$_2$>

Quality Attribute Scenario



Architecture Condition
Capturing conditions

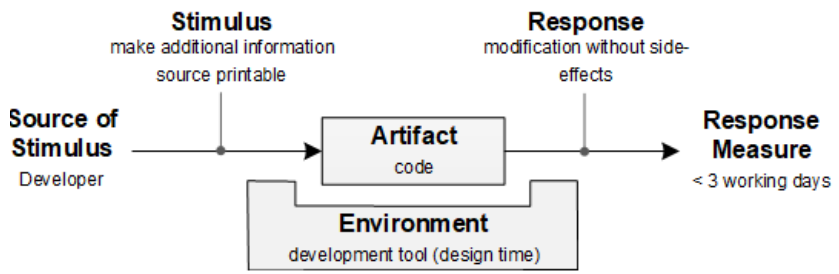
Architecture Conditions can be captured via …

- **Use Cases** (functional requirements) represent needs an organization wish to addresses when it responds to a problem.
- **Quality Attribute Scenarios** (non-functional requirements, constraints, and assumptions) are not existential, such that they are no initial reasons for a system's existence.

Withdraw Money Use Case

Name	Withdraw money
Description	Bank customers can withdraw money from their account up to their credit limit.
Actor Description	Bank customer
Pre-conditions	- Successful authentication of bank customer - ATM displays menue
Behavior	1. Bank customer choses to withdraw money from his account 2. Bank customer enters desired amount 3. ATM checks whether amount is within limit 4. ATM deposits amount from account 5. ATM dispenses amount
Alternative Behavior	1. If amount was not within limit ATM makes a max amount offer 2. User accepts offered amount and proceeds XOR refuses and ends the session
Post-conditions	Bank user cancelled session XOR received amount ATM displays landing page
Special requirements and invariants	ATM information must be secured (at rest and in transit)

Quality Attribute Scenario



Lecture Agenda

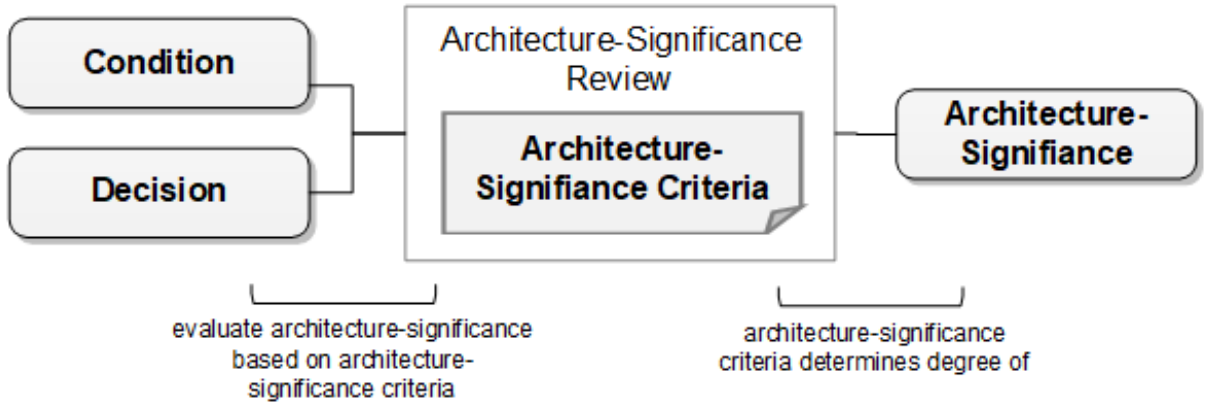
- Architecture Condition
- Architecture-Significance
- Architecture Approach

Architecture-Significance
Definition

Grady Booch [Booch 2009] used an indirect (i.e., from the perspective of effect – not cause) approach to define architecture:

> "Architecture is the set of significant design decisions that shape a system, where significant is measured by cost of change"

One cannot always clearly and unequivocally determine whether a given condition or decision is of architecture significance. At the same time, **you increase your productivity** to the extent that you draw your attention to architecture-significant items and relieve it from insignificant contributions.



Architecture-Significance
Definition

A pragmatic heuristic to determine the significance of architecture conditions as well as decisions [Arnold 2021]:

- **Number of impacts caused by a given condition**. Expected number of architecture decisions or approaches required to address a condition. If a condition is likely to impact multiple architecture decisions simultaneously, it is more likely to be architecture-significant than conditions for which this does not apply.
- **Stakeholder importance**. Importance of the stakeholders who contributed the condition, such as business stakeholders are more important than stakeholders representing technical concerns. A condition raised by important stakeholders tends to be more significant than conditions raised by less important stakeholders.
- **Non-functionality**. Non-functional conditions, such as availability, security, or performance conditions, often have a significant impact on the architecture. Consider all conditions affecting system quality attributes as architecture-significance candidates.
- **Novelty**. Degree to which the condition is unusual or new. The newer a condition and the less experience in dealing with it, the more significant it should be rated as from an architecture perspective.
- **Volatility**. Probability that a given condition will change continuously. Conditions that change continuously place high demands on the modifiability of a system and are therefore considered drivers of architecture significance.

Architecture-Significance
Definition

A pragmatic heuristic to determine the significance of architecture conditions as well as decisions [Arnold 2021]:

- **Degree of conflict**. Conflict between a given and other architecture conditions. The degree of conflict is derived from the number of trade-offs required to resolve clashes. The more compromises are likely required to be found for a given condition, the more architecturally significant such condition is.
- **Degree of strategic orientation**. Extent to which a condition directly contributes to achieving the enterprise's strategic objectives. If a condition's estimated contribution to achieving strategic enterprise goals is high, it is to be assessed as more architecture-significant than if only few direct strategy contributions are made.

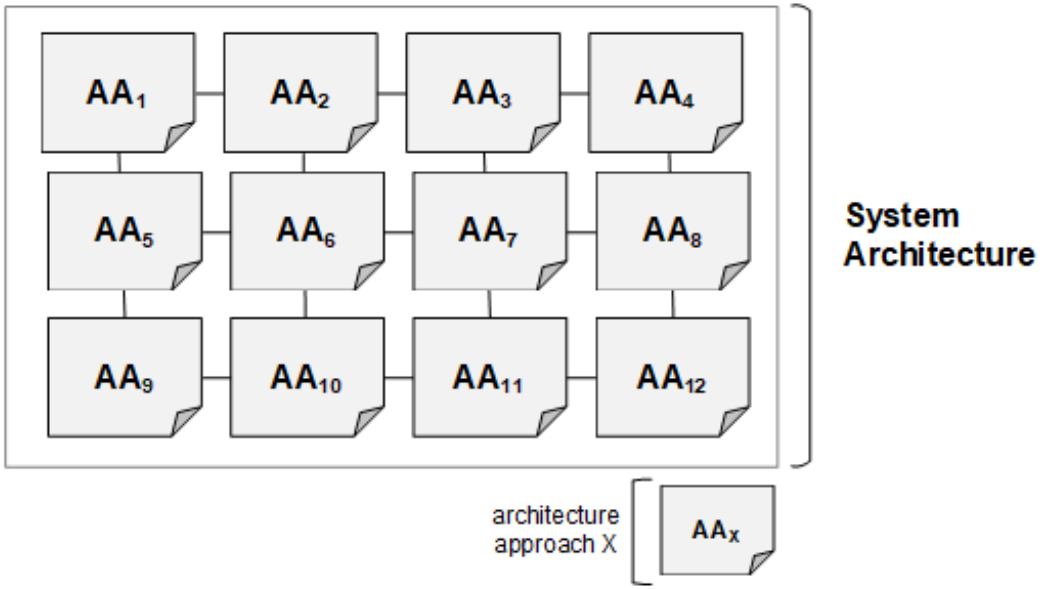
Lecture Agenda

- Architecture Condition
- Architecture-Significance
- Architecture Approach

Architecture Approach
Overview

An **architecture approach** is a part of a corresponding overall architecture. Each approach addresses a subset of the conditions of an overall problem [Arnold 2021].

From the perspective of the overall design, an **architecture approach represents a partial solution**. Partial solutions are combined into the overall architecture by the architect responsible for the entire solution.



Architecture Approach
Overview

You plan, design, and describe individual architecture approaches based on problem-solving methods and techniques.

In this respect, an architecture approach is no different from an overall architecture. Architecture approaches are therefore also referred to as solution design segments, micro-solutions, or microarchitectures.

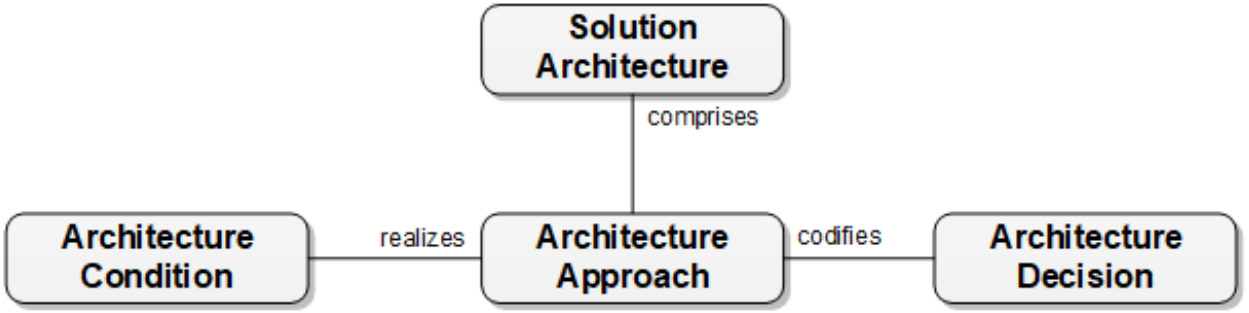
Each **architecture approach implements one or more conditions** – for example, requirements, constraints, or assumptions.

For a functional requirement, an architecture approach suggests a design that realizes a use case.

For a non-functional requirement, constraint or assumption, an architecture approach specifies how a corresponding quality attribute scenario is realized based on its design.

Architecture Approach
Overview

Architecture approaches provide a hinge between architecture conditions and decisions, enabling detailed traceability between them.



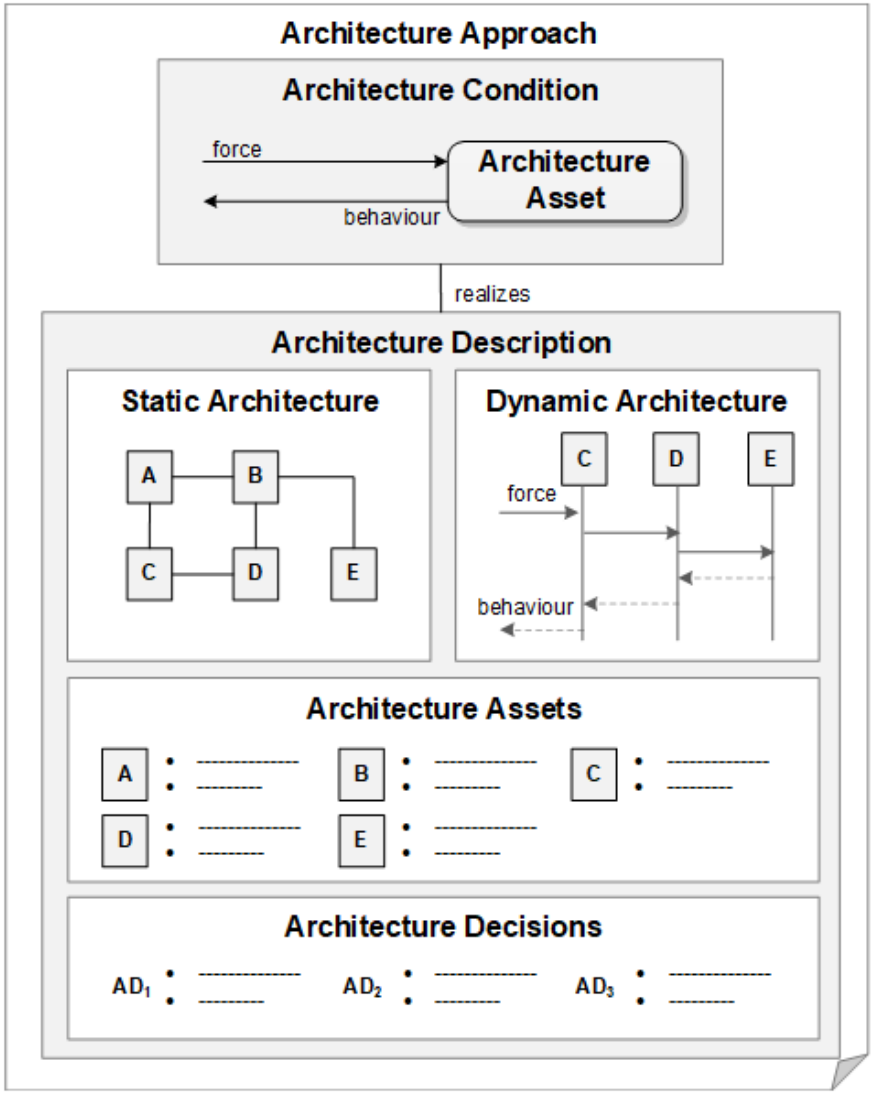
Architecture Approach
Overview

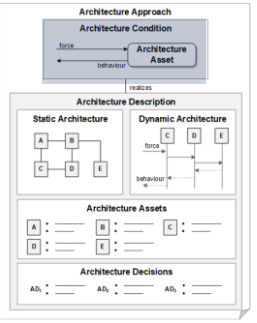
Architecture approach descriptions refer to the conditions they address.

So, they describe both the impact (force) on a corresponding system and the desired response (behavior).

Also, they describe the architecture building blocks (aka: architecture assets) constituting the architecture approach dynamically and statically.

Finally, they capture the most prominent architecture decisions.

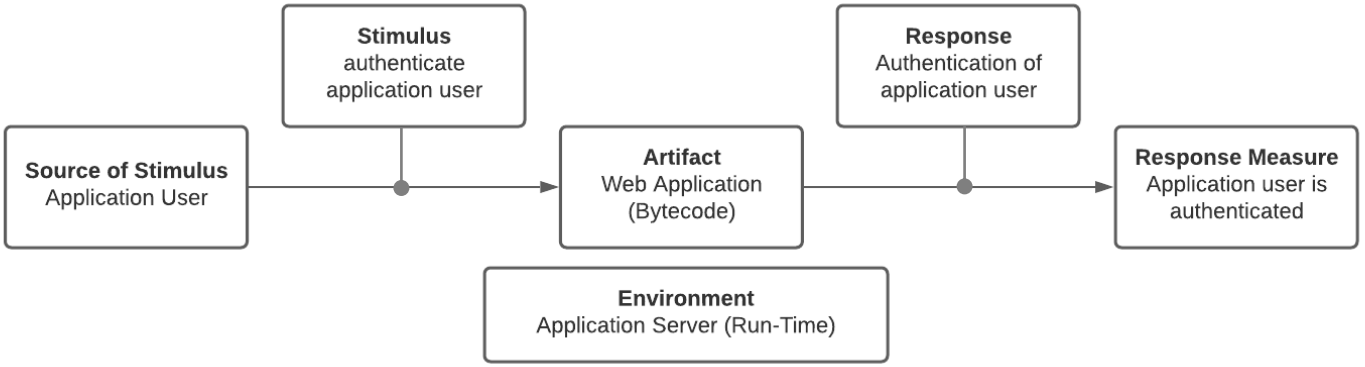




Architecture Approach
Example

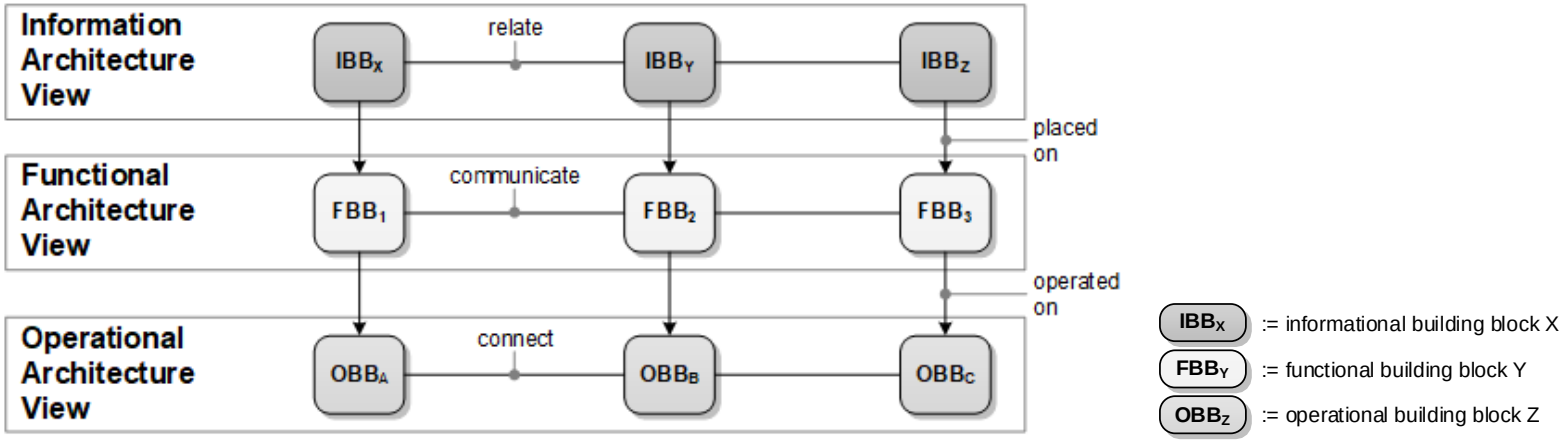
Users of a web application should be checked to see if they are who they claim to be – in other words, they should be authenticated.

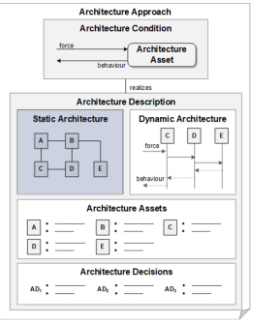
A user base already exists in the company's own user directory (i.e., technically an LDAP directory).



Architecture Approach
Example

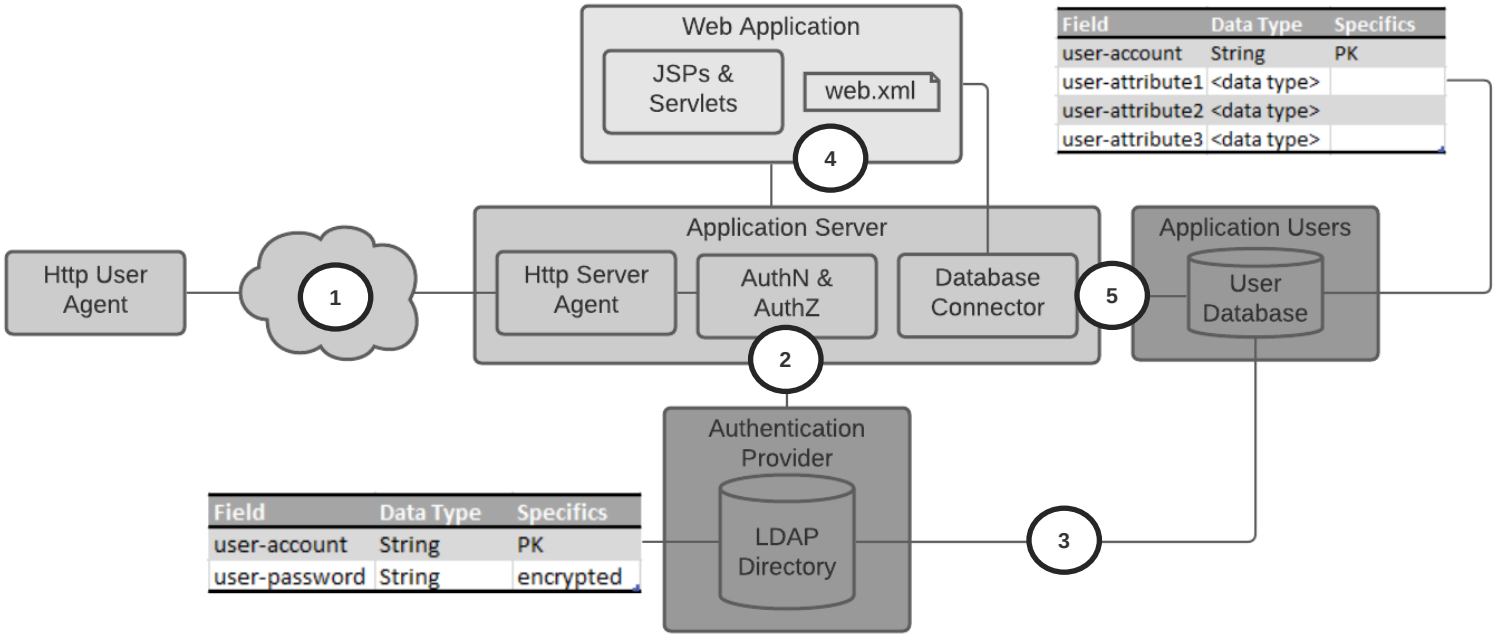
Remember that **solution architecture-related view models** typically distinguish between three elementary views and differentiate between functional, informational and operational building blocks. Furthermore, such view models clarify elementary relationships that exist between their views.

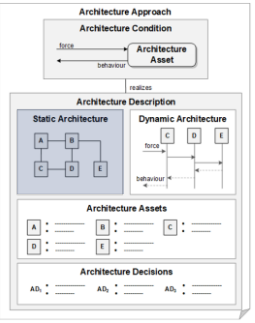




Architecture Approach
Example

A possible static design for addressing the authentication requirement could look like this.





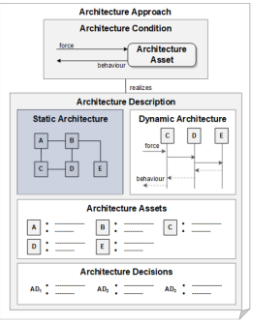
Architecture Approach
Example

A possible static design for addressing the authentication requirement could look like this.

1. An Http User Agent is connected via Internet (http) protocol to a corresponding Http Server Agent. The server agent is part of an application server.
2. The application server has a modular structure and provides for a connection to an (external) authentication provider via an AuthN & AuthZ module (configured in server.xml).

```
<Realm  className="org.apache.catalina.realm.JNDIRealm"
        connectionURL="ldap://localhost:389"
        userPattern="uid={0},ou=people,dc=company,dc=com"
        roleBase="ou=groups,dc=company,dc=com"
        roleName="cn"
        roleSearch="(uniqueMember={0})"
/>
```

3. Synchronisation must be ensured between the LDAP directory and the authentication information for users contained in the application-specific user database.

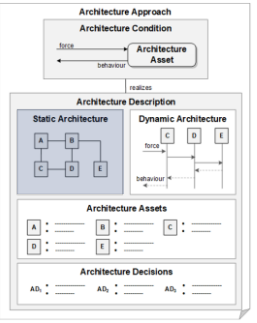


Architecture Approach
Example

A possible static design for addressing the authentication requirement could look like this.

4. The web application is deployed on the application server. The deployment descriptor of the application specifies which of its areas are to be protected by which authentication provider.

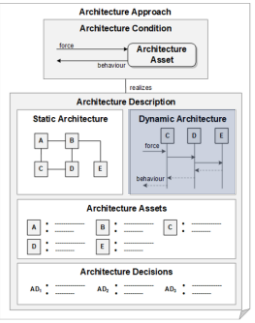
```
<web-app>
        <security-constraint>
            <display-name>SecurityConstraint1</display-name>
            <web-resource-collection>
              <web-resource-name>WRCollection</web-resource-name>
              <url-pattern>/*</url-pattern>
            </web-resource-collection>
            <auth-constraint>
               <role-name>loginUser</role-name>
            </auth-constraint>
        </security-constraint>

        <login-config>
          <auth-method>FORM</auth-method>
          <realm-name id="JNDIRealm"/>
          <form-login-config>
            <form-login-page>/login.jsp</form-login-page>
            <form-error-page>/login-failed.jsp</form-error-page>
          </form-login-config>
        </login-config>
</web-app>
```



Architecture Approach
Example

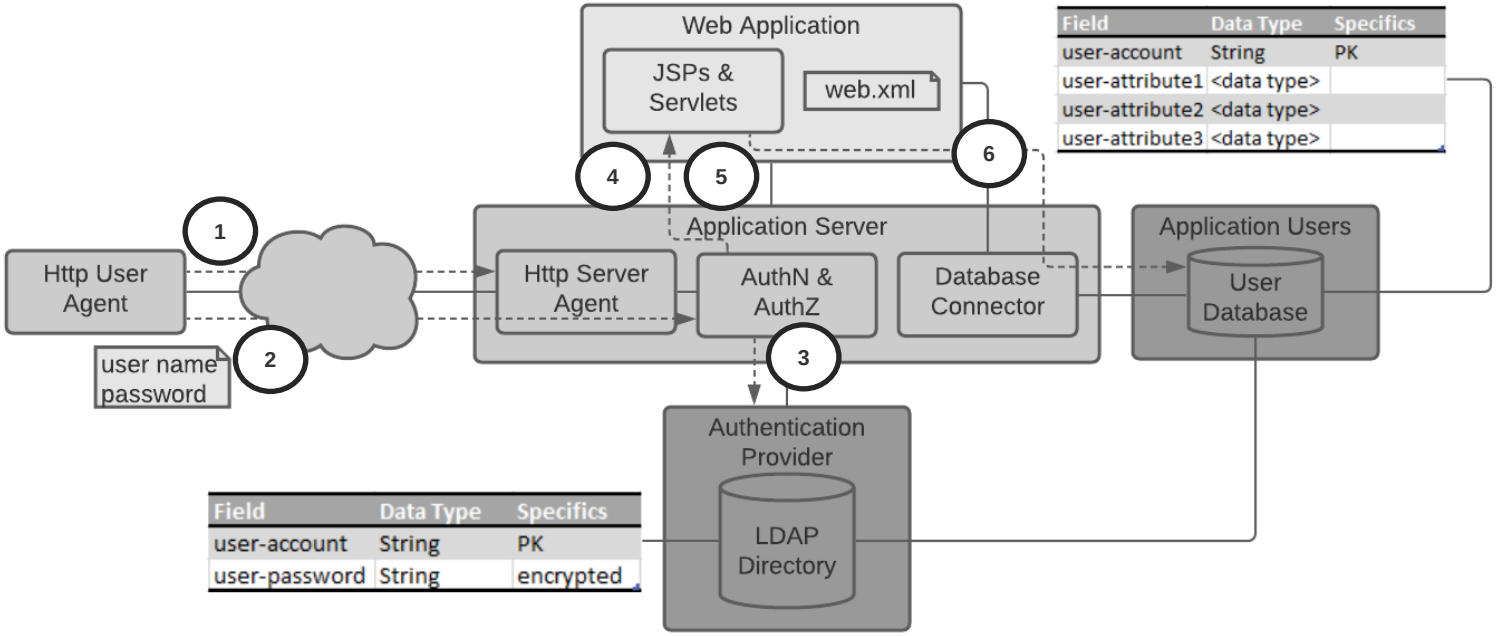
A possible static design for addressing the authentication requirement could look like this.

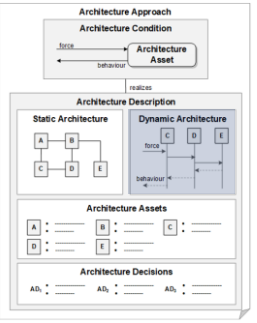
5. The web application can read out further user attributes from the application-specific user database via a corresponding database connector and base the corresponding application logic on this. In addition, the methods getRemoteUser() and isUserInRole() are available to the web application (via HttpServletRequest object).



Architecture Approach
Example

A possible dynamic design for addressing the authentication requirement could look like this.

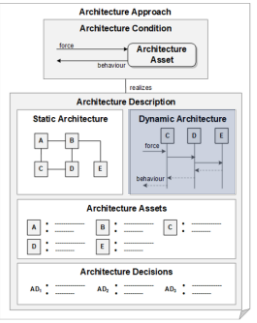




Architecture Approach
Example

A possible dynamic design for addressing the authentication requirement could look like this.

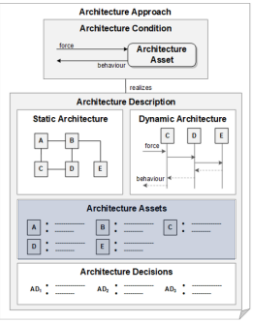
1. The user attempts to access the web application. The application server uses the authentication information from the deployment descriptor to perform the authentication with the user. It may send the browser an instruction to display a dialog box to enter user name and password (HTTP Basic Authentication). Or it may use a login form defined by the web application to gather user name and password (Form Based Authentication).
2. The application server gets the authentication credentials from the user.
3. The application server is configured to use an external LDAP-based authentication provider. Via the authentication provider the application server delegates the verification of the user credentials to the external authentication provider.
4. If authentication is successful, the application server verifies that the user is authorized to access the requested resource (authorization).



Architecture Approach
Example

A possible dynamic design for addressing the authentication requirement could look like this.

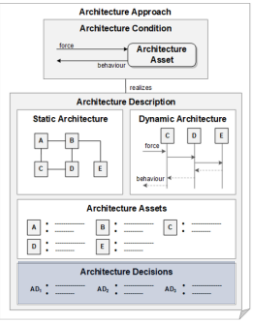
5. If authorization is successful, the user is authenticated and authorized to access the protected resource and the request (i.e., execution control) is forwarded to the web application. The web application may query the request object for information about the authenticated user. For this purpose, class HttpServletRequest defines a set of methods to access the user name or to check for membership in security roles. This allows the web application to implement fine grained services like enforcing additional web application internal security constraints.
6. For further application-specific user attributes, the web application can access the user database at any time via the database connector.



Architecture Approach
Example

A possible description of architecture assets may look like this.

Architecture Asset	Description
Http User Agent	A standard web browser that supports at least HTML 5 and basic authentication, either URL rewriting or cookie-based security session handling.
Application Server	A standard JEE application server that supports the JAAS API and the connection of LDAP authentication providers.
LDAP directory	The company's own LDAP directory service, with the correspondingly defined LDAP information tree - schematically defined as follows: Field / Data Type / Specifics user-account / String / PK user-password / String / encrypted
Web application	A custom developed web application that specifies its security requirements mainly declaratively. In addition, more complex authorisation rules are defined programmatically (i.e., by using the getRemoteUser() and isUserInRole() methods as well as by querying user-specific attributes in the application-specific user database.
User database	The application-specific user database in which further user attributes are managed. Permanent synchronisation is ensured between this database and the LDAP directory.



Architecture Approach
Example

A possible description of architecture assets may look like this.

Architecture Decision	Description
Adaption of standard JEE authentication provider mechanism	The standard mechanism for authentication providers - as specified in JEE - is used to delegate authentication to an external authentication provider. This approach allows easy changeability (e.g. switching to another LDAP directory in the company) and supports declarative security very efficiently.
Declarative and programmatic authorization	Programmatic security is only used very selectively to implement complex authorisation rules. The web application relies on the JAAS standard API and user attributes in the user database.
DB-specific replication mechanism to ensure synchronization between user database and LDAP directory	A proprietary database mechanism (DB replication) is used to guarantee synchrony between the user database and the LDAP directory. In case of replication conflicts, specially authorised users (i.e. administrators) are automatically informed.
...	

Architecture Approach
Exercise

*i.5-BSc_SWA_GamePlatform

Bibliography
Lecture

[Arnold 2021]
Arnold, Ingo, *Enterprise Architecture Function — a pattern language for planning, designing, and executing*, Springer Science and Business Media, Berlin Heidelberg, 2021

[Booch 2009]
Booch, Grady, Object-Oriented Analysis and Design with Applications, Pearson Education, Amsterdam, 2009

[Vogel, Arnold et al 2011]
Vogel, Oliver; Arnold, Ingo; Chugtai, Arif; Kehrer, Timo, *Software Architecture: A Comprehensive Framework and Guide for Practitioners*, Springer Science and Business Media, Berlin Heidelberg, 2011

Questions