INTRODUCTION TO SOFTWARE ARCHITECTURE

Software architectures provide high-level abstractions for representing structure, behavior, and key properties of a software system. These abstractions are useful in describing to various stakeholders complex, real-world problems in an understandable manner. Software architectures are described in terms of components, connectors, and configurations. Architectural components describe the computations and state of a system; connectors describe the rules of interaction among the components; finally, configurations define topologies of components and connectors.

Design frameworks

Architectural frameworks provide support for implementing, deploying, executing, and evolving software architectures. A framework is a skeletal group of software modules that may be tailored for building domain-specific applications, typically resulting in increased productivity and faster time-to-market.

Design pattern

In software engineering, a design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations. So patterns are formalized best practices that you must implement yourself in your application.

Design patterns can speed up the development process by providing tested, proven development paradigms. Reusing design patterns helps to prevent subtle issues that can cause major problems, and it also improves code readability for coders and architects who are familiar with the patterns.

Design patterns are composed of several sections. Of particular interest are the Structure, Participants, and Collaboration sections. Common design patterns can be improved over time, making them more robust than ad-hoc designs.

Design patterns were originally grouped into the categories: creational patterns, structural patterns, and behavioral patterns, and described using the concepts of delegation, aggregation, and consultation.

The documentation for a design pattern describes the context in which the pattern is used, the forces within the context that the pattern seeks to resolve, and the suggested solution.

Architecture description language (ADL)

ADLs are formal languages that describe or represent software architectures

An ADL must explicitly model components, connectors and their configurations

ADL must provide tool support for architecture-based development and evolution

There is a large variety in ADLs developed by either academic or industrial groups

Examples of some ADL's are ACME, ADML, Rapide, Wright, Unicon, Aseop, MetaH, AADL, Darwin, etc…

Positive elements of ADL

ADLs are a formal way of representing architecture

ADLs are intended to be both human and machine readable

ADLs support describing a system at a higher level than previously possible

ADLs permit analysis and assessment of architectures, for completeness, consistency, ambiguity, and performance

ADLs can support automatic generation of software systems

Negative elements of ADL

There is no universal agreement on what ADLs should represent, particularly as regards the behavior of the architecture

Representations currently in use are relatively difficult to parse and are not supported by commercial tools

Most ADLs tend to be very vertically optimized toward a particular kind of analysis

ADLs have in common:

Graphical syntax with often a textual form and a formally defined syntax and semantics

Features for modeling distributed systems

Little support for capturing design information, except through general purpose annotation mechanisms

Ability to represent hierarchical levels of detail including the creation of substructures by instantiating templates

ADLs differ in their ability to:

Handle real-time constructs, such as deadlines and task priorities, at the architectural level

Support the specification of different architectural styles. Few handle object oriented class inheritance or dynamic architectures

Support the analysis of the architecture

Handle different instantiations of the same architecture, in relation to product line architectures

Source : http://praveenthomasln.wordpress.com/category/s8-cs/