The role of models in software design

A model is an abstraction of entities and their relationships within a system. A model can be used in many contexts such as communication with stakeholders or generating work assignments. We are concerned in this article with the uses of a model as a aid in designing a system and will ignore the other uses.

By capturing the aspect of interest and ignoring others, the model allows the designer to analyze a proposed design of a system to determine how well the proposed design meets its requirements. It also allows the designer to modify the entities or relationships to determine whether an alternative arrangement will better satisfy the requirements.

Three types of models are relevant to software design: protypes, visualizations, and mathematical models.

Prototypes

A prototype is an implementation of some aspect of the final system. Its purpose is either to test a hypothesis or to gather feedback from stakeholders. The gathering of feedback is useful for designing the user interface or determining the functionality of a system, but this type of prototype is not in our scope. In our case, a prototype is constructed to test a hypothesis.

The hypothesis with respect to software architecture design relates to one or more quality attributes or achievement of functionality. Examples are:

- Performance. Will a specified collection of functionality execute within a performance budget?
- Performance. Is the overhead introduced by a proposed synchronization mechanism tolerable?
- Availability. Will this design support failure of one of the components?
- Security. Will this design resist a specific type of attack?
- Functionality. Will this algorithm produce the desired output?

The prototype is constructed in response to the hypothesis. The hypothesis determines the tests that the prototype will undergo and whether the prototype is successful under the conditions of the test. To the extent possible, the tests should be conducted in an environment that replicates the production environment of the final system. This is not always possible, but the environment may introduce variables that influence the results of the test. Consequently, it is ideal to replicate the production environment as closely as possible.

Visualization

A visualization displays entities and some relationships among them. It is important to understand that the design process is iterative and proceeds by refinement. This causes a design to become successively more detailed. The detail shown in a visualization will depend on the type of reasoning the designer wishes to perform at the current state of the design. For example, showing interface parameters between two modules will enable reasoning about type correctness. If the information to be exchanged between those two modules has not been defined, reasoning about type correctness is premature.

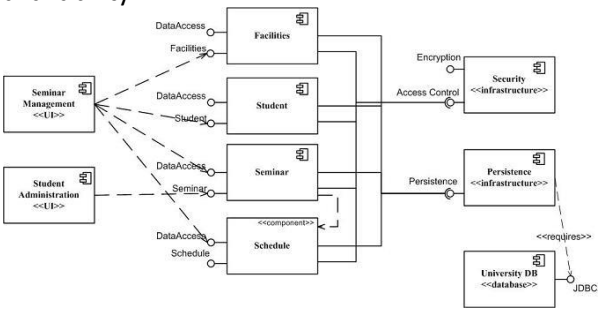
Showing too much detail in the visualization will add clutter but not enhance the reasoning ability of the designer.

Some common visualizations are:

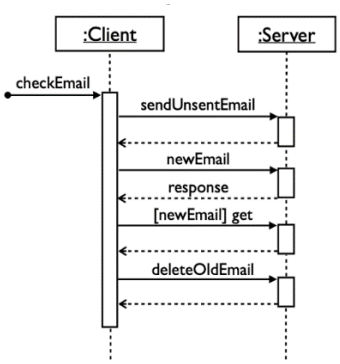
Context diagram. A context diagram displays an entity and other entities with which it communicates. It separates the responsibilities allocated to the chosen entity from those responsibilities allocated to other entities, and shows the interactions needed to accomplish the chosen entity's responsibilities. A context diagram can be used to reason about

- different classes of users (system administrators, normal users, etc),
- discovery mechanisms,
- protocols

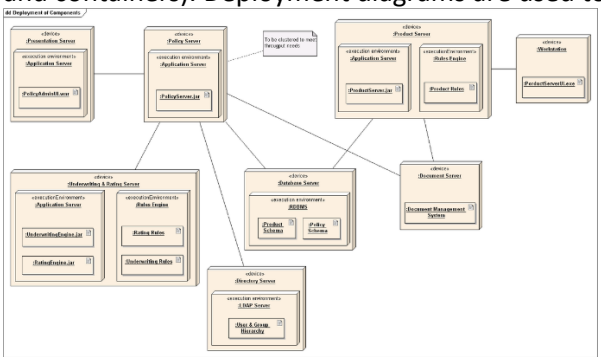
Component. a component is an identifiable part of a larger program or construction. Usually, a component provides a particular function or group of related functions. In programming design, a system is divided into components. A component can be independently deployable. Components can be used to reason about allocation of functionality (are the necessary functions included in the system) and together with sequence diagrams, about the satisfaction of use cases and other requirements. Components, together with deployment diagrams can be used to reason about performance and availability.



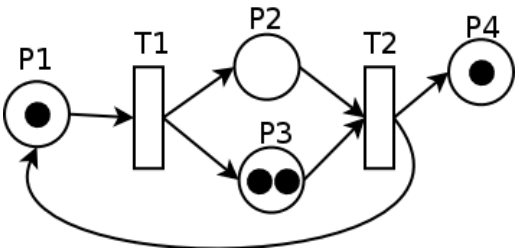
Sequence diagrams. A sequence diagram describes how—and in what order—a group of components work together. A sequence diagram can be used to reason about the achievement of use cases and other requirements.



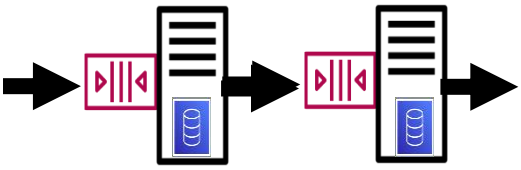
Deployment. Deployment views describe the mapping of components to elements of an environment in which it executes. The environment in such a view might be the hardware or the virtual hardware (VMs and containers). Deployment diagrams are used to reason about performance and availability.



Petri nets. A petri net is a formalism for modeling distributed systems. It can be used to reason about concurrent behavior.



Performance model. A performance model displays the significant aspects of resources consumed, contention for resources, and delays introduced by processing or physical limitations (such as speed, bandwidth of communications, access latency, etc.). It is used to reason about performance.



Design Structure Matrix. A Design Structure Matrix is a representation of the dependencies among modules. It allows the determination of various types of anti-patterns such as cyclic dependencies and layer bridging.

	A	B	C	D	E	F	G
Element A	A	1				1	
Element B		B		1			
Element C	1		C				1
Element D				D	1		
Element E		1			E	1	
Element F			1			F	
Element G	1				1		G

Mathematical (logical?) models

A mathematical model produces either a numeric or a yes/no output. Mathematical models can be derived from visualizations such as performance models or petri net models. Unfortunately, most modern systems are too complicated for the mathematical models to have closed form solutions. Simulation Is a technique to get values from a mathematical model without the necessity of explicitly solving them.

Another form of a mathematical model is model checking. This is a technique that relies on the symbolic execution of a program to find incorrect functionality. Most systems are too large for modeling checking to be of use, but it does have use in domains such as device drivers or microkernels.