Eindhoven University of Technology

Computer Science and Engineering

Architecture of Distributed Systems

Homework Assignment 1

Authors:
Jasper Selman
Ramon de Vaan

September 10, 2015
Eindhoven

1 Model 1 - Saas-Mode

1. The building blocks we see are the firewalls (conceptual). The purpose of these firewalls are obviously security related, such that the site is protected from malicious attacks. We also see load balancers and filtering. These are for work distribution and are also conceptual. The other components are all physical, namely the servers (application and database), databases and NAS devices. These are for storing and sharing the information on the site.

2. You see the red lines connecting components within a site to each other. This represent how components in a site are linked to each other and with which components they can communicate. You also see the green arrows, these connectors show how the main site communicates with the secondary site(s). All these connectors are conceptual.

3. We see a clear Design view with regards to the communication. You can clearly see which components should interact with other components. You can also clearly see the Deployment view. You can clearly see all the components in the view and it gives an overview of the services.

4. **Performance/Scalability** Yes, the use of the filtering and clustering give notion of the performance of the model. Also the scalable parts are shown in groups from 1, 2 to N.

   Availability/Reliability No.

   Security Yes, the use of firewalls and synchronous replications define the security.

   Maintainability No.

5. Yes, You can clearly see the flow throughout the system. You see load balancers, applications, filters, Clustered Databases, Severs and NAS devices and how these are linked to each other.

6. You can clearly see what all the components in the sites are and how they are connected. Different components have there own colouring and smaller parts of the model, belonging to each other are structured in a group (dashed lines). Different ways of communication are shown with different arrows. Not really clear what the model does precisely and what users/systems are using it.

2 Model 2 - VICSDA

1. The building blocks are system services, it is the logical model of a virtual community. Thus, all of these building blocks are conceptual.

2. The connectors are relationships between these building blocks, denoting in how the services use one another.

3. **Development view** The structure of the program/database is visualized, such that the programmer has a clear view of how the program should be organized, and in what way the different parts use/depend on one another.

   Process view It is to some extent visible how the different entities interact, via the text that is written next to the relations. This is useful for system integrators.

4. **Performance/Scalability** No.

   Availability/Reliability No.

   Security No.

   Maintainability Yes, the overview of the structure of the program is given, so the influence of altering/adding parts can be analysed.

5. No.

6. The model is very clear. It is using standard drawing conventions, and the lines can be clearly distinguished, even when crossing each other. You can easily see the relations between the different entities in the model. However, it does seem some information is missing. For the development view, it would be better to also include functions. It also seems like not all variables are declared.

3 Model 3 - AUTOSAR

1. The building blocks we see in this model are all the components in the RTE system. They are divided in several categories, namely pink blocks for drivers, green for abstractions, blue for services. We also see some grey blocks. These are external OS-applications or Micro-controllers. At last we also have blue blocks. These are "actors" in the error throw part. So they are Receivers of the errors or Senders. All these building blocks are conceptual.

2. The connectors we see are the arrows from certain parts to other parts indicating the errors which are thrown by the hardware and/or software. They are all conceptual.

3. We see a bit of an implementation view. All the parts which have to be implemented in the RTE are shown, but how they are implemented and what they do however is left free to the developer. We also see a bit of the Process view, mostly how the errors have to be handled and from which part they came.

4. **Performance/Scalability** Yes, they try to address the performance. Libraries are used and every part of the software, like communication, memory, I/O and on-board devices have their own abstraction, drivers and services. I do not know however how well this model scales when you enlarge it.

   Availability/Reliability Yes, a lot of errors are handled and pointed out.

   Security No.

   Maintainability No.

5. A little bit. We know that parts of the software, like memory, I/O, communication, etc. have their own services, abstractions and drivers, but we do not really know how this is distributed in the RTE so we do not know a lot.

6. I think this is not a very clear diagram, because it looks kind of messy at the start. This is because there are a lot of blocks and lines all over these blocks and on the right is some extra text. But when you look a little longer at the diagram you see that actually it is not that bad at all. All the blocks which have something in common, like OS, drivers, services etc. are coloured in the same way. Errors are always shown as a red lightning flash and the text on the right is the legend for the errors. Besides that I am missing the context of the model. I know that it has something to do with an automotive system from the URL and that a lot of errors are handled (or at least mentioned) but I do not really know what the system does.

4 Model 4 - Collaboration Diagram

1. The model has an actor, which is the customer, and several objects in the object-oriented programming language that the system is programmed in. All of these are conceptual.

2. The connections seem to be function calls on the objects. These are conceptual.

3. **Development view** The objects that are to be programmed, and the functions they should call are visible. This is useful for the programmer.

   Process view The model gives an overview of the information flow of the program via the function calls and the objects. System integrators and testers can see how the parts interact, and again, programmers know which functions to call.

4. **Performance/Scalability** No.

   Availability/Reliability No.

   Security No.

   Maintainability Yes, the model gives a clear description of the components in the system, and how they interact. Using the model, it is clear to see which components are affected when extending/updating the system.

5. No.

6. The model is pretty clear, as it is rather basic. It does not seem to have been drawn by conventions, but the actions and the components are clear enough to convey the meaning. This is also due to clear function and component names. However, in the case of the development view, the model leaves some things to be desired. There is no mention of class variables or other class functions apart from usages by other objects.

5 Model 5 - LinkedIn

1. The blocks in this model are the applications, the services, the "cloud", and all the databases and servers. The databases and servers are all physical and of course for storage of all the LinkedIn data. The applications, services and cloud are all conceptual. The applications represent the profiles of different users. The services represent what is used on LinkedIn, reading the news, communicate with other users, sharing something in a group, etc. The cloud is where all these web apps are used.

2. There is only one sort connector in this case, namely an arrow. Sometimes the arrow literally states what happens between the components (for example profile updates). Other times it is less obvious and you only know to components are connected and communicate. All the connectors are conceptual.

3. We see a clear Design view with regards to the communication. You can clearly see which components should interact with other components and if it should be a read/write action or something else. You can also clearly see the Deployment view. You can clearly see all the components in the view and it gives an overview of the services. You can also see implementation things in the building blocks and connectors, so you also see the implementation view.

4. **Performance/Scalability** Yes, you see a stash of databases, and that every service has its own database. It is shown that this is extendible by the ... etc.

   Availability/Reliability No.

   Security No.

   Maintainability Yes, since every component has its own database, you can easily change some of the services or add/remove one.

5. Yes, you can clearly see the distribution between services and databases and which application uses which service.

6. You can clearly see all the components in the sites and how they are connected. Different components have there own colouring and smaller parts of the model. Communication is clearly shown with arrows, sometimes the arrows also note what the communication is. Not really clear where the architecture starts (I guess in the web apps). But over all very structured and easy readable and understandable.

6 Model 6 - NHIN

1. The model shows several actors, like the user, provider, consumer, which are conceptual. Then there are several records and systems, such as the gateway, the public health system, the warehouse, which are physical. There is a distinction between a local organisation's existing system-collection, and the NHIN system-collection, which denotes that there is a data flow between an organisation and perhaps a larger (national?) database, to which more organisations can connect. In the centre is a large node, saying "connect", the use of which is a little unclear. It seems to be a central interface through which all of the systems in order to pass requests to other systems, which I assume to be conceptual.

2. The arrows represent the interactions between systems and other systems, and systems and users. The arrow denotes the direction of data flow.

3. **Logical view** The model denotes per user which actions/information are available to them.

   Process view The model shows the different components of the system, and the information flow between them. This is useful to system integrators.

4. **Performance/Scalability** No.

   Availability/Reliability No.

   Security Yes, the system makes notion of authorization and authentication.

   Maintainability No.

5. Yes, multiple systems are visible, a distinction is made between a local organisation and a public system. There are multiple machines all exchanging data with each other.

6. The model is moderately clear. At first hand, it is hard to notice what is going on, because of the multitude of arrows, all in close proximity. Additionally, where some of the components are very clearly labelled, it is unclear what the "connect" node in the centre entails. However, the actions do seem to make sense, and there is a clear view on which actors are present and how they can interact with the system.

7 Model 7 - Vending Machine

1. The states, sub-states and superstates. They are all conceptual and model in which state the vending machine is after an event has occurred.

2. The arrows represented by the events. These are all conceptual. They model an event which would happen in real life.

3. **Process View** System integrators, testers can see how the various parts interact. Developers see what the machine has to do when something happens. They also see what they have to implement to see how the state changes.

   Development View The state-chart is used to create the model by developers.

4. **Performance/Scalability** No.

   Availability/Reliability Yes, there are error methods when an error occurs.

   Security No.

   Maintainability Yes, the chart is very clear and easy to maintain. It is both useful for implementation and modification.

5. No.

6. It is a very clear diagram. In development view it is clear in which states the machine can be and how these states can be reached. It is however free for the programmer how this is implemented. It is also very clear whether something is a state or an action. These are the only two possibilities in the chart. Furthermore are the sub-states very useful to see which components communicate with each other.

8 Model 8 - Consumer Website

1. The model is divided into 4 sections, each of which contains several nodes that seem to model access to and the contents of a certain website, including backend. There is a Java servlet, a database, etc. All of these are conceptual.

2. The connections show both the flow of the program as the protocol used. They are all conceptual.

3. **Development view** Different subsystems are listed, along with access protocols. This is beneficial to programmers, to get an overview of the system to be implemented.

   Process view There is a notion of information flow in the connectors, along with the execution environment, as it lists the protocols used. This may come in handy for programmers, as they can look up how to program the protocols at hand.

4. **Performance/Scalability** No.

   Availability/Reliability No.

   Security No.

   Maintainability Yes, the main systems, subsystems, and interactions between these are all given.

5. Yes, there is a notion of distribution, as there are multiple tiers in the system.

6. The model is moderately clear. Instead of adhering to standard drawing conventions, the model defines its own shapes and lists them below. It is divided into different tiers, which adds to the clarity, but both the irregular shapes as the sheer number of different type components make the system unnecessarily complex. Additionally, there is a vast number of arrows, all differing in length, shape, and direction, which makes the model rather messy. On top of that, the arrows cross a lot of times. As such, it is difficult to understand the flow of the model.