CS446 / ECE452 Assignment 1: Software Architecture
Supplementary Information

by Chris Mennie (camennie@cs.uwaterloo.ca)
May 9 2003

I want to take the time to explain a little about what is expected of each group for this assignment. As you know, you're expected to come up with a software architecture for the IPPhone system you modelled in your SRS. But how exactly do you do this?

Architecture

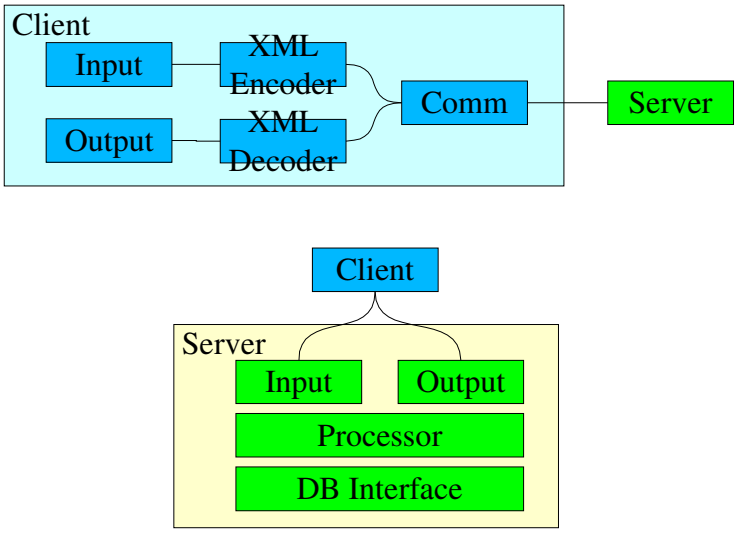
The most important parts are the architecture itself and the external interfaces. We're essentially expecting some nicely formatted diagrams with supporting explanation. I should be able to look at your diagrams and largely understand your system. To do this you want to start with a high level view and then continually "zoom in" into each component that needs more description. Figuring out the degree to which you do this is part of the problem. You want to balance having too little and too much information. So a single high level diagram with only a few boxes probably isn't enough, but I don't want to start giving details that are almost on par with functions/modules and the like. Let me start with a little example:

I'd start with this high-level view and explain what the Client, Server and DB were and roughly why they're connected the way they are. If I used a particular architectural style, or a hybrid of styles, I'd describe that here too. So in this case I'm mostly likely using a client/server style.

While we want you to describe your system in terms of Garlan and Shaw's terminology, we don't mean to constrain you into using just a single style. Most often a mixture of styles are used, and if you choose to do that, you simply have to explain it. Nor should you think that a particular style needs to be used throughout the entire system, within all components. What we're doing is dividing up the system into smaller and smaller parts, and explaining how each part works. At the high level I may be using a client/server style, but the Client could use a pipe and filter style, while the Server uses a layered style.

If more than one style seems appropriate for a component, try and explain why you chose the one you did. We're not asking for more than a paragraph or two of discussion for each instance. We want to understand why you chose what you did and see how well you understand the different architectures. Keep in mind that any architectural style can be used for anything; it's just a matter of how hard it is to use.

Going back to our example, I'll go down one level and diagram the Client and Server. The database I'll say will be implemented by some third-party SQL-based system that'll be figured out later (ie: when coding).



So this would be one of my lower-level views. I'd again explain what each of the boxes were and how they relate to each other. Next I'd see if any of the components needed more explanation and further expand them. In this example “Processor” seems pretty general and would most likely need more explanation. I'd continue this until I felt all the relevant information was covered.

One thing that's missing is what the lines between components mean. You can't assume that the notation you use will be immediately understood. If you're trying to convey control flow or data flow with arrows, then you need to explain that. If you're using some sort of funky made up notation you're going to need to explain that as well. No matter how obvious it may seem all your notation needs to be explained using legends with your diagrams and/or in a separate “Documents Conventions” section. They important thing in all of this is to be clear with what you mean. If the TAs have a hard time understanding you, you'll lose marks.

External Interface

Students are often confused by what exactly an external interface is. My take on it is that the external interface for a component is the part of it which interacts with something else, or with which something else interacts with it. So in your diagrams, chances are every line exposes at least one external interface. Whenever two parts of your system communicate with each other, the type of information being passed between them, and possibly how, needs to be explained.

This is also an appropriate place for an ER diagram of your database (if you have one), if it wasn't given in the architecture section.

Scenarios

Not only is coming up with an architecture important, but you need to show that it's reasonable and works. In the past I've had students tell me about how they had to change their architecture around later in the course because of poor decisions made on their first assignment.

It's ok to change your architecture later on if you need to. Part of the point of doing all this is to correct mistakes as early as possible. However you want to minimise the number of times you need to do this. A good way of doing this is by running through some scenarios and describing how they work with your proposed architecture.

Being able to convincingly describe a scenario is important. So if with my example above I just used the high-level diagram, I probably wouldn't convince anyone that my system really worked. I'd have to run through some scenarios using the lower-level diagrams/descriptions. There's a lot of flexibility with this. If the core of the scenario is with a single component (as seen from a higher level), I can mix the level in which I'm treating components. So some of them could be at a high level, while the ones that are more involved are at a lower level. It's up to you how to do this, or even if you do. We suggest you do do them, or you'll lose marks.

A good place to look for scenario ideas is in your SRS. You've already come up with a number of them there, so it makes sense to use them if you can.

Processes and Files

Each group is strongly encouraged to come up with an initial architecture for the process and directory structures of their system. What I mean by this is tell me what executables will be part of your system and how the overall architecture relates to it. So in my example, I would say that the Client and Server are each separate executables. Furthermore I could make the Input and Output subcomponents threads within each executable. This helps show how the system is connected together and gets you thinking about potential problems you might have.

It'd also be a good idea to give a directory and (possibly) file structure of your system. Nothing set in stone, but the idea is to give the developers an idea of how to divide up the system. Try an say what components of the architecture go where, on high and low-description levels, or whatever seems reasonable. Be clear and descriptive.

Physical Layout

For larger systems the physical layout is another important architectural concern. Describing on what system each executable runs, over what network they communicate, and so forth, is important in discovering problems with the architecture. For example, if you have hard real-time constraints and propose distributing your system over a slow network, this architectural view would quickly show the problem.

I recommend each group to include this. It should be very simple for everyone, and it'll reassure me that you understand the relationship between your phone processes and the physical phones. In particular I can't stress enough that conceptually your phone processes are running on the actual phone themselves. Yes, they will be a regular process on commando.math talking with the interface process, but we're pretending that this code is running on the phones themselves. I want everyone to understand this and architect your systems with this in mind.

Other Issues

There are a few other things I want to mention. First and foremost is to use your SRS. Your SRS is a model of your system. So you should be able to take your class diagram(s) and use them as a guide in

deciding what components to have in your system. You probably won't be able to come up with a nice mapping from your SRS's class diagrams and your architectural diagrams, but you should have something that's close.

Don't worry too much about page limits, insofar as going over them goes. If you take the advice given in this paper you'll easily fill 30-40 pages. It's not as bad as it sounds, honest. A lot of your document will be diagrams and tables. The 25 page limit worked well in the past, but those assignments tended to be done differently and I wasn't happy with how they tended to turn out. My hope is for better overall quality of assignments.

On the flip-side, if you're documents are under 15-20 pages you may want to think about anything that may be missing and/or check your formatting. I don't want to see 20+ pages of solid text. You may have a great architecture, but if your document is poorly formatted and painful to read that'll count against you. We aren't expecting professionally published documents, but that the same time we don't want to read poorly done ones either. I don't like saying this to 4th year students, but experience has shown that a gentle reminder is necessary for some.

One thing that really bugs me when reading papers and assignments is when references are used poorly. What I mean by this is if you're going to add references to your documents, then make sure you actually refer to them. A reference is almost useless if it shows up at the end of a document without being used before. I'm not concerned much with the layout of your references, but I want to see you use them. So if you have a reference listed in your references section, I want to see it referred to in the main body of the document.

Finally, I'm making available a document Troy and I wrote for a grad course in software architecture. It's sort of similar to what you'll be doing. You should take a look at it to get an idea of how to do scenarios and perhaps external interfaces. Just take ideas from it, don't follow it too closely. A link to it should be found on the page you got this document from.