CSCE 742 - Project Part 1 - System Selection

Due Date: Tuesday, September 18th, 11:59 PM (in PDF format, via Dropbox)

Your job, this semester, is to understand, document, and extend the architecture of a non-trivial real-world software system. This system must be a piece of implemented software or library that has a non-trivial architecture.

- You are free to choose any software project as long as the size of the code base for the project is substantial (at least 10,000 lines of code). If unsure, ask me if it is suitable.
- Given the wealth of open-source software, there are a number of readily available projects from which to choose, but it is also possible to use software that you have developed or are currently working on, as long as there are no confidentiality issues with your employer (code can be shared with your team and myself).

Since architectures usually consist of a number of abstraction layers, the size of the project, as long as it is "big enough", should not be a significant impediment. Your job is to explain the architectural layers of the project: the components and connectors (which are probably themselves classes) describing the primary interactions of the software. You will be required to explain the topmost layer of the architecture and at least one other layer of your choosing. The important thing is to be able to discern whether a detail is 'architectural', and this description depends on the layer being examined. It is very important not to get lost in the details of all of the source code. Your job is to use the existing documentation and the source code to build an accurate view of different layers within the architecture.

For example, **Eclipse** is a large open-source project consisting of thousands of files. At the top level, it has a well-defined and reasonably small architecture that describes the interactions of the top-level components in the system. Additionally, there are good documentation resources that can help you gain an understanding of the architecture. A smaller project, such as **jUnit** - with dozens of files - still has an interesting top-level architecture and other layers to explore.

Project Part 1

In this phase, you should choose a project to analyze. In a report, you must provide:

- A description of the system on interest.
- A scope section, explaining why this system is substantial enough to contain an architecture that can be studied.
- A Quality Attributes section, explaining which quality attributes are important to this system, and why they are important.
    - For each identified quality attribute, explain and justify why it is likely to be important to the overall success of this project.

This document does not have a minimum length. However, it is expected to be detailed and provide a complete treatment of the subject.

Example - NIMBUS

This is not a complete example, but is intended to give you a basic idea of what to include. Your submission should be more detailed (especially in the quality attributes section).

1. System Description

Assurance that a formal specification (system specification or software specification) possesses desired properties can be achieved through (1) manual inspections, (2) formal verification of the desired properties, or (3) simulation and testing of the specification. To achieve the high level of confidence in the correctness required in a safety-critical system, all three approaches must be used in concert.

The Critical Systems group has developed an environment, called NIMBUS, which provides support for all these activities:

- Manual inspections and visualization provide the specification team, customers, systems engineers, and regulatory representatives the means to informally verify that the behavior described formally in the specification matches the desired “real world” behavior of the system.
- Formal analysis is helpful to determine if the specification possesses desirable properties, for instance, if the specification is complete and consistent, and whether unsafe states are reachable
- Simulation and testing helps the analyst to evaluate and address poorly understood aspects of a design, improves communication between the different parties involved in development, allows empirical evaluation of design alternatives, and is one of the more feasible ways of validating a system’s behavior

2. System Scope

NIMBUS is architecturally interesting because it combines several different capabilities around a “blackboard” description of a specification. The simulation capability is event-based, multithreaded, and supports large scale, distributed simulation of specifications through communications over Microsoft’s distributed COM or OMG’s CORBA. For example, in a Rockwell Collins simulation of portions of a flight deck, 20 different simulations controlling flight guidance, flight management, instrumentation and the external environment were concurrently executed in a real-time simulation on two machines. Through the use of standard communications protocols, support is provided for simulations involving input files, other software tools (such as Microsoft Office or Simulink), and actual hardware devices.

Also, NIMBUS offers automated translation of specifications to other tools. These utilize a pipe and filter architecture and support fairly sophisticated transformations of the specification during translation. These translations allow a wide range of analysis capabilities. These capabilities include both checking of safety and liveness properties on specifications (a standard use of model-checking), and also new analysis techniques. NIMBUS supports automated structural test case generation from specifications up to an MC/DC level, allowing the toolset to significantly reduce the amount of effort required for verifying certain safety-critical systems.

3. Quality Attributes

High Performance

...

Extensible

Need to be able to easily add additional input/output data sources. Need to be able to add new translators for RSML-e specifications.

Maintainable

High rate of developer change is expected. New developers need to be able to make changes to the system.

Verifiable

To trust that it functions correctly, we need multiple independent cross-checks of functionality (simulation, formal analysis, etc.).