Chapter 1

Introduction

1.1 Document use

The document is meant for developers new to this project and for maintainers considering a change. It allows isolation of the partition that needs change. Finally, the document is for programmers looking for design guidelines.

1.2 Document structure

The document consists of the high level class diagrams, some sequence diagrams, some design patterns and finally the design guidelines and specific platform dependencies.

1.3 Document maintenance

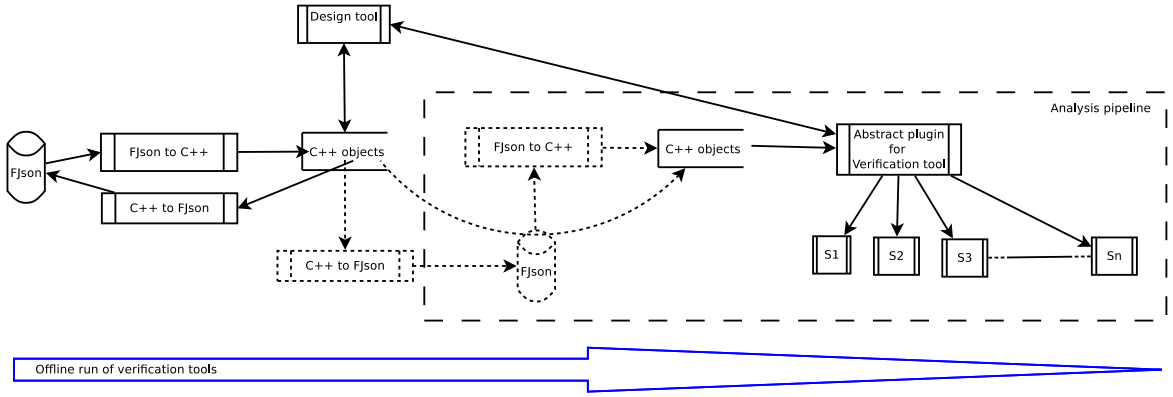
Use Umbrello to create and modify the architecture diagrams. Use the menu option export all diagrams as picture or the equivalent option for one diagram.

Make sure to create names without spaces so the exported filenames will also be valid. Even though most OS's can work around spaces they still are a pain in the neck.

Chapter 2

Architecture Overview

2.1 Partitioning



The complete tool consists of a verification pipeline and the graphical user interface where the research people can design a Network on Chip (NoC). Both partitions have differing use cases.

The use case for the verification pipeline is for verifying a model that needs a final check, or for verifying a model that is too big to verify completely during graphical edit cycles.

The use case for the graphical editor is for creating, modifying and intermediately verifying the NoC. The editor allows for selecting verification tools to run, and will verify parameters before each run.

The verification tools and the graphical toolkit communicatie through a plugin architecture.

2.2 Graphical Design Tool

The graphical editor of NoC consists of a toolkit containing the XMas primitives, the composites available in separate libraries, and the graphical window, and the commands to prepare and execute the verification tools.

The process of designing an NoC consists of alternating edit-verify cycles. Once the model is finished, all parameters per component are filled in, the verify tools are selected, the the controller will copy the current model and kick off the verification tools for an execution run.

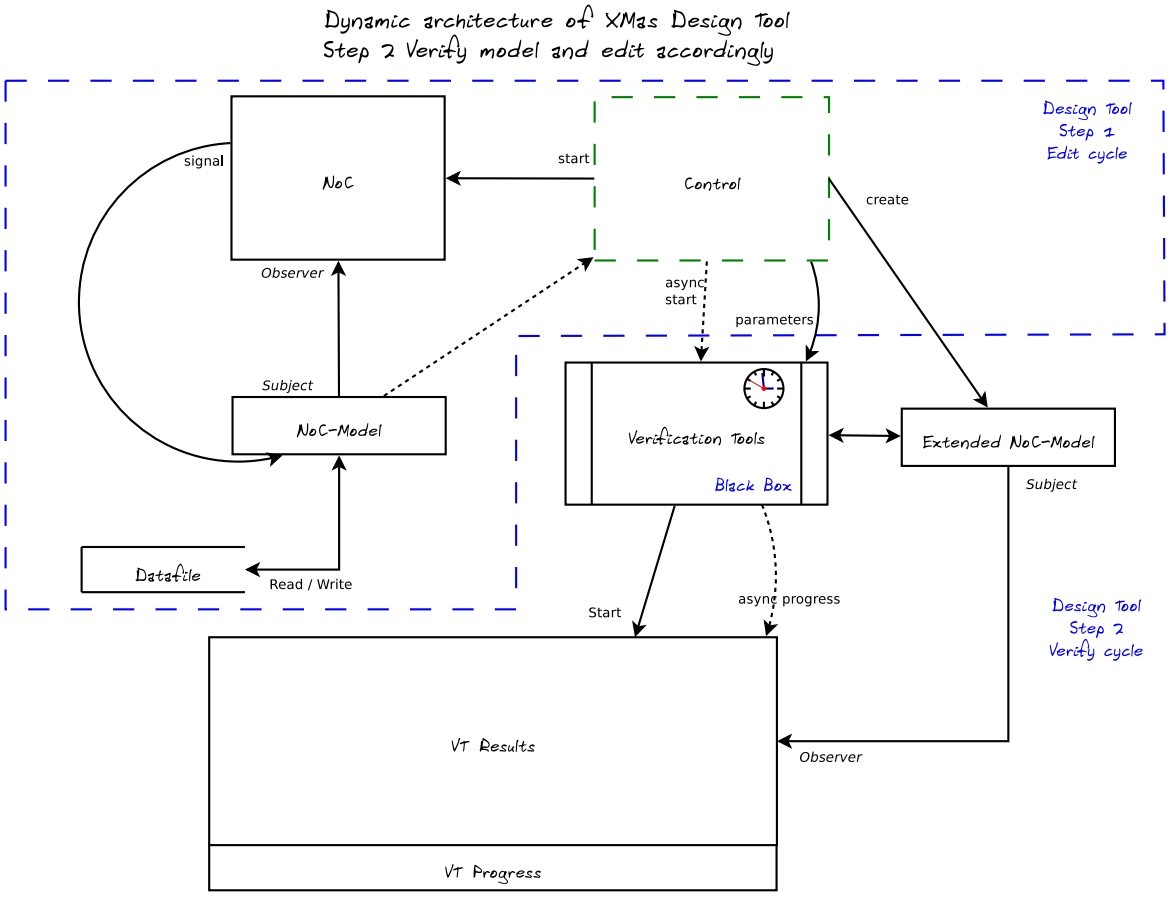


Figure 2.1: Dynamic process of editing and verifying

2.3 Verification pipeline

For the graphical editor the verification pipeline itself is not in scope. The connection to the verification pipeline is through the plugin architecture, where a plugin is wrapped in a C++ class that derives from VerificationTool.

Each verification tool must register with the VTController or the controller. The registration involves informing the VTController about function, and parameters of the verification tool at hand. The graphical editor will ask the VTController for available verification tools.

2.4 Layer architecture

2.5 Plugin communication

2.6 Primitive toolkit communication

2.7 Toolkit repository

2.8 Model repository

2.9 Architecture High Level Diagrams

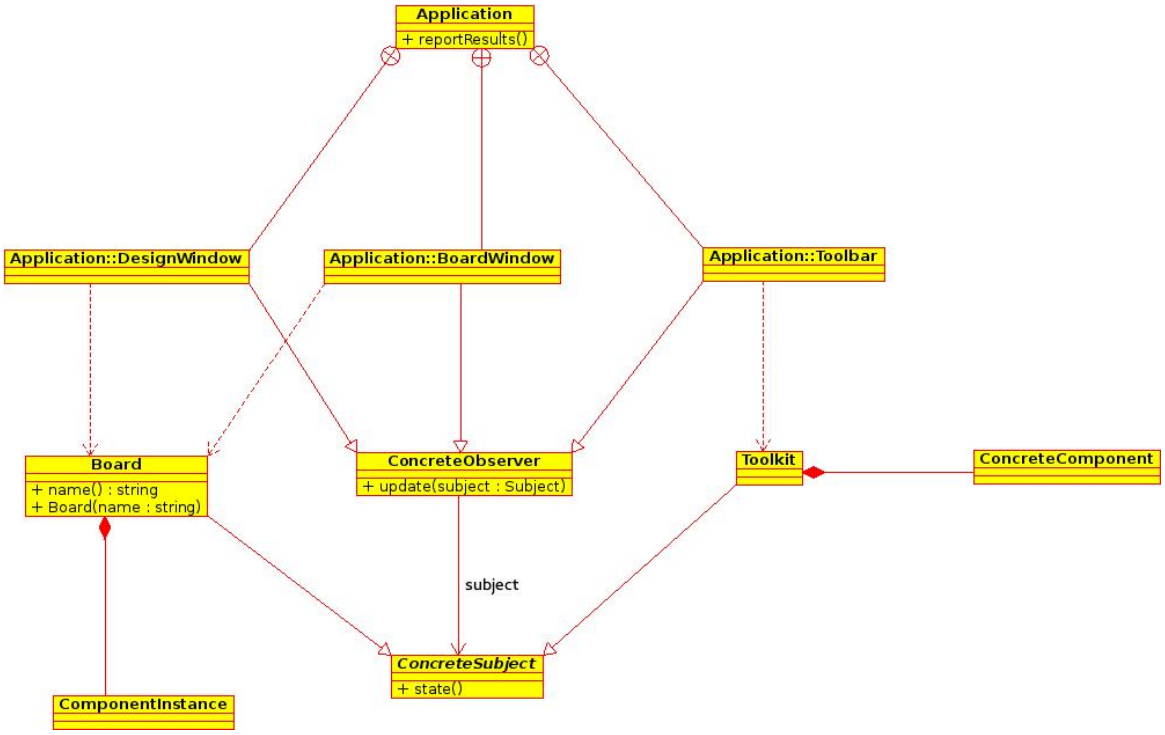


Figure 2.2: User Interface high level class diagram

Chapter 3

Architecture Design Guidelines

3.1 Observer pattern

We use our own observer pattern as described in the GoF. Figure 3.2 the pattern applies the principle. Any object wanting to observe one of the subjects, needs to derive from ConcreteObserver.

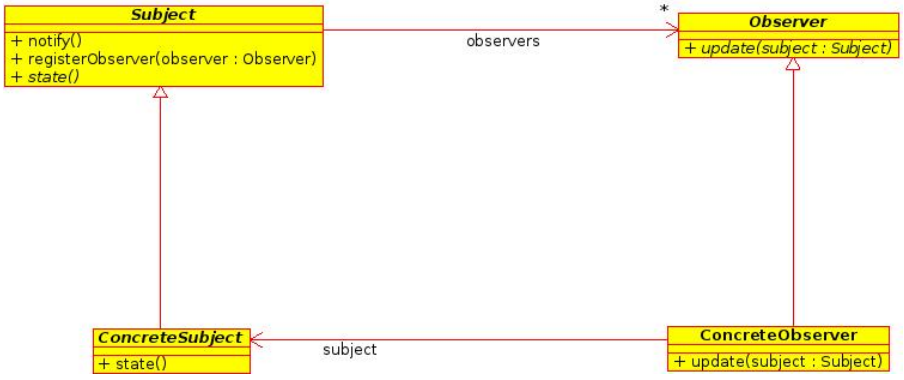


Figure 3.1: Observer pattern as described in GoF

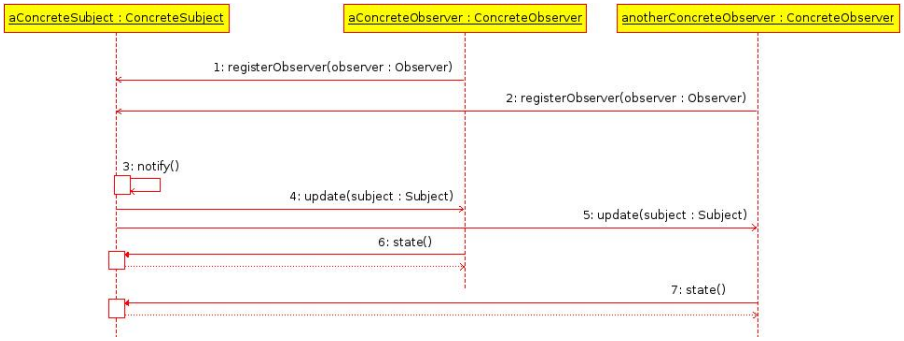


Figure 3.2: Observer pattern applied

3.2 Plugin construction

Any verification tool must override the ConcreteVerificationTool and register with the application in order to have the option of being executed on request.

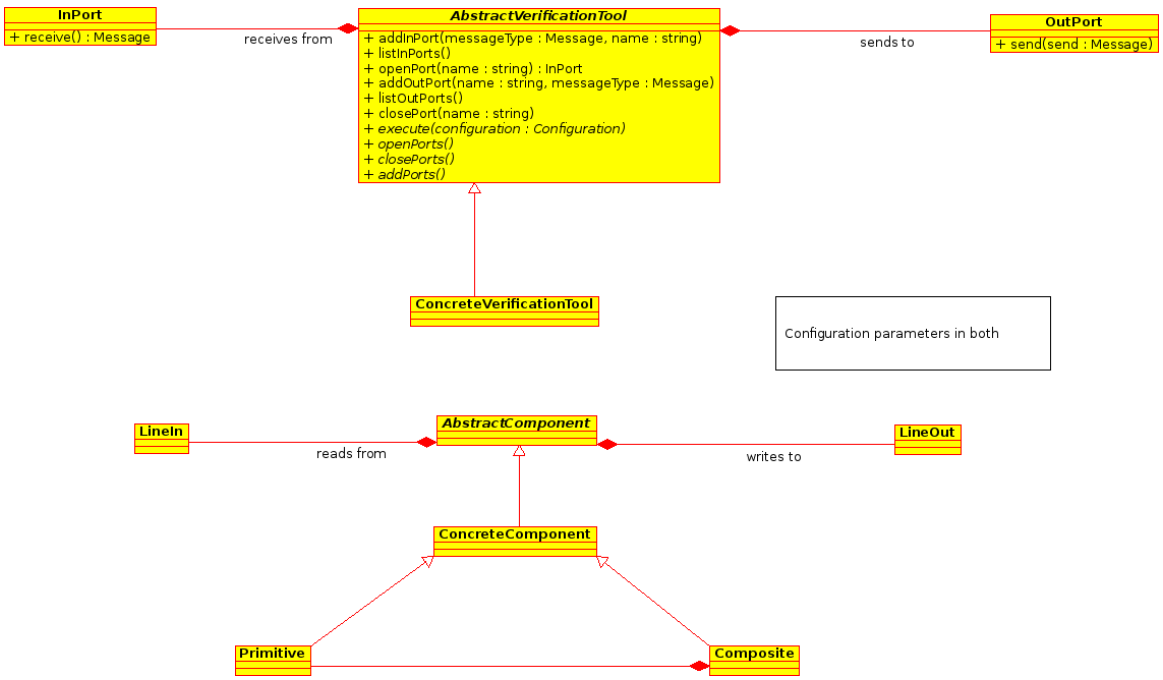


Figure 3.3: The plugin to build verification tools.

Chapter 4

Platform dependent constraints

4.1 Home of the software

4.2 Linux

4.2.1 Download and install

4.2.2 Regular build procedure

4.3 Macintosh

4.3.1 Download and install

4.3.2 Regular build procedure

4.3.3 Build limitations

4.4 MS Windows

4.4.1 Download and install

4.4.2 Regular build procedure