School of Electrical, Electronic & Computer Engineering


# Software requirements analysis for asynchronous circuit modelling and simulation tool

I. Poliakov, D. Sokolov, A. Yakovlev

Technical Report Series

NCL-EECE-MSD-TR-2006-118

September 2006

Contact:

Ivan.Poliakov@ncl.ac.uk
Danil.Sokolov@ncl.ac.uk
Alex.Yakovlev@ncl.ac.uk

Supported by EPSRC grant number EP/D053056/1

NCL-EECE-MSD-TR-2006-118


School of Electrical, Electronic & Computer Engineering,
Merz Court,
University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/

# Software requirements analysis for asynchronous circuit modelling and simulation tool

I. Poliakov, D. Sokolov, A. Yakovlev

September 2006

**Abstract**

This paper presents an overview, initial requirements analysis and software system architecture solution proposed for the development of a tool for modelling, simulation and analysis of concurrency models used in asynchronous circruit design.

## 1 Introduction

For a long time, token-flow based models, such as Petri nets or static dataflow models, have been used as a major tool for modelling and analysis of concurrent systems. However, while some of their inherent properties make them an obvious choice for describing data-path and control-path of asynchrnous circuits [2], they also somewhat lack flexibility and physical realism (particularly because token transitions are instant and tokens themselves are atomic). This calls for introduction of different modelling methods, which may be an extension to currently existing token-flow models or even a totally different concept.

Unfortunately, it is extremely hard, if not impossible, to analytically determine which model will be most appropriate for a particular modelling task, or to compare it with other models; it requires actual modelling to be done to discover practical advantages or disadvantages. This generates a need for a software tool which would provide means to integrate several different models into one consistent framework, thus enabling a circuit designer to effectively test and compare them.

## 2 Software vision

Because the software aims to provide powerful means of analysis and comparison of different modelling approaches, the two of its fundamental aspects are *extensibility* and *efficient user interface*. The software essentially provides a framework - a common, robust environment where different tools and different models co-exist and interact. This results in a complete, comprehensive view of available means for a circuit designer without a need to keep several similar tools which can have totally different interface

(some may be graphical, and some command line; they may even run on different platforms) and worry about their coordination.

Typical usage scenario will look like this: a designer starts a new model by selecting its type from the set of available model types. He is then presented with a blank workspace and the tools relevant to this particular model type are loaded automatically. The designer builds a test model by adding elements to the workspace, visually arranging them, setting some properties and creating necessary connections. At any time he is free to use any of the tools to check or analyse the model. After acquiring sufficient data about the new model by using the tools, the designer opens a previously saved model of another type without leaving the software. The type of model is recognized and a different set of tools is loaded, and the user is set to continue his work - all that in a single operation.

Because the software architecture is open and built from the start to provide extensibility, advanced users who write their own tools can spend little time to integrate their work with the system. This will benefit both the author of the tool and the users - the users won't have to do anything but copy a single file to insantly extend their tool set, and the author won't have to worry about user interface, inventing his own file format and so on, thus concentrating his efforts on the actual functionality of his new (great) tool.

# 3 Software requirements specification

## 3.1 Introduction

### 3.1.1 Purpose

This document describes initial draft of proposed software requirements.

### 3.1.2 Document Conventions

Any sections that are not clear as of yet are marked as TBD - "to be defined".

### 3.1.3 Intended Audience and Reading Suggestions

Anyone related to the project. Since this is yet a draft version, much of the following is subject to change. Any feedback on any part of the document is appreciated.

### 3.1.4 Project Scope

The software's primary goal is to provide a common, unified environment for both circuit design specialists and tool developers. The software will primarily be used by researchers at University of Newcastle-upon-Tyne, however it may also attract interest from other research institutions from similar area.

## 3.2 Overall Description

### 3.2.1 System Perspective

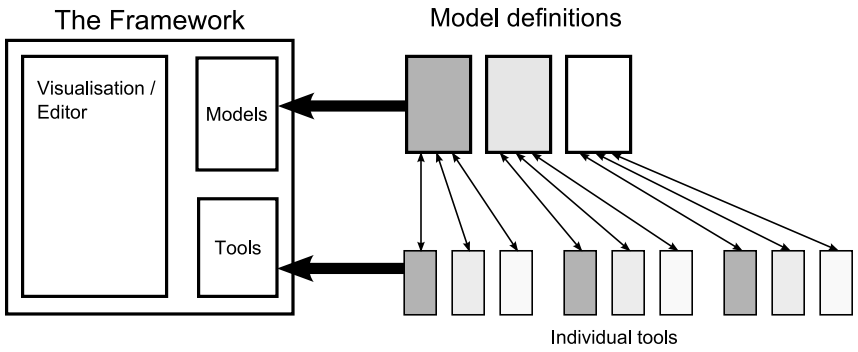


Figure 1: Component interaction

The software will be composed of a base framework and 2 major component types as shown in Figure 1. The framework will provide common services for all other components, such as schematics visualisation and editing facilities, and a GUI. Model definition components will define simulation behavior, visualisation features and a specification format for a particular model type. The tools will report supported model(s), and thus be automatically managed by the framework. The framework will also be responsible for translating visual model schematics into a specification (proposedly - an XML document) which will be used by tools to analyse the model.

### 3.2.2 System Features

- Exstensibility
- Cross-platform functionality
- Automatic tool integration and management
- Visual editor
- Hardware-accelerated visualisation allowing for hundreds of objects to be drawn or animated at interactive speed

More detailed explanation of the features is available in section 3.3

### 3.2.3 User Classes and Characteristics

Two major user classes can be identified: *a circuit designer*, who will primarily work using the visual editor and will concentrate on modelling and analysis of circuits, and *a*

*tool developer*, who is mainly interested in practical implementation of his theoretical findings and doesn't want to bother with development of his own visualisation/editing system.

#### 3.2.4 Operating Environment

The software will run on most 32- and 64-bit modern systems. A multi-platform solution is proposed, based on one of the 3rd party cross-platform GUI toolkits, Java, or .Net technology, so that the software will support most of the major operating systems (main focus being Windows and Linux systems). An OpenGL hardware acceleration graphics card will also be required, but since the software will only be using basic graphical functions, a very wide range of hardware will be supported.

#### 3.2.5 Design and Implementation Constraints

TBD

#### 3.2.6 User Documentation

TBD

#### 3.2.7 Assumptions and Dependencies

The software may be dependent on a number of third-party components, specifically a cross-platform GUI toolkit and an XML parser, however these dependencies are not final. All such components are assumed to be available under GPL or other (free of charge) license. An alternative approach is to use Java or .Net technology, which both have inherent support for GUI and XML; this will alleviate many potential problems with third-party software, however at the price of considerable performance penalty.

### 3.3 System Features

#### 3.3.1 Extensibility

**Description and Priority** Priority: Medium, Risk: Low to High (depending on solution chosen)

The system will be able to be extended with additional model types and arbitrary number of tools.

**Functional Requirements** An underlying cross-platform plug-in mechanism must be designed to implement this functionality. This presents a considerable complication in case if C++ solution is chosen, because plug-in loading system will have to be written for each supported OS type separately. However, if either .Net or Java is to be used, this becomes a trivial problem, since both languages use intermediate byte-code form for executables, which is freely interchangeable between platforms, and also provide powerful reflection mechanisms ([1, 4]) which facilitate plug-in handling even further.

### 3.3.2 Cross-platform functionality

**Description and Priority** Priority: Medium, Risk: Low to Medium
The system will at least run on Windows and Linux platforms.

**Functional Requirements** This feature requires that the system implementation is indepentent on platform-specific components. For C++ solution, this will require number of third-party cross-platform components, which is potentially risky. For Java/.Net solution, this is already an inherent feature and does not require any additional work whatsoever.

### 3.3.3 Automatic tool integration and management

**Description and Priority** Priority: Low, Risk: Low
The system will automatically gather information from available plug-in tools and properly arrange them on the interface.

**Functional Requirements** This feature will depend on chosen extensibility solution, but in any case does not present any considerable complications, and requires minor GUI coding.

### 3.3.4 Visual editor

**Description and Priority** Priority: High, Risk: Low
The system will provide standard functions for management of graphical objects, such as drawing, moving, scaling, detecting selection, etc. The editor will also be able to transparently export data at any time in a format understandable by tools.

**Functional Requirements** This subsystem requires definition of standart graphical objects set (and possibly a way of extending it); developing means of drawing them either using direct framebuffer access or a 3D API; developing means of detecting user-generated events (mouse over a certain object, click on a certain object, dragging of an object, etc.) and properly responding to them; object hierarchy management subsystem (for grouping and connecting); font-drawing subsystem (to draw zoomable, rotateable labels); an uniform format of data interchange between editor and tools is also to be designed.

Most of these problems were examined in [3], and efficient solutions were proposed which can be re-used.

### 3.3.5 Hardware-accelerated visualisation

**Description and Priority** Priority: Medium, Risk: Low
The system will use a hardware-accelerated drawing method allowing for animation to be used even with a large number of objects.

**Functional Requirements** OpenGL is currently the only available cross-platform hardware-accelerated graphics API, so it will definetly be used. Hence, an interface to OpenGL is needed for the system; fortunately it is available for almost every programming language and platform.

## 3.4 External Interface Requirements

### 3.4.1 User Interface

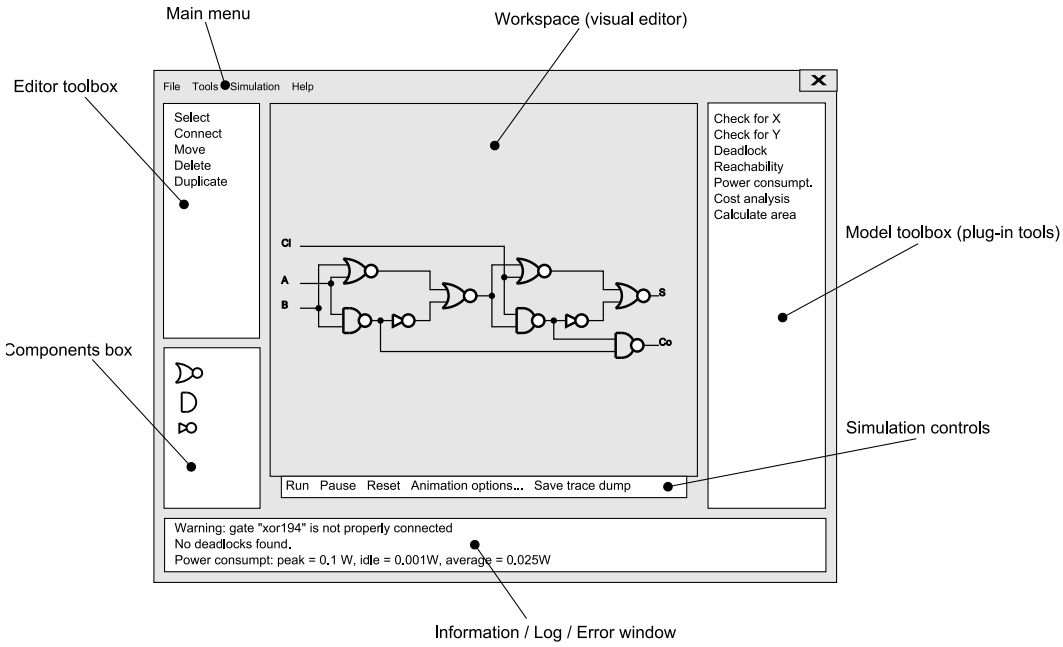


Figure 2: Main application window

User interface will be composed of several main components, as shown in figure 2. *The workspace* is the main editor interface, through which the user is able to interact with model elements. It is also the main visualisation window where all simulation-related animation will take place. *The main menu* is a standard menu, exposing most of the software functionality. *The editor toolbox* is a box with controls related to editing of the model, such as selecting, moving, grouping, connecting objects. *The components box* will hold a model specification-provided set of components, which may represent gates of a circuit, generic state nodes, transitions, etc. *The model toolbox* will include automatically generated tool set, constructed from plug-in tools by selecting only those relevant to the current model. *The simulation controls* will provide fast access to the simulation-related functionality. Finally, *the log window* will hold various messages from all of the software components: warnings, errors, notifications, messages.

The interface will be primarily mouse-controlled, but frequently used functions will have keyboard shortcuts.

#### 3.4.2 Software Interfaces

The software will use following third-party software components:

- C++ solution: OpenGL API (version 2.1), GTK+ GUI toolkit (version 2.8) OR wxWidgets GUI toolkit (version 2.7.0), Xerces XML parser (version 2.7.0)
- Java solution: Java OpenGL (JOGL) (version JSR-231 1.0.0)
- .Net solution: Tao Framework (version 1.3.0)

### 3.5 Other Nonfunctional Requirements

#### 3.5.1 Performance Requirements

TBD

#### 3.5.2 Safety Requirements

N/A

#### 3.5.3 Security Requirements

N/A

## 4 Conclusions

Considering aforesaid, following conclusions can be made:

- C++ implementation is a risky approach, because it can lead to unforeseen complications;
- For a cross-platform solution, a choice between Java or .Net (Mono implementation) should be made;
- For a single-platform (Windows) solution, a .Net (Microsoft implementation) is the best choice.

## References

[1] *MSDN Library: Reflection Overview*.

[2] Alex Yakovlev. Albert M. Koelmans. Petri nets and Digital Hardware Design Lectures on Petri Nets II: Applications. Advances in Petri Nets, Lecture Notes. *Computer Science*, 1492:154–236, 1998.

[3] Ivan Poliakov. Development of real-time computer 3d-graphics visualization system. Master's thesis, Kyrgyz-Russian Slavic University, 2005.

[4] R. Morrison & D.W. Stemple. *Software - Practice and Experience*, chapter Linguistic Reflection in Java, pages 1045–1077. John Wiley & Sons, 1998.