VT - XMD integration

ABI team 33
(Stefan Versluys, Guus Bonnema, Jeroen Kleijn)

20/02/2015

Abstract

* Why did we write this document? * What does it cover? * How to read it?

This document explains why and in what manner the team decided to decouple the designer subsystem XMD and the VT subsystem. The contents documents the background, the problem, the decison and the motivation for the decision, and finally the detailed considerations that lead to the decision.

The final structure of the subsystems (the diagram) will be part of the system documentation.

Modification History		
Stefan Versluys	17-02-2015	Initial discussion formulation
Guus Bonnema	20-02-2015	Transformation to decision document

Contents

1 Problem background and description

Background During the planning phase of the project, the customer requested that the designer be integrated with the verification tools in a way that should be transparant to the user. Seamless integration should be the appearance, whether or not the technical integration would be seamless.

For that reason the team experimented with different forms of integration and finally reached the stage where we could formulate the alternatives that we deem viable and decide.

This document describes in summary which alternatives we compared, what we decided and what our motivation was.

Description and alternatives Once we decided to switch to QT as the user interface toolkit, we were confronted with 3 different technologies that QT uses to implement a user interface: GraphicsView, QTQUICK 1 and QTQUICK 2. The decision at the time was to use QTQUICK 2 and the related replacement for javascript QT (referred to as QT 2).

Immediately after this decision we started studying the interface between QT and C++ in order to interface the XMD subsystem (the designer) with the VT subsystem (verification tools). The experimentation that followed lead us to three alternative implementations that we partially implemented before reaching the conclusion formulated below.

We considered the following alternatives:

C++ parser In this alternative we communicate from the designer to a mapper class that catches each signal for creation of a component on the designer and translates it into the currently existing structure (`xmas.h`). This implies complete and integral coupling of the XMD subsystem and the VT subsystem through this mapper class. Also, the flat json designed in previous activities were to be amended for the new designer capabilities including graphical data requirements.

Partial C++ parser and javascript parser This alternative implies two structures that contain the network design: one for the designer and one for the VT subsystem which is the existing `fjson` structure. This implies translation of one datastructure to another from the mapper class but limits the exposure of graphical needs of the designer to the verification structure.

Full javascript parser This alternatives implies two structures as well, but uses the `fjson` text stream as interface between the two subsystems. This implies that there is no other coupling between the XMD subsystem and the VT subsystem and no modifications to the VT system are necessary other than those needed to run and compile on all specified platforms.

2 Decision and motivation

Due to the massive advantages that the loose coupling of the alternative *Full javascript parser* offers, we decided in favour of this alternative.

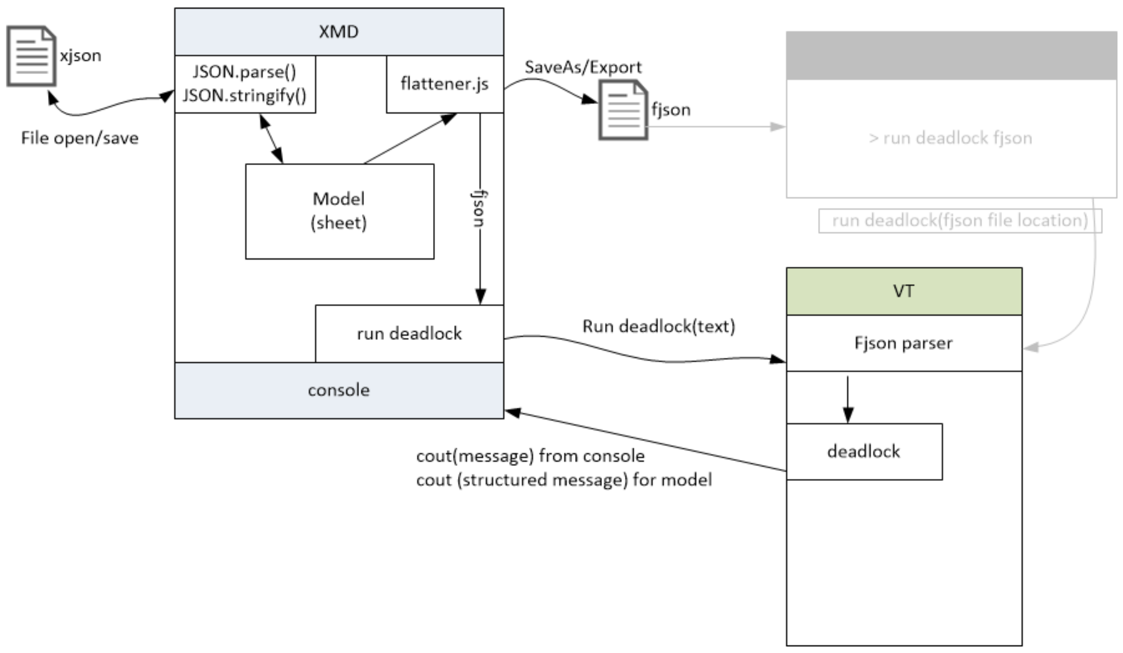

Figure 1: Diagram for *Full javascript parser*

3 Implications for future changes

The decision implies that VT development is relatively independant of designer development. Any changes to the interface of the `fjson` structure impacts both subsystems, but all other changes are independent.

Additionally we still need a mechanism to start and stop verification tools, track the error responses and we need to couple the verification tools executables into the designer without creating a development dependency.

4 Details of the original discussion for VT - XMD integration

Subject for discussion This discussion concerns the way verification tools (VT) can be integrated into or interact with the Xmas Model Designer (XMD). There are several options but not clear which is the best, because of conflicts in their requirements & specifications, additional risks, increase of complexity and fade of basic rules concerning separation.

VT requirements & specification which affect integration decision:

- The VTs are written in pure c++ because they can run on platforms without any specific software tools.

- VTs their mayor key is performance.
- The VT parser can handle a flat json structured file.
- The VT parser cannot handle composites.
- The VT parser feeds the VT model and has no GUI properties.
- The VTs can be started and get their input via the command prompt. (standalone)
- The VTs can be started and get their input via the designer tool.
- The VTs structure is complex because its focus is performance and not maintainability.
- The VTs give feedback via the standard output

XMD requirements & specification which affect integration decision:

- XMD is written in qml/c++ with Qt classes which gains development speed, fancy GUI, easy scripting.
- XMD must handle composites.
- XMD its major key is maintainability
- Composites can be inside other composites and implies a hierarchical structure
- XMD needs a hierarchical json parser
- XMD needs GUI specific component properties like x,y,orientation
- XMD doesnt need VT specific component model data structures.
- XMD must be pure GUI and doesnt need to know about complex VT structures
- Ideally VT integration in XMD must be purely interaction, as if they are one thing for the XMD user but technically totally separated.
- XMD must show the VTs feedback in the console (normal text message)
- XMD can show the VTs result in the model (structured text message)

Team:

- VTs are out of scope.
- Modifying VTs parser and the structure to pass additional data is a risk:
  - planning : complexity gains time?
  - focus is XMD : loss of VT requirements/correctness?
  - Because of OU changes, Bernard cannot be part of the team as developper.
  - XMD interaction is done through the whole structure and makes VT and XMD very dependent from each other.
  - Because of this dependence a change at the VT side can harm XMD and vice versa and breaks the maintainability rule. The same holds if changing XMD
- Using the VT parser and underlaying data structure extends the complexity into XMD which conflicts with its maintainability requirement.
- Bernard: use only one data model, easier to maintain.
- Bernard: VT has a parser you can use it.
- Freeks concern of diving too deep into the VTs because of too many risks and out of our scope. (see email)

5 Options

VT pure c++ parser only:

- +/+ only one file type for XMD and VT
- +/- one parser and model : not an advantage see remarks (*)
- -/- flattening must be done at VT side in pure c++
- -/- XMD depends on the VT parser and complex VT structure for mapping.
- -/- increased exposure to change issues between VT and XMD.
- -/- VT structure mixed with GUI parameters (x,y,orientation,scale,..)
- -/- VT parser and structure modification = risk + out of scope
- -/- current fjson files not compatible anymore = loss of test references
- -/- composites need to be handled/flattened in VT parser

- -/- handling composite file locations, XMD vs. VT standalone on a pure platform
- -/- id references of components or ports can become inconsistent.

(*)The advantage of having a parser already is not an issue because in javascript a parser is one line of code. The disadvantage of having to modify two parsers /models isnt an issue because in javascript, modifying is just adding a key/value and simple mapping. If VT and XMD are independent, a modification of the XMD parser doesnt lead to a VT parser modification if its just about a GUI parameter, flattening can skip these.

XMD Javascript parser + VT dependent:

- +/+ XMD can have its own json structure with GUI specific parameters
- +/+ flattening can be done outside VT, no need to use pure c++
- +/+ fjson stays compatible
- +/+ no need to modify VT to handle composites (parsing+flattening)
- +/+ composite handling not necessary in VT for this project.
- -/- two file types , one hierarchical json and one fjson. Same as the current WickedXmas.
- -/- mapping for flattening is still done via VT parser/data, meaning complex VT data structures into flattener.
- -/- a designer must hand over flat files to a verifier using VT as stand alone.

XMD Javascript parser + VT independent:

- same advantages as previous
- +/+ flattening done totally independant in e.g. Javascript , no need to know VT data structures. No mapping
- +/+ XMD and VT are completly independent, only the fjson structure is common.
- +/- flat structure in XMD must match VT flat structure , but in javascript very small modification. Much easier than adopt mapping to complex datastructures.
- +/+ flattening in the design tool can be used to see if there are no cycles before these are send to de VTs
- +/+ very low risk, only the fjson structure must match
- +/+ no need of complex mapping code in designer, pure compact, robust and easy javascript

- -/- two file types xmd + fjson
- -/- XMD needs and SaveAs or export function to fjson , so VTs can run models as standalone

Figure 1 is a setup where VT and XMD are totally independent but can interact with each other as if they are one thing. By this XMD can keep its maintainability and apart from minor changes VT stays untouched. Teh fjson structure is the only key they have in common.