verum

Alarm System Example

ASD:Suite

Copyright © 2013 Verum Software Technologies B.V.

ASD is licensed under EU Patent 1749264, Hong Kong Patent HK1104100 and US Patent 8370798

All rights are reserved. No part of this publication may be reproduced in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the copyright owner.

TABLE OF CONTENTS

Copyright © 2013 Verum Software Technologies B.V.

1 INTRODUCTION

1.1 PURPOSE

In this document we describe a burglar Alarm System built with the ASD:Suite. The Alarm System is used to demonstrate the power of automatically verifying a complete design with the ASD:Suite to make behavioural correct software.

This document describes a typical ASD Development cycle:

1. Edit an ASD model
2. Verify a model, change and re-verify it
3. Generate code
4. Download ASD:Runtime (once)
5. Build
6. Validate

Important note: The ASD:Suite Alarm System example includes a design defect. The example files are intended to be used as a tutorial to show new ASD:Suite users how to use ASD:Suite verification to find and resolve such design defects.

1.2 USEFUL LINKS

For more information about designing and completely verifying ASD:Suite models see the ASD:Suite User Manual and the ASD:Suite Visual Verification Guide.

You can find additional information, FAQ's, How-To videos, and a User Forum on Verum's ASD:Suite Community site.

The ASD:Suite 'Quick Start' video demonstrates much of what has been described in this document.

When you want to start creating your own ASD models, we recommend having a look at our training materials, explaining the concepts of ASD and including a number of exercises that lead you step by step through the process of creating a software component using the ASD:Suite.

Additional examples of ASD models for various domain applications can also be found in the E-Examples section on our community site.

1.3 DOWNLOAD AND INSTALLATION

The Alarm System Example is packaged as a zip file and can be found on the Verum Community site. Download it and extract all files into a local folder (e.g. …\My Documents\My ASD Models\Examples).

This zip file contains the ASD models, a copyright text file, two executable versions of the Alarm System, a 'code' sub-directory and the pdf description of the Alarm System example (this document).

Copyright © 2013 Verum Software Technologies B.V.

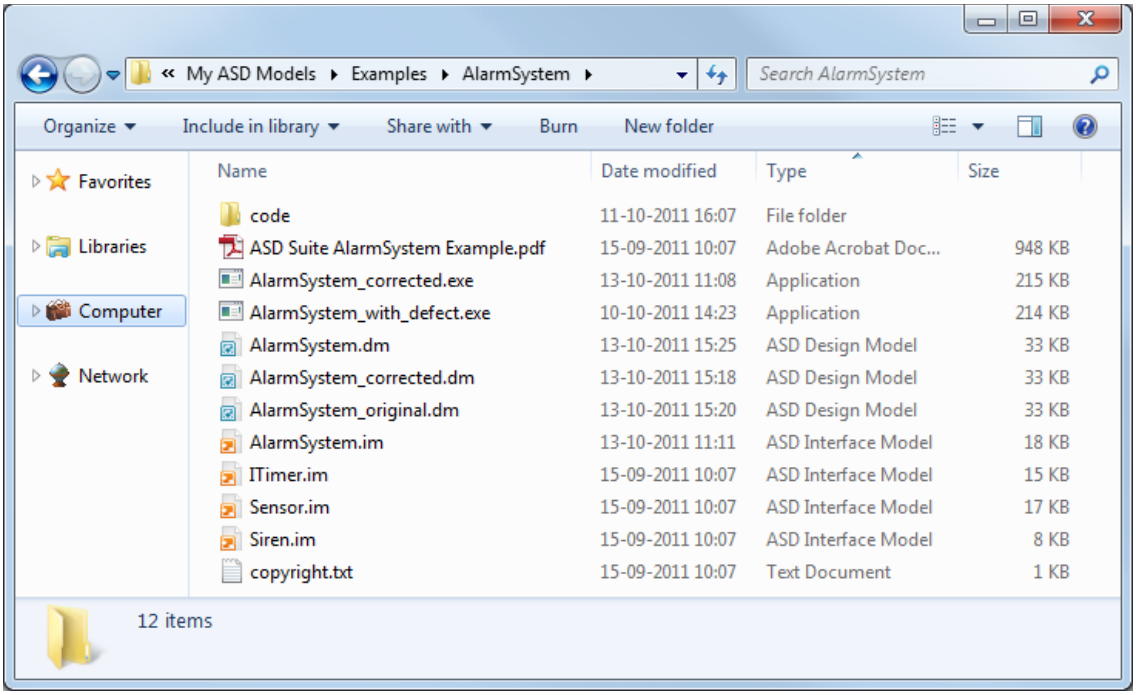


Figure 1-1 Top level directory structure of the Alarm System example.

The ‘code’ sub-directory contains source and project files code for Java, C++, and C#. The example is distributed without generated code and ASD:Runtime, necessary to build and run the project. How to obtain these is described in sections 3.3, *Generating Behavioural Correct* Code and 4.1, *Downloading the ASD:Runtime*.

Copyright © 2013 Verum Software Technologies B.V.

2 THE ALARM SYSTEM

The Alarm System consists of the following ASD:Suite models:

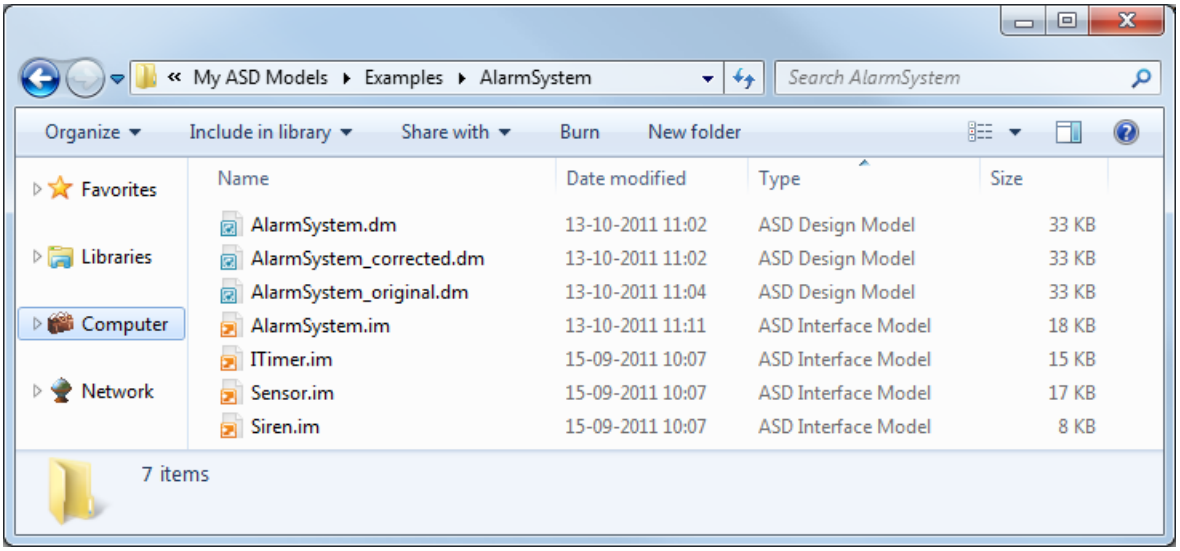


Figure 2-1 Design models and Interface models of the Alarm System.

The 'AlarmSystem.dm' design model is the main component of the Alarm System. It implements the service as specified by the 'AlarmSystem.im' interface model. The 'AlarmSystem.dm' design model makes use of the services provided by the 'Sensor.im' and 'Siren.im' interface models.

This is the context diagram of the AlarmSystem component:

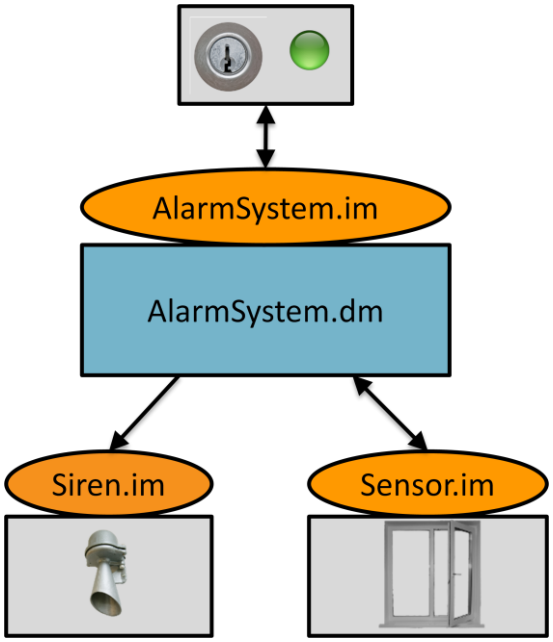


Figure 2-2 Context diagram of the AlarmSystem.dm design model.

Copyright © 2013 Verum Software Technologies B.V.

Part of the code for Java, C++ and C# is a handwritten GUI which creates the Alarm System console. The console shows a key for switching the Alarm System on and off, and a LED light for indicating the Alarm System status. When the Alarm System is switched on, a window sensor button is displayed on the console. A user can trip the sensor, which makes the Alarm System activate the siren after 5 seconds. When the siren is activated, it displays a siren icon and plays a siren sound.

2.1 RUNNING THE ALARM SYSTEM EXAMPLE

The Alarm System is operated via a graphical console. The console contains the following:

A. Key switch - This is a toggle switch. Clicking on the key button will switch the Alarm System on. Clicking on the key button again will switch the Alarm System off.

B. LED light – The colour indicates Alarm System status:
   - Green - the Alarm System is switched off.
   - Yellow - the Alarm System is switched on
   - Red - Alarm System has been tripped by a sensor.

C. Window sensor – A window image will appear when the Alarm is switched on. Clicking on the window image trips the sensor.

D. Siren – This image will appear if the Alarm System is not switched off within 5 seconds after the window sensor has been tripped. A siren noise also plays when the image appears.

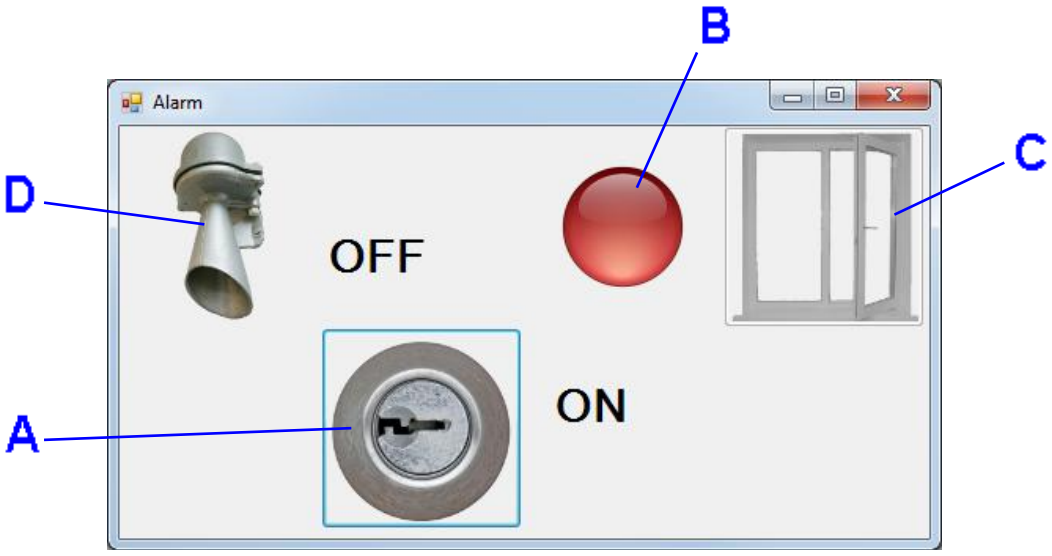


Figure 2-3 Alarm System console with activated siren.

Note: Actually the sensor and siren are not part of the console, they are simulated on the GUI for convenience.

2.2 ALARM SYSTEM SOFTWARE DEFECT

The Alarm System Example comes with two executables:

Copyright © 2013 Verum Software Technologies B.V.

- 'AlarmSystem_with_defect.exe', this executable is generated from the ASD models without being fully verified, and it still contains a design defect, to demonstrate the effect of no using formal verification.
- 'AlarmSystem_corrected.exe', this executable is fully verified and is behavioural correct, to be able to compare the alarm system behaviour with and without defect.

To see the defect appearing in the Alarm System, do the following:

- Run the 'AlarmSystem_with_defect.exe'.
- Switch the Alarm System on by clicking on the key switch. The LED light will turn yellow.
- Click on the window image to trip the sensor and immediately click on the key switch to switch the Alarm System off.

When tripping the sensor and then switching off the Alarm System happen together too quickly, the Alarm System gets into an undefined, illegal state and an assertion failure will be triggered:

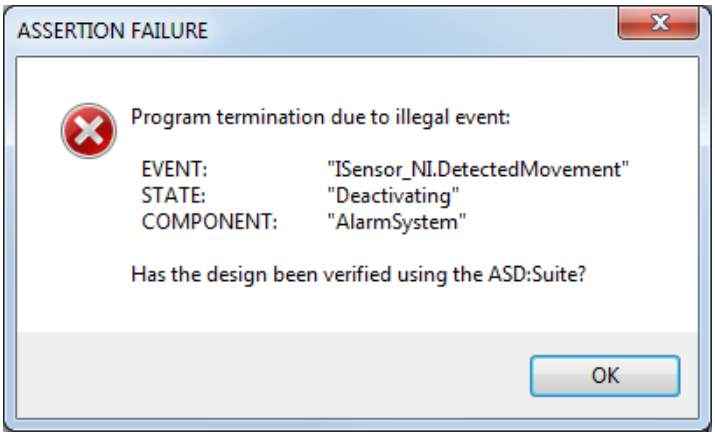


Figure 2-4 Assertion failure due to undefined, illegal state.

This problem is caused by a race condition that occurs between detecting that the sensor has been tripped and switching the Alarm System off. This is explained in more detail in the next section.

Copyright © 2013 Verum Software Technologies B.V.

3 VERIFICATION AND CODE GENERATION

Conventionally, testing is used to increase the quality of software. However, during testing not all execution scenarios can be tested. Some test case scenarios are difficult to replicate since they depend on timing and the relative order in which events can occur. It is quite common that the race condition illustrated in the Alarm System example, or other concurrency related bugs are missed during testing.

The next steps show how the ASD:Suite can be used to mathematically verify all execution scenarios in the Alarm example and then generate behavioural correct code.

3.1 VERIFYING ALL EXECUTION SCENARIOS

To begin verification, launch the ASD:Suite. Open the 'AlarmSystem.dm' design model from the folder where you extracted the Alarm Example zip file (e.g. …\My Documents\My ASD Models\Examples\AlarmSystem).

Start the verification of the design model by pressing F5. Since the model properties do not have a source code language and version set yet, you are asked to set these properties:

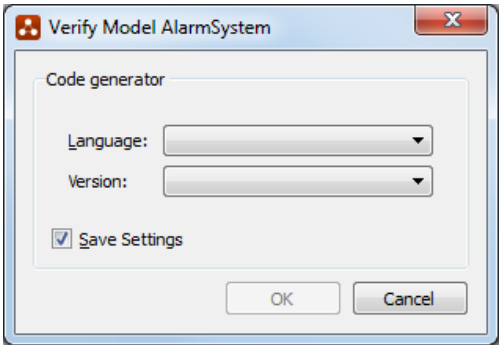


Figure 3-1 Language and Version selection dialog.

Click OK, then the Verification selection dialog is shown, containing the list of different types of verification checks to perform for each model:

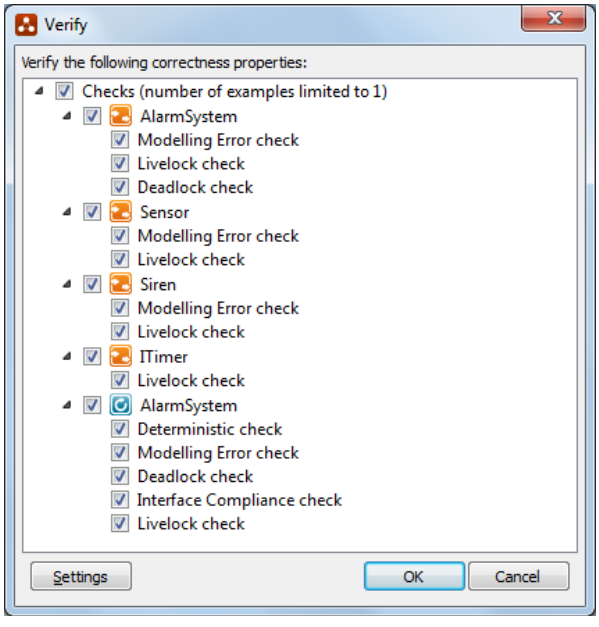


Figure 3-2 Verification selection dialog.

Copyright © 2013 Verum Software Technologies B.V.

Click OK. During verification, the design model is checked against all possible execution scenarios. When verification is finished, the Verification Results window is displayed:

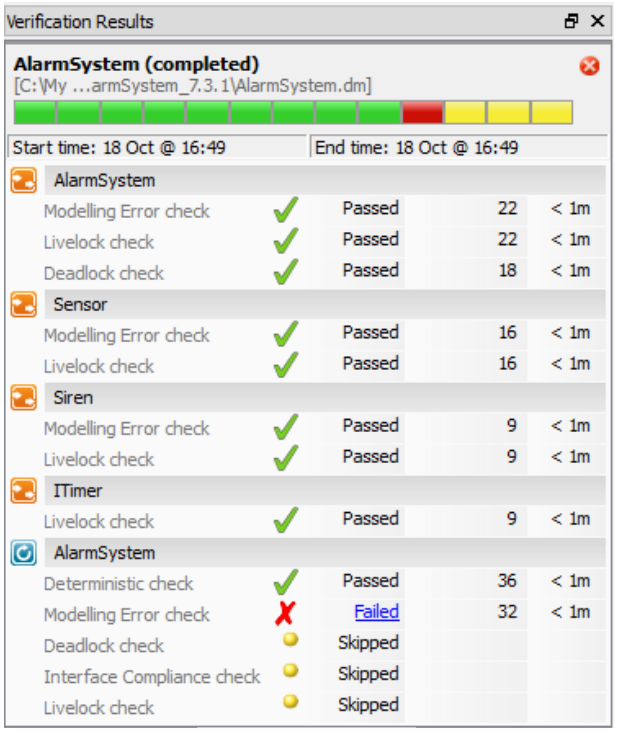


Figure 3-3 Verification results.

This window shows that the AlarmSystem design model has a defect. Clicking on the 'Failed' link brings up the Visual Verification window with a sequence diagram. This gives a more graphical presentation of the problem:

Copyright © 2013 Verum Software Technologies B.V.

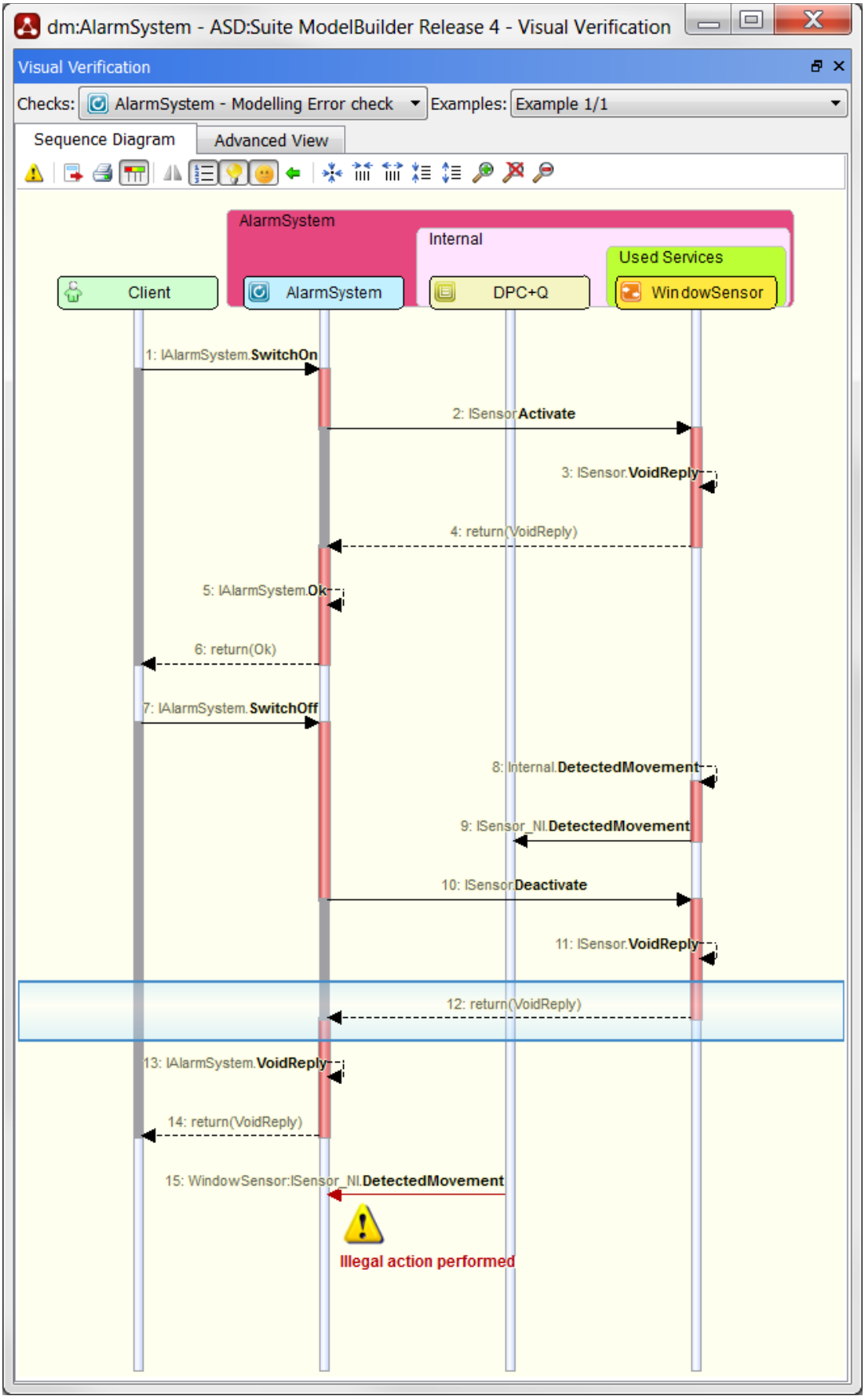
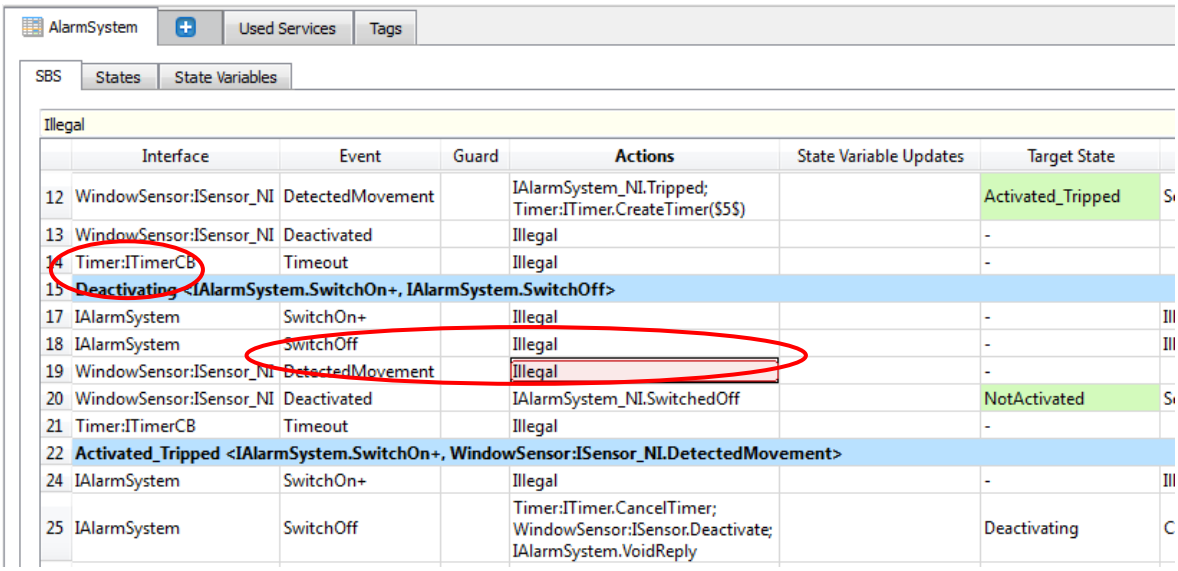


Figure 3-4 Sequence diagram leading to the reported problem.

Copyright © 2013 Verum Software Technologies B.V.

When you double-click on the yellow triangle, that indicates the location where the error occurred, then the ASD:Suite shows in the design model where the defect is located.

(tip: you can undock the various windows to display the sequence diagram and design model side by side)



	Interface	Event	Guard	Actions	State Variable Updates	Target State	
12	WindowSensor:ISensor_NI	DetectedMovement		IAlarmSystem_NI.Tripped; Timer:ITimer.CreateTimer(5)		Activated_Tripped	S
13	WindowSensor:ISensor_NI	Deactivated		Illegal		-	
14	Timer:ITimerCB	Timeout		Illegal		-	
15	**Deactivating <IAlarmSystem.SwitchOn+, IAlarmSystem.SwitchOff>**						
17	IAlarmSystem	SwitchOn+		Illegal		-	Il
18	IAlarmSystem	SwitchOff		Illegal		-	Il
19	WindowSensor:ISensor_NI	DetectedMovement		Illegal		-	
20	WindowSensor:ISensor_NI	Deactivated		IAlarmSystem_NI.SwitchedOff		NotActivated	S
21	Timer:ITimerCB	Timeout		Illegal		-	
22	**Activated_Tripped <IAlarmSystem.SwitchOn+, WindowSensor:ISensor_NI.DetectedMovement>**						
24	IAlarmSystem	SwitchOn+		Illegal		-	Il
25	IAlarmSystem	SwitchOff		Timer:ITimer.CancelTimer; WindowSensor:ISensor.Deactivate; IAlarmSystem.VoidReply		Deactivating	C

Figure 3-5 In state ‘Deactivating’, Rule case 19: ‘DetectedMovement’

From this point you can step forward and backwards through the design model (using F10 and Ctrl+F10) to follow the exact sequence of events that led to the error.

In this design model, deactivation of the sensor works asynchronously. As you can see, the design model reports that in state ‘Deactivating’, the occurrence of the event ‘DetectedMovement’ is declared to be illegal. So, although the designer of the component assumed that ‘DetectedMovement’ will not happen in this state, it did happen for this execution scenario. The designer has overlooked the potential race condition where *just before* the alarm system is switched off, a movement was detected by the sensor, and that notification event was only received by the alarm system component *after* the switch off event was received. The DetectedMovement event is already in the queue when the Alarm System receives the request to switch off the alarm. Eventually the event in the queue must be dealt with.

Note: This is the same state and event as reported by the assertion failure message seen in the Alarm System console.

3.2 CORRECTLY HANDLING THE RACE CONDITION.

Fixing the race condition in this example is simple. As this particular race condition cannot be prevented in this design, it should be allowed and properly dealt with. To do that, you should change the state where the Alarm System design is waiting for the completion of the deactivation of the sensor such that an arrival of the detected movement is simply ignored. This is done by changing the action ‘Illegal’ to ‘NoOp’, in rule case 19 in the ‘Actions’ cell. Note that ‘NoOp’ indicates that no action needs to be taken. The target state of rule case 19 has to be defined as

Copyright © 2013 Verum Software Technologies B.V.

'Deactivating' making the rule case a self-transition. The result of this is that when the DetectedMovement event comes out of the queue in this state, it will be processed without changing the state.

To make this change to the design model, select the Actions cell in rule case 19 and press F2 to edit it. In the Edit dialog, delete the Illegal entry, and then double-click on 'NoOp' in the list of possible actions:

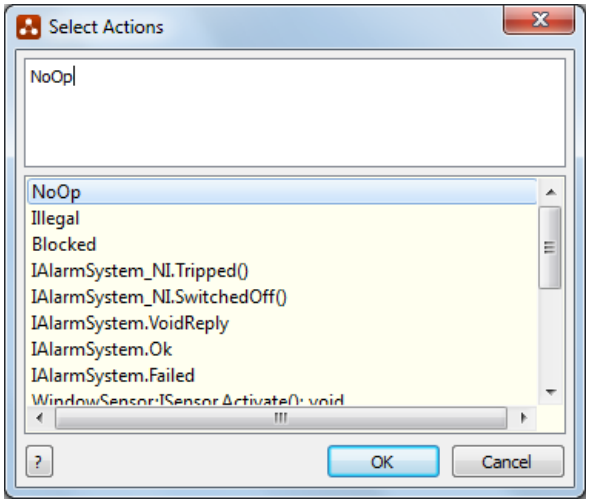


Figure 3-6 Action editor dialog.

Then click OK to apply the change. Do the same for the Target State cell: select it, press F2 to edit, click on 'Deactivating' to select that state and click on OK to apply the change:

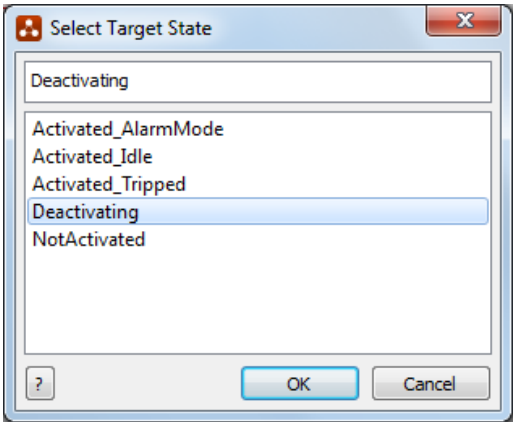


Figure 3-7 Target state editor dialog.

This is a very simple race condition case! In general, fixing a race condition can be much more complicated. It depends on the root cause of the race condition and what should happen for each of the events involved in it.

After changing the Alarm System design to fix the race condition, run verification again. Confirm that verification runs completely to the end and no more failures are reported in the Verification Results window.

Note: The Alarm System example includes a second design model 'AlarmSystem_corrected.dm'. This design model has already been fully verified with the fix described above.

Copyright © 2013 Verum Software Technologies B.V.

In case you have changed the ‘AlarmSystem.dm’ and want to revert to the original situation, you can copy the ‘AlarmSystem_original.dm’ over the ‘AlarmSystem.dm’, this file contains the initial situation with the design error.

3.3 GENERATING BEHAVIOURAL CORRECT CODE

Once verification reports no further defects for the design model, (re-)generate the Alarm System code. Select the Alarm System design model in the Model Explorer of the ASD:Suite and Choose ‘Code Generation’ > ‘Generate All Code’ from the menu. In the ‘Generate All Code’ dialog that appears, select your preferred language and version.

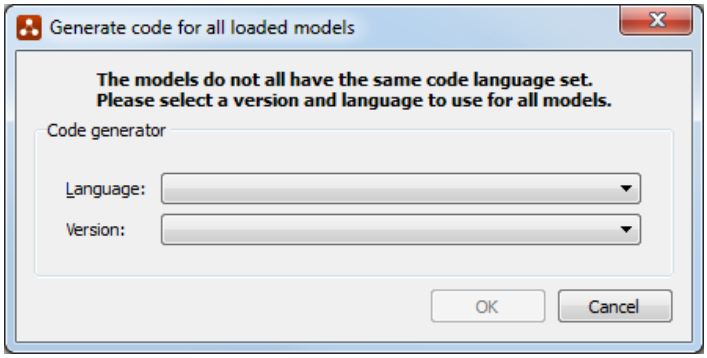


Figure 3-8 Select language and version for code generation.

Next, rebuild the application as described in the next section.

Copyright © 2013 Verum Software Technologies B.V.

4 BUILD AND RE-RUN THE ALARM SYSTEM

4.1 DOWNLOADING THE ASD:RUNTIME

The ASD:Runtime is a software package which enables ASD:Suite generated code to run on various software development platforms. The ASD:Runtime is provided as source code in a target programming language, therefore it has to be compiled by you in your software project, in the same way as you compile any other source module written (or generated) in the respective programming language.

The ASD:Runtime software package is available for download from the ASD Server. The ASD:Runtime is language specific, i.e. there is one package for each target programming language supported by the ASD:Suite. The number and type of files in the ASD:Runtime software package vary according to the selected language.

The used version of the ASD:Runtime must match the selected code generator version. A new ASD:Runtime is released with each code generator version. If the ASD:Runtime version does not match the version of the generated code, this will show up as a compilation error.

Important: The ASD:Runtime files (with the exception of the diagnostics handler) should not be modified/altered by you. If you do so the guarantees provided by ASD are invalidated.

Here is how you download the ASD:Runtime:

1. In the ASD:Suite, select 'Code Generation' > 'Download Runtime …'
2. In the Download Runtime dialog, select the desired language and version, and click the 'Browse…' button to select the location where to store the ASD:Runtime (in the local folder that you used to store this example):

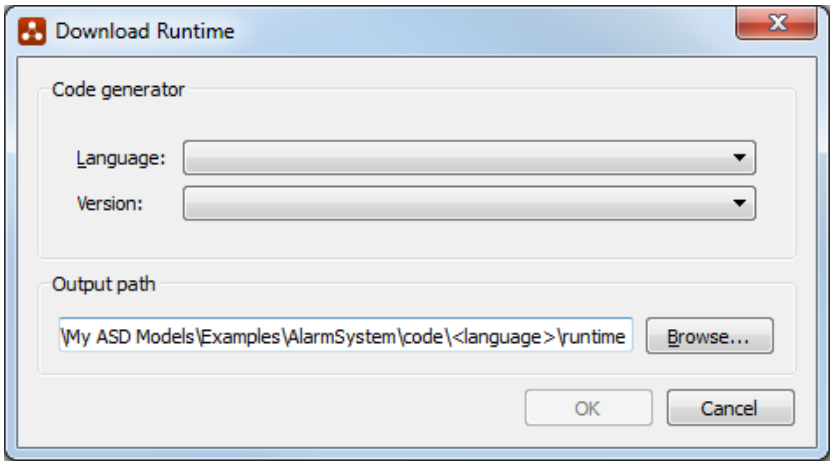


Figure 4-1 Download ASD:Runtime dialog.

   a. For C++ this location must be <local folder>\'code\cpp\runtime'
   b. For C# this location must be <local folder >\'code\cs\runtime'
   c. For Java this location must be <local folder >\'code\java\runtime'

   Note: For C there is no demo-system included in the downloaded Alarm Example.
3. Click OK, the ASD:Runtime is now downloaded into the selected folder.

Copyright © 2013 Verum Software Technologies B.V.

4.2 BUILDING AND RUNNING THE SOURCE CODE

Once all code is generated from the ASD models and the ASD:Runtime has been downloaded, the Alarm System can be built and run for C++, C#, or Java. The corresponding project files can be found for each language in the directories below:

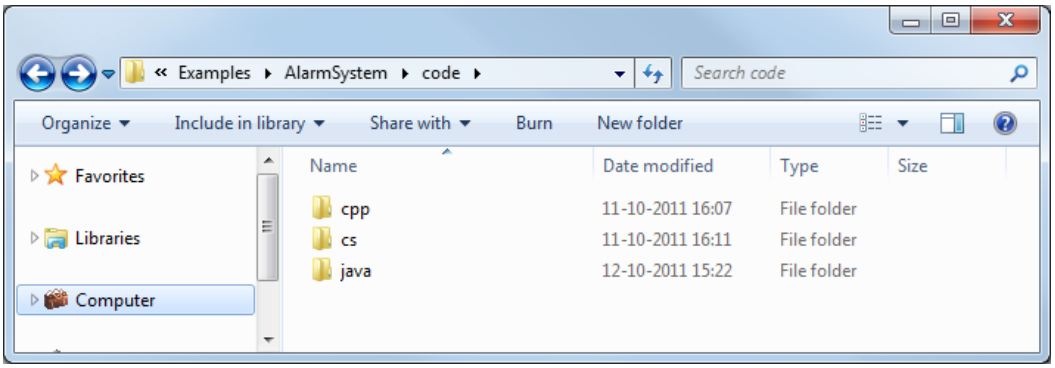


Figure 4-2 Top level of the code directory for the supported languages.

The next sections describe in detail how to open, build and run the projects in different development environments. When the built Alarm System is running, try the same sequence of actions as before, described in section 2.1 *Running the Alarm System Example* and section 2.2 *Alarm System Software Defect*. When you do so, you'll see that the assertion failure does not occur any more, by verification of all possible execution scenarios the design defect has been found and fixed, the Alarm System application is now behaving correct.

Note: If you change the ASD models you should verify and generate the code again as explained in Section 3 *Verification and Code Generation.*

4.2.1 C++

The 'cpp' code directory contains two Visual Studio C++ solution files. One is for Visual Studio 2008 and one is for Visual Studio 2010. Select the solution file (.sln) for the corresponding version of Visual Studio you want to use. The solution files will also work with the Visual Studio Express C++ edition 2008 or 2010. Visual Studio Express can be downloaded from the Microsoft website at http://www.microsoft.com/express/Downloads/.

Copyright © 2013 Verum Software Technologies B.V.

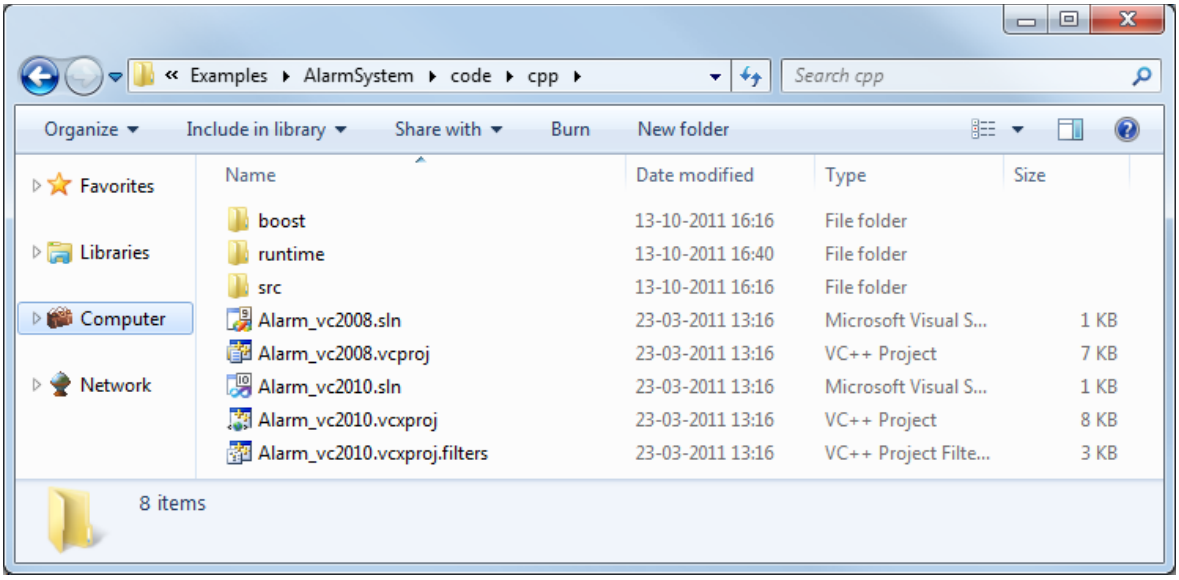


Figure 4-3 C++ code directory including Visual Studio solution files.

Double-click on the solution file to open it and launch Visual Studio. Press F5 to build and run the Alarm System application.

4.2.2 C#

The 'cs' code directory contains two Visual Studio C# solution files. One is for Visual Studio 2008 and one is for Visual Studio 2010. Select the solution file (.sln) for the corresponding version of Visual Studio you want to use. The solution files will also work with the Visual Studio Express C# edition 2008 or 2010. Visual Studio Express can be downloaded from the Microsoft website at http://www.microsoft.com/express/downloads/.

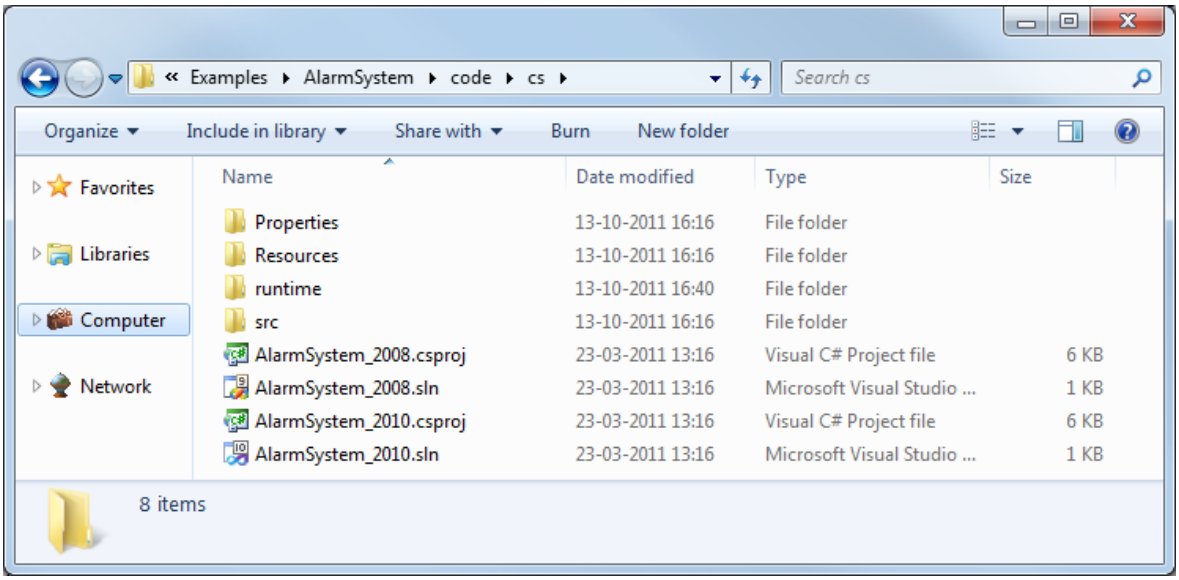


Figure 4-4 C# code directory including Visual Studio solution files.

Copyright © 2013 Verum Software Technologies B.V.

Double click on the solution file to open it and launch Visual Studio. Click on F5 to build and run the Alarm System application.

4.2.3 Java

The Eclipse development environment or the NetBeans IDE can be used for building and running the Alarm System in Java. The ASD:Suite supports JRE 1.6 and only needs Java SE (Standard Edition).

4.2.3.1 Eclipse

Start Eclipse and import the Alarm System project: from the menu, select 'File->Import...'. In the dialog that appears, select 'Existing Projects into Workspace' under the 'General' template:

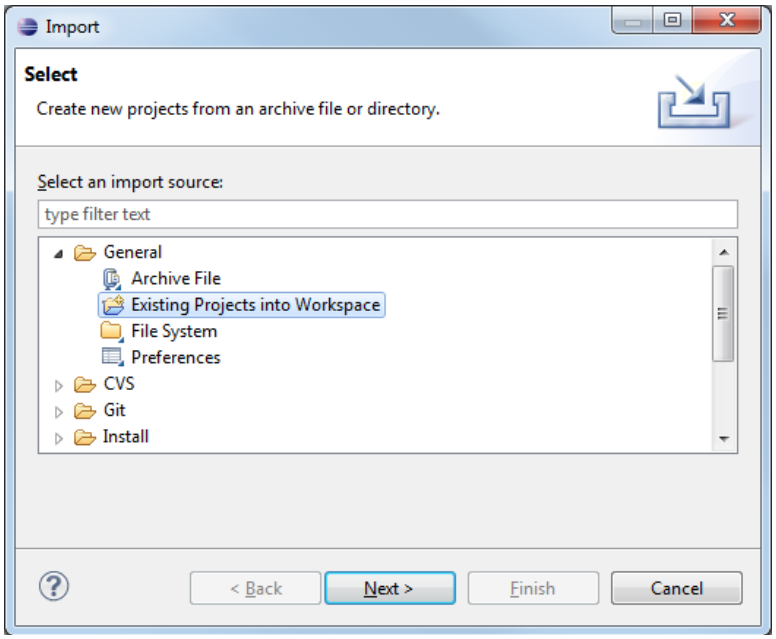


Figure 4-5 Eclipse import: 'Existing Projects into Workspace'.

Click on the 'Next' button. In the following dialog use the browse button to find the topmost directory of the Alarm System project: '...\AlarmSystem\code\java':

Copyright © 2013 Verum Software Technologies B.V.

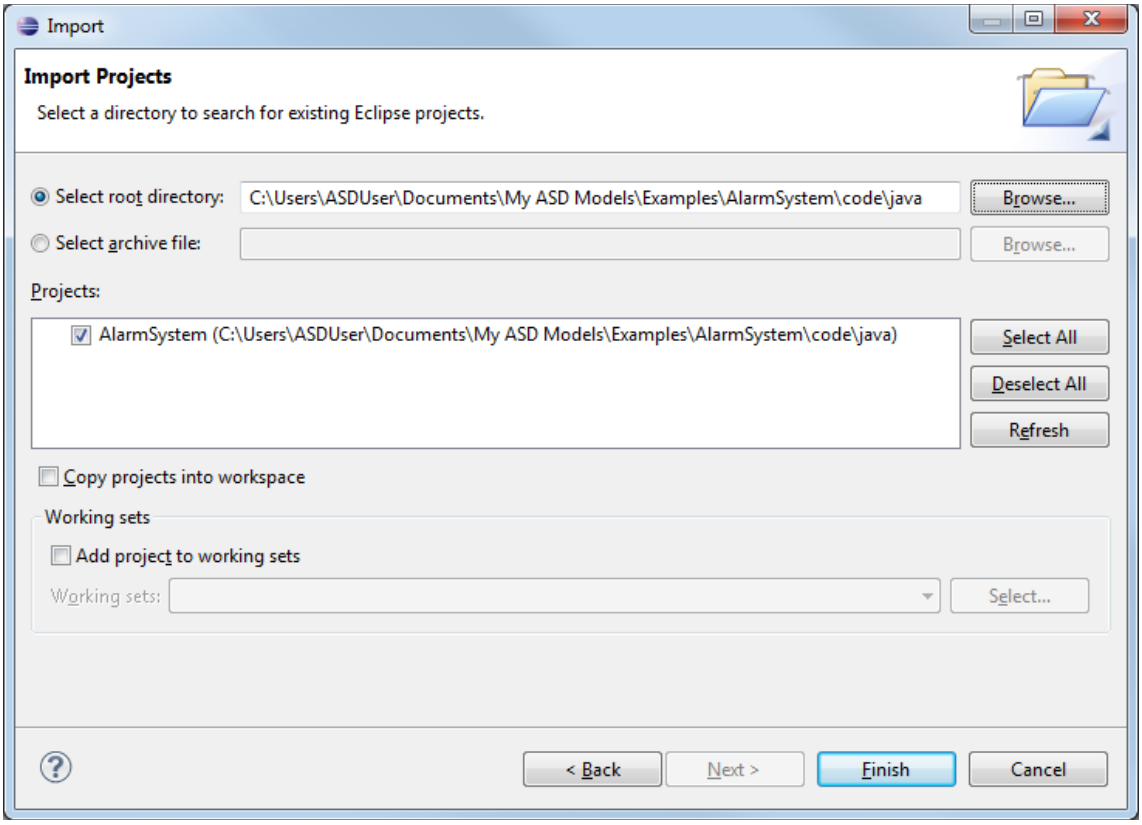


Figure 4-6 Import Projects dialog.

Click on 'Finish' to add the Alarm System project to your Eclipse workspace.

Note: Do **not** select the option 'Copy projects into workspace'. This creates a copy of the Java source files from the Alarm System project. If you subsequently make changes to the Alarm System ASD:Suite models and regenerate the Alarm System code, the source files at the original location get updated but not those of any project that has been copied.

Select the Alarm System project in the Eclipse Project Explorer and press Ctrl-F11 to build and run the Alarm System application. If you get a dialog asking how to run the application, then select run as 'Java Application'.

4.2.3.2 NetBeans IDE

Start the NetBeans IDE. From the menu, select 'File'. In the file dialog, select 'Import Project', and then select 'Eclipse Project...'.

In the next dialog, select the second option 'Import Project ignoring Project dependencies'.

- For 'Project to Import', use the Browse button to select the topmost folder of the Alarm System project: '…\AlarmSystem\code\java'.
- For 'Destination Folder', select the directory where you store your NetBeans projects:

Copyright © 2013 Verum Software Technologies B.V.

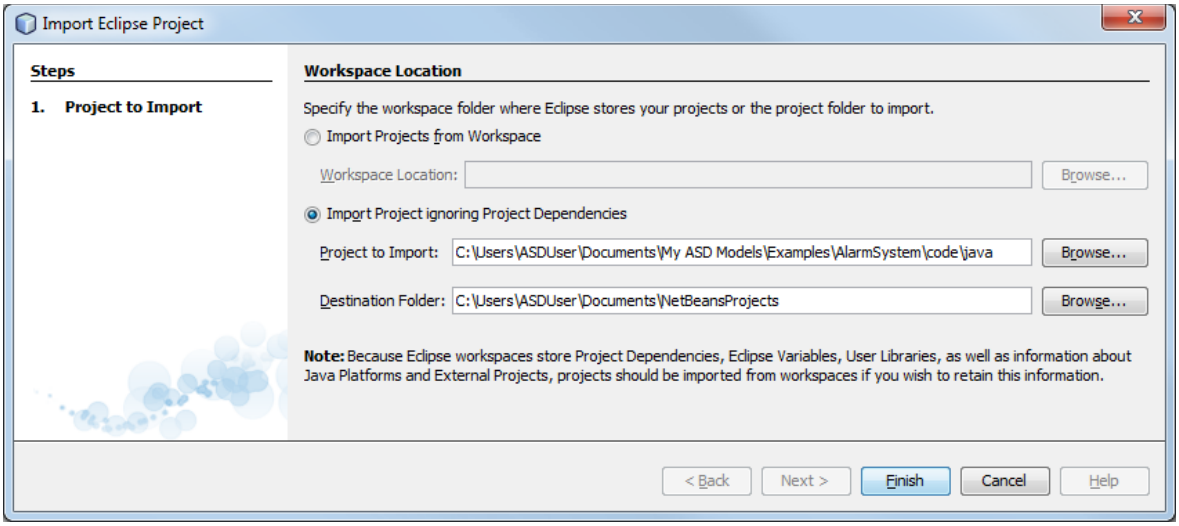


Figure 4-7 Import Eclipse project: Import Project ignoring Project Dependencies.

Note: You can ignore the warning in the dialog note, since this Eclipse project does not have any additional stored project information.

Click on 'Finish' to create the NetBeans project. Select the AlarmSystem project in the 'Projects' window. Press F6 to build and run the application. When requested to select the main class, choose 'AlarmSystem.Console':

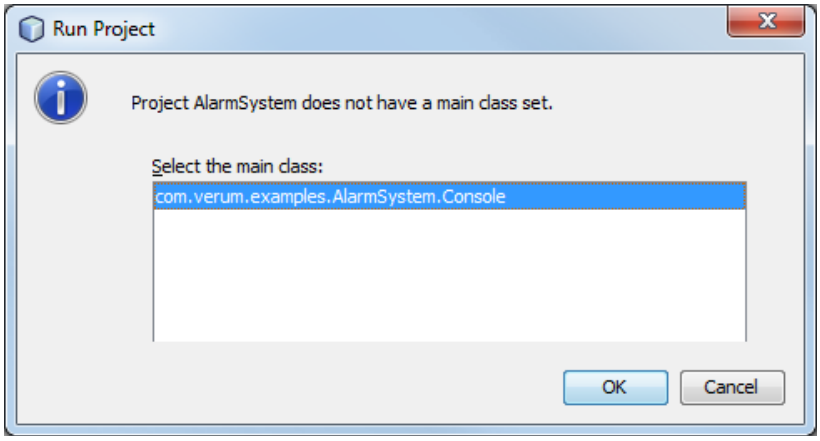


Figure 4-8 Setting the main class.

Copyright © 2013 Verum Software Technologies B.V.