CHESS Architectural Patterns- Quick Guide

May 2020

1 Table of Contents

2 List of Figures

3 Document history

Date	Changes
May 2020	First version, from user manual

4 Introduction

An architectural pattern is a general, reusable solution to a commonly occurring problem in system architecture within a given context.

The CHESS tool includes a library of architectural patterns to be instantiated in a model. It is also possible to create new patterns and so different libraries.

A pattern represents “virtual” components instances connected and collaborating in a given way; the goal of the pattern application process is to identify the actual components instances that will play the roles of the virtual component instances defined by the pattern.

4.1 Instantiate an architectural pattern into the system architecture

Predefined patterns can be instantiated into a system architecture. Such patterns may defined in the CHESS library or defined in a given Papyrus project available in the current workspace. A pattern is applicable in the context of a composite Block, where owned component instances are available, or can be created.

> *Note: in order to use the patterns instantiation feature, the CHESS constraint “Cannot apply further profiles in the model” has to be disabled; for the current AMASS platform, this can be done by using the CHESS preferences page available from the main Eclipse menu “**Window → Preferences**”.*

In concrete, an architectural pattern can be instantiated in a given CHESS model by applying the following steps:

1. Open the CHESS model where the pattern must be applied.
2. Open the ModelExplorer view, right click the CHESS root model entity and select:
   2.1. Import Registered Package to load the CHESS library, or
   2.2. Import Package from User Model to load the available Papyrus project
3. In the ModelExplorer select a system component where the component instances playing the pattern roles will be identified/created; right click and select “**CHESS → DesignPatterns → Select and Apply a Design Pattern**”. A dedicated wizard is shown (see Figure 1 and Figure 2).
4. Select the pattern to instantiate from the Available Patterns lists (this list is initialized with the patterns library/projects that have been imported in step 2) and click “Apply”.
5. Bind the information available in the pattern into the current system model.

   Figure 1 shows the role binding dialog available for the Triple Modular Redundancy pattern. The upper part shows the roles defined by the pattern, while the lower part shows the available candidates for binding in the system under design. Candidate matching simply relies on the meta-model kind, i.e. components match components, components parts to components parts, ports to ports, connections to connections; dedicated binding dialogs are available for each kind of meta-model entity by using the "Next" button of the main dialog window. In order to declare a binding, select a pair in the upper and the lower part of the dialog, respectively and then click on the "Create mapping" button.

Note: A design pattern describes a set of roles that elements play in a pattern. In many cases, elements that play a certain role do already exist in the application model. Therefore, it is important to identify these and declare a binding to a role in the pattern (i.e. a role binding). This information is used to determine which elements of a pattern need to be copied/created into the application and which don't.

6. Click “Finish”; patterns entities (types, ports, instances, connectors) which have not been manually mapped to system entities are automatically created in the model, and dependencies between system model entities and pattern ones are created and stored in the CHESS system model under a dedicated PatternApplication entity (see Figure 3). Component instances playing a part in the pattern are tagged as <<PatternRole>> which stores the information of the applied pattern.

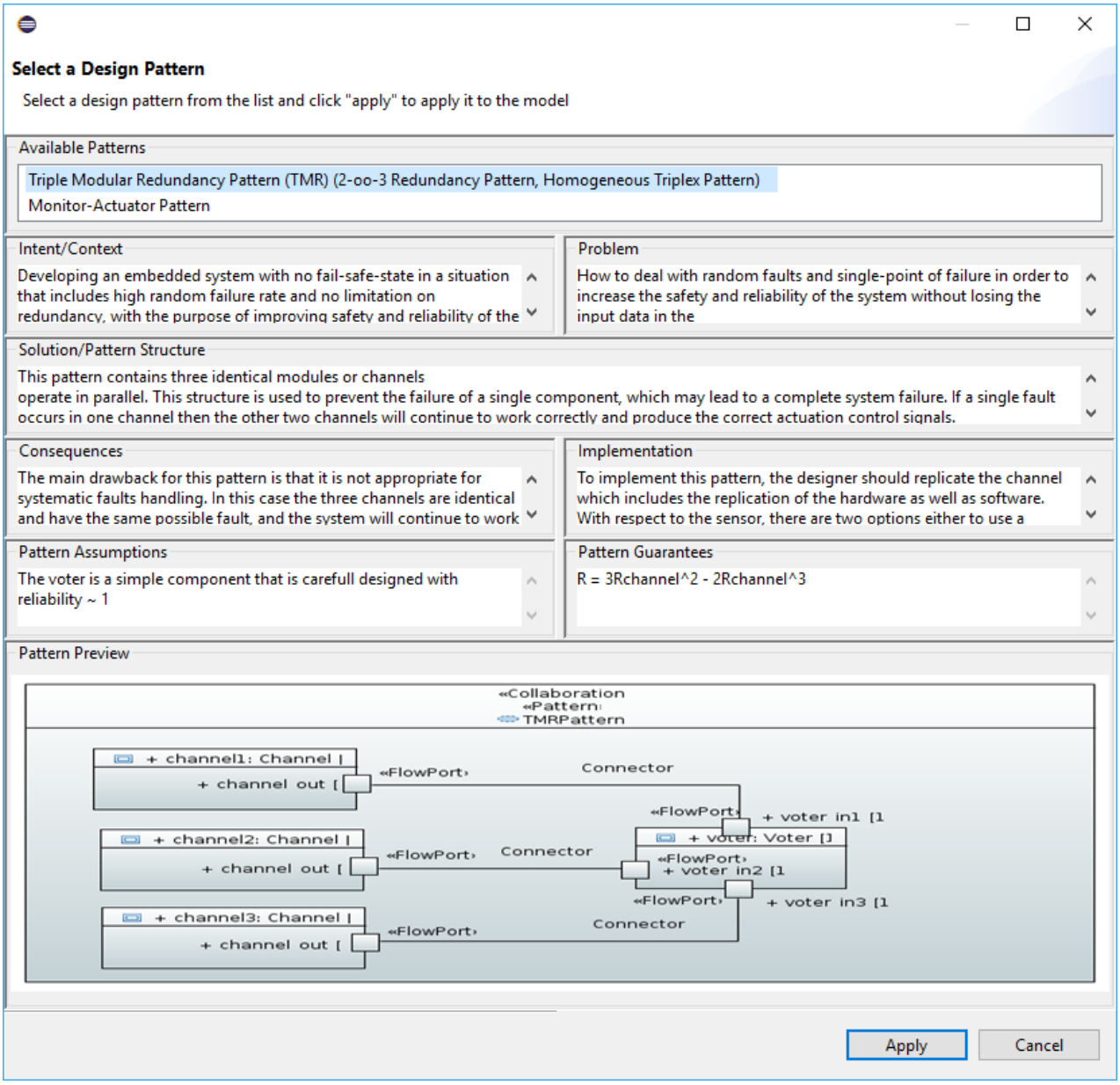

Figure 1. Design Pattern selection

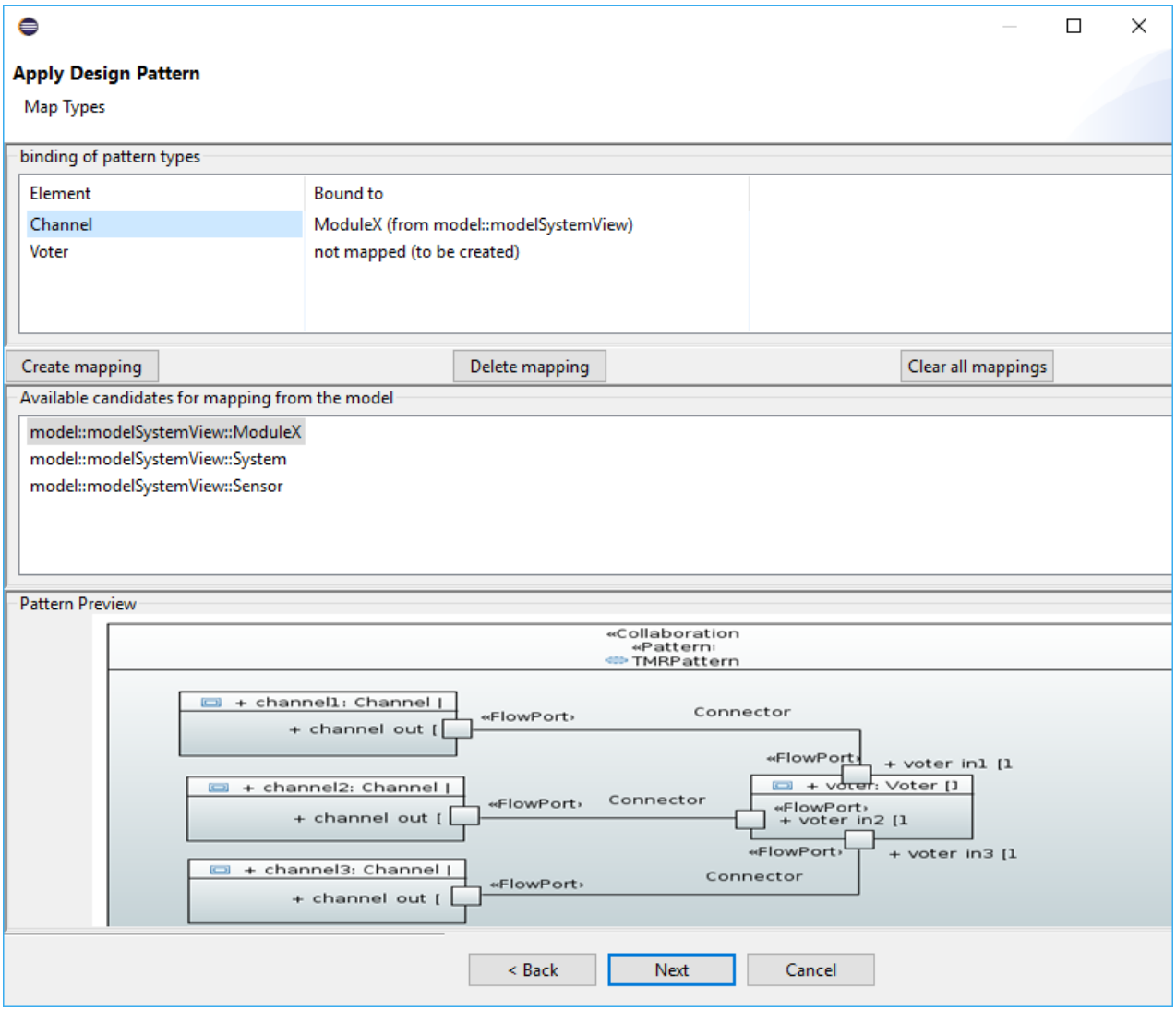

Figure 2. Design Pattern instantiation wizard

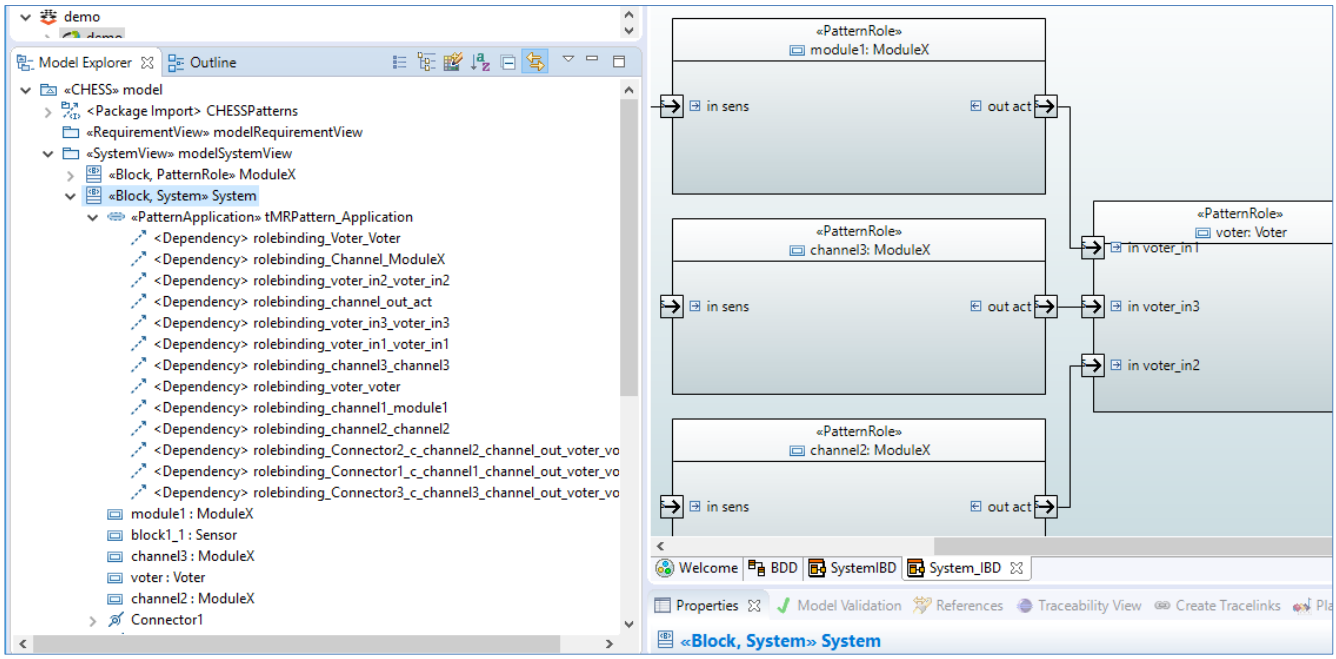

Figure 3. Pattern Application

4.2 Define a new architectural pattern

Patterns are stored in Papyrus projects; it is recommended to use dedicated Papyrus projects for patterns definition, so to keep system components and patterns models separated and be able to share the latters without the need to share the formers. It is possible to use the same Papyrus project to define and store different patterns or split the patterns in different Papyrus projects.

To create a new pattern, create or open a Papyrus model; in case of a new model, apply the registered profile “CHESS Design Pattern” to the model itself (select the CHESS model in the Model Explorer Papyrus view, open the “Profile” tab in the Properties view and select the “Apply Registered Profile” command).

The structure of a pattern is shown in the Figure 4 taken from the Papyrus Model Explorer View; the Triple Modular Redundancy pattern coming with the CHESS library is taken as example (see Figure 5).

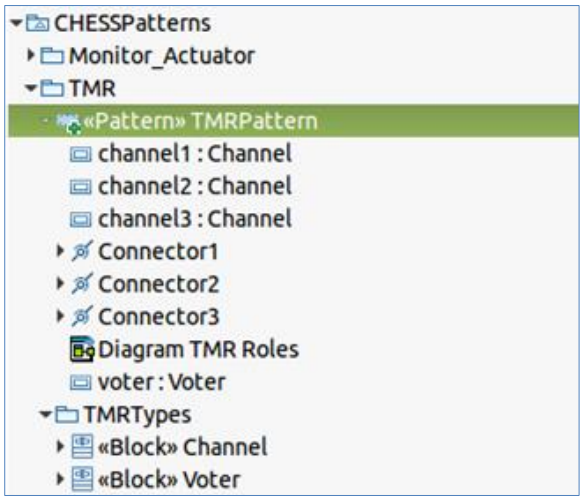

Figure 4. Pattern structure

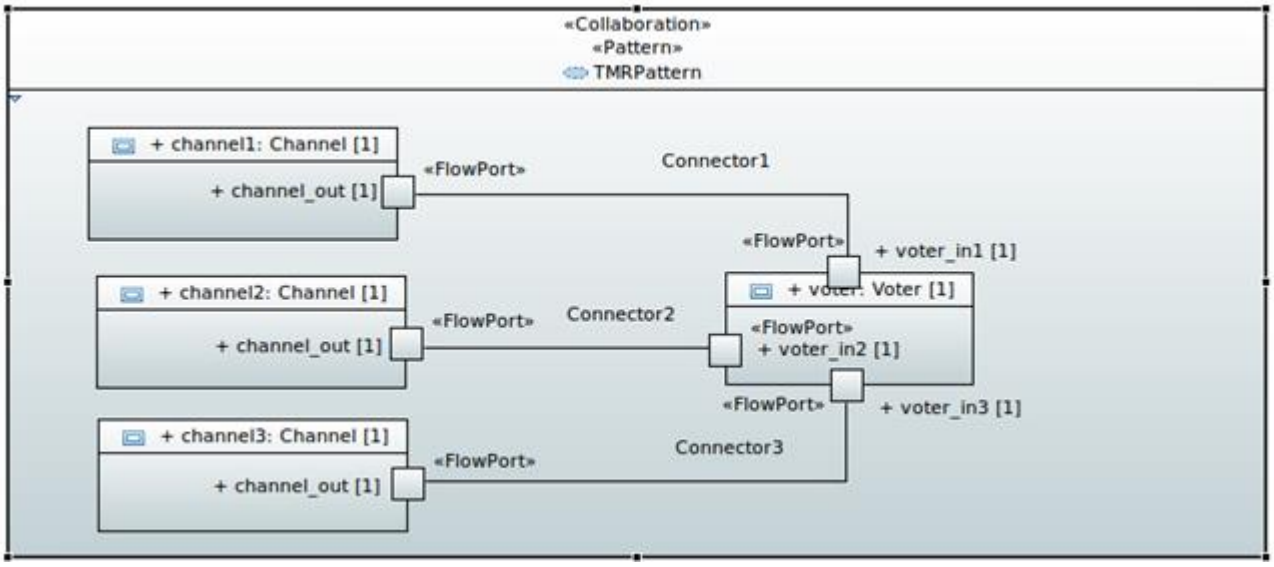

Figure 5. Pattern Structure, Composite Structure Diagram

The pattern itself is UML Collaboration stereotyped as Pattern (from the *PatternsProfile* profile coming with CHESS). A role is a UML Property defined as CollaborationRole.

To create a new pattern:

1. (optional but recommended) Create a new UML Package (from the ModelExplorer) to contain all the elements of the new profile.
2. (optional) Create a Class Diagram (from the ModelExplorer).
3. Create (from the ModelExplorer or from the Class Diagram Palette) UML Class elements to be used as type for the roles of the pattern (see Channel and Voter Blocks for the TMR example)
4. Create a UML Collaboration (from the ModelExplorer) and stereotype it as <<Pattern>>.
5. Set the values of the stereotype properties (from the “Profile” tab of the Eclipse Properties View). Indeed, Pattern stereotype comes with a list of attributes through which it is possible to declare the functional and extra functional (safety, security, performance) properties that can defined for the current pattern and reused across pattern instantiation; for instance, when the pattern is instantiated in a given system architecture, it is possible to reuse the aforementioned properties to support specific claim about system properties.
6. Create (from the ModelExplorer) a Composite Structure Diagram (CSD) for the UML Collaboration.
7. Create Property elements inside the CSD (from the CSD Palette/Model Explorer) and set their type to the appropriate UML Class created at 2.
8. Set each Property element as CollaborationRole (from the CSD Palette).
9. Add Ports and Connectors to the CSD elements (from the CSD Palette/Model Explorer).