D-Case Editor: A Typed Assurance Case Editor

Yutaka Matsuno
The University of Tokyo, Japan
JST, CREST
matsu@cc.u-tokyo.ac.jp

Abstract

System assurance has become an important issue in many system domains, especially in safety-critical domain. Recently, assurance cases [3] have been getting much attentions for the purpose. We demonstrate D-Case Editor [10], which is an assurance cases editor being developed in DEOS (Dependable Embedded Operating System for Practical Uses) project funded by Japan Science and Technology Agency. D-Case Editor has been implemented as an Eclipse plug-in using Eclipse GMF framework. Its characteristics are (1) supporting GSN (Goal Structuring Notation) [8], (2) GSN pattern library function and prototype type checking function [9], and (3) consistency checking function by an advanced proof assistant tool [13]. To achieve these characteristics, we have exploited types in several ways. In this paper, we briefly introduce assurance cases, and demonstrate the functions of D-Case Editor. Because it has been implemented on Eclipse, it is interesting to make a tool chain with existing development tools of Eclipse. D-Case Editor is available as an open source in the following web page: http://www.il.is.s.u-tokyo.ac.jp/deos/dcase/.

1 Introduction

System assurance has become a great importance in many industrial sectors. Safety cases (assurance cases for safety of systems) are required to submit to certification bodies for developing and operating safety critical systems, e.g., automotive, railway, defense, nuclear plants and sea oils. There are several standards, e.g. EUROCONTROL [5], Rail Yellow Book [12] and MoD Defence Standard 00-56, which mandate the use of safety cases.

There are several definition for assurance cases [3]. We show one of such definitions as follows[1].

"a documented body of evidence that provides a convincing and valid argument that a system is adequately dependable for a given application in a given environment."

Assurance cases are often written in structured documents using a graphical notation to ease the difficulty of writing and certifying them. Goal Structuring Notation (GSN) is one of such notations [8]. Writing assurance cases and reusing them in a cost effective way is a critical issue for organisations. Patterns and their supporting constructs are proposed in GSN for the reuse of existing assurance cases, which includes parameterized expressions.

Assurance cases have been recognized as a key method for dependability of systems. However, currently there have been not so much tools for assurance cases (very few in open source.) A notable tool is ASCE tools [1], which has been widely used in several areas such as defense, safety critical area, and medical devices.

To make assurance case more familiar to developers who are using open sources tools, we have released D-Case Editor, an open source assurance case editor implemented on Eclipse GMF. The web page is http://www.il.is.s.u-tokyo.ac.jp/deos/dcase/. The characteristics are as follows.

1. Supporting GSN (Goal Structuring Notation) [8],
2. GSN pattern library function and prototype type checking function [9], and
3. Consistency checking function by an advanced
proof assistant tool [13].

To achieve these characteristics, we have exploited types in several ways. For example, we introduce types for variables used in GSN patterns [4]. Our intention is to make assurance cases to be shared among various tools for wider use. Introducing types is one attempt for the purpose.

The structure of this paper is as follows. In Section 2, we introduce assurance cases and patterns, and some standardization efforts for assurance cases. Section 3 introduces several functions of D-Case Editor. Section 4 states a few concluding remarks.

2 Background Knowledge

2.1 Goal Structuring Notation (GSN)

Goal Structuring Notation (GSN) is introduced by Tim Kelly and his colleagues at University of York [8]. It is a graphical notation for assurance cases. GSN is widely used for safety cases. Some safety cases written in GSN are publicly available [2]. We briefly explain constructs and their meanings in GSN. Arguments in GSN are structured as trees with a few kinds of nodes, including: goal nodes for claims to be argued for, strategy nodes for reasoning steps that decompose a goal into sub goals, and evidence nodes for references to direct evidences that respective goals hold. Figure 1 is a simple example of GSN. The root of the tree must be a goal node, called top goal, which is the claim to be argued (G1 in Figure 1.) For G1, a context node C1 is attached to complement G1. Context nodes are used to describe the context (environment) of the goal attached to. A goal node is decomposed through a strategy node (S1) into sub goal nodes (G2 and G3). The strategy node contains an explanation, or reason, for why the goal is achieved when the sub goals are achieved. S1 explains the way of arguing (argue over each possible fault: A and B). When successive decompositions reach a sub goal (G2) that has a direct evidence of success, an evidence node (E1) referring to the evidence is added. Here we use a result of fault tree analysis (FTA) as the evidence. For the sub goal (G3) that is not decomposed nor supported by evidences, a node (a diamond) of type undeveloped is attached to highlight the incomplete status of the case. The assurance case in Figure 1 is written with D-Case Editor.

2.2 GSN Patterns

Writing and certifying assurance cases are difficult because they tend to be huge and complex, and they require domain specific knowledge of target systems. To ease the difficulties, it has been recognized that assurance case patterns should be collected and available for reuse, similarly to design patterns in object oriented languages. There have been several publicly available GSN patterns ([7, 14, 4]).

Figure 2 is a simple example of GSN patterns in [4]. The top-level goal of system safety (G1) is re-expressed as a number of goals of functional safety (G2) as part of the strategy identified by S1. In order to support this strategy, it is necessary to have identified all system functions affecting overall safety (C1) e.g. through Functional Hazard Analysis (FHA). In addition, it is also necessary to put forward (and develop) the claim that either all the identified functions are independent, and therefore have no interactions that could give rise to hazards (G4) or that any interactions that have been identified are non-hazardous (G3).

Figure 2 includes main GSN extensions for GSN patterns, as defined in [6]:

- Parameterized expressions. {System X} and {Function Y} are parametarised expressions. We can instantiate X and Y by appropriate (possibly safety critical) system and function, respectively.
- Uninstantiated. Triangles (△) attached to nodes indicate that the nodes contain uninstantiated parameterised expressions. To instantiate the GSN pattern as an assurance case, we need to instantiate the expressions.
- 1 to many expressions (multiplicity). Number of functions are different by the target system. We can instantiate the number of functions (n) for the target system.
- Choice. By this extension, we can choose appropriate goals for the target system.

2.3 Assurance Cases and Standardization Efforts

Two major graphical notations for assurance cases are GSN and CAE (Claims, Arguments, and Evidence) [1]. There are two standardization efforts for assurance cases; the system assurance task force at the OMG (Object Management Group) and GSN standardization effort [6]. OMG has standardized
Goal: G_1

C/S Logic is free from possible faults

Strategy: S_1

Argue over each possible fault

Goal: G_2

C/S Logic is free from fault A

Evidence: E_1

FTA analysis Result

Undeveloped: U_1

No evidence is currently given

Goal: G_3

C/S Logic is free from fault B

Risk Analysis Result: Possible faults are fault A and fault B

Figure 1: A simple GSN Example

Q1: {System X} is Safe

S1: Argument over all safety-related functions of system

Q2: {Function Y} is safe

Q3: Interactions between system functions are non-hazardous

Q4: All system functions are independent (no interactions)

Indicates a 1-to-many relationship

Indicates that element remains to be instantiated and then developed

Indicates that element remains to be instantiated

Indicates a choice

Indicates that element remains to be developed (supported)

Figure 2: An example of GSN patterns [4]
the meta-model for assurance cases called ARM (Argument Metamodel) [11] by which both notations are in fact interchangeable. The main aim of the ARM is to align two major notations and facilitates the tool support. Unfortunately it only reflects main constructs between the two, and some specific features, which are not compatible are missing from it. For instance, patterns are not included in the ARM.

3 Overview of D-Case Editor

Figure 3 shows a screen shot of D-Case Editor. Users can draw GSN diagrams in the canvas. In the right, there is a pattern library. From the library, users can choose already existing, good assurance case patterns and fragments, and copy to the canvas. Current D-Case Editor has the following functions (some functions are omitted in current version.) Consistency checking with an advanced proof assistant tool [13] will be available soon.

- Checks on the graph structure of D-Case (e.g. no-cycle, no-evidence directly below a strategy, etc.)
- External info via url can be attached to a goal.
- “Patterns” with typed parameters can be registered and recalled with parameter instantiations.
- Graphical diff to compare two D-Cases.
- A “ticket” in Redmine, a project management web application, can be attached to a goal; the ticket’s status can be reflected graphically in D-Case (color change.)
- Monitoring: a url to be polled by Editor can be attached to a node; the answer is dynamically reflected in D-Case (color change.)
- Scoring: calculates a weighted score for a D-Case indicating how much of it is completed.
- connection with uml2tools: generating a D-Case subtree for a component diagram data.

Among these functions, we show how patterns with typed parameters can be registered and recalled with parameter instantiations. Current implementation is limited that variables and types can only be declared in the top level of the GSN term. Declaration of variables and types are written in an XML file, as shown in Figure 4. In Figure 4, variables STATUS, CPU, USAGE, and MESSAGE are declared, and given types enum, int, double, and string, respectively. Furthermore, these types are given useful restrictions such that the value of CPU (this variable is intended as the CPU resource usage rate of the target system) is restricted within 0 – 100%. Users of D-Case Editor can assign values to these variables via the parameter setting window. If a user mis-assigned a value (e.g., 150 for CPU), then D-Case Editor reports the type error. As far as we know, there is not any assurance case editor which has such parameterized expressions and type checking mechanism. We plan to implement the type checking mechanism in Section 3.

4 Concluding Remarks

We have presented our assurance case editor, called D-Case Editor. It has been implemented as an Eclipse plug-in using Eclipse GMF, and released as an open source. We hope that D-Case Editor would contribute to make assurance cases more familiar to developers by making a tool chain of D-Case Editor with Eclipse and other development tools. We plan to comply to OMG ARM [11] and other international standards related to assurance cases in next release.

References

<xml version="1.0" encoding="UTF-8"?>
<!-- all element names and attributes are case sensitive -->
<dataType>
  <parameter name="STATUS" type="enum">
    <items>
      <item value="NORMAL"/>
      <item value="ERROR"/>
      <item value="RUNNING"/>
      <item value="SATISFIED"/>
    </items>
  </parameter>
  <parameter name="CPU" type="int">
    <range min="0" max="100"/>
  </parameter>
  <parameter name="USAGE" type="double">
    <range min="0.00" max="999.99" digit="2" inc="0.02"/>
  </parameter>
  <parameter name="MESSAGE" type="string">
    <length min="0" max="20"/>
  </parameter>
</dataType>

Figure 3: A Screen Shot of D-Case Editor

Figure 4: Variables and Type Declarations XML file for D-Case Editor


