NASA/CR-2011-217149

Error Propagation Analysis in the SAE Architecture Analysis and Design Language (AADL) and the EDICT Tool Framework

Brian W. LaValley, Phillip D. Little, and Chris J. Walter
WW Technology Group, Ellicott City, Maryland

May 2011

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Report Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include creating custom thesauri, building customized databases, and organizing and publishing research results.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at *http://www.sti.nasa.gov*

- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA STI Help Desk at 443-757-5803

- Phone the NASA STI Help Desk at 443-757-5802

- Write to:
  NASA STI Help Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320

NASA/CR-2011-217149

Error Propagation Analysis in the SAE Architecture Analysis and Design Language (AADL) and the EDICT Tool Framework

Brian W. LaValley, Phillip D. Little, and Chris J. Walter
WW Technology Group, Ellicott City, Maryland

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NNL10AB32T

May 2011

Available from:

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320
443-757-5802

Table of Contents

1. Error Modeling and Analysis with AADL and EDICT

This report documents the capabilities of the EDICT tools for error modeling and error propagation analysis when operating with models defined in the Architecture Analysis & Design Language (AADL). We also discuss our experience using the EDICT error analysis capabilities on a model of the Scalable Processor-Independent Design for Enhanced Reliability (SPIDER) architecture that used the Reliable Optical Bus (ROBUS). Based on these experiences we draw some initial conclusions about model based design techniques for error modeling and analysis of highly reliable computing architectures.

1.1. Architecture Framework

Conducting dependability analysis requires the ability to evaluate various design alternatives to determine which system design alternative best meets all dependability requirements. There needs to be a framework in place that defines the levels of abstraction and how the levels relate to one another in order to construct a set of design alternatives that can be evaluated against a common set of design criteria. The EDICT tools contain an architecture framework that establishes the levels of abstraction and the relations between the levels. The EDICT architecture framework defines five levels of abstraction:

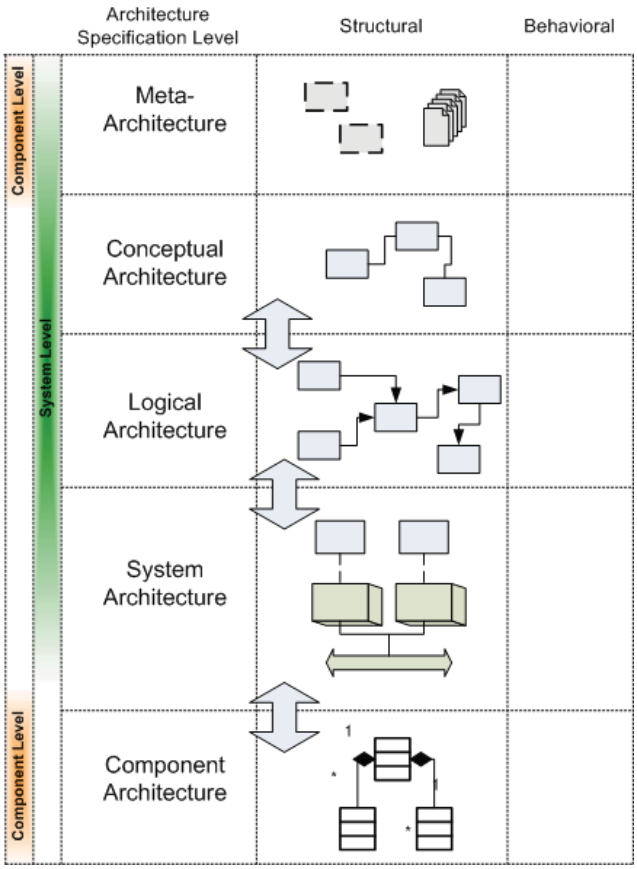

Figure 1: EDICT Architecture

- **Meta-Architecture**
  Identifies the governing architectural style, principles, templates and other reusable constructs.

- **Conceptual Architecture**
  Identifies appropriate decomposition and serves as a useful vehicle for non-technical communication of structure.

- **Logical Architecture**

  Identifies detailed blueprint of functional areas to be developed and a detailed specification of interfaces and interactions.

- **System Architecture Framework**
  Identifies how functional areas will be realized in software (processes, threads, etc.) and the hardware platform (processors, networks, etc.) that will host the software.

- **Component Architecture**
  Identifies the detailed structural design of specific software components

The levels of abstraction that are most relevant to this project are the Logical and System architecture levels, as these levels are where the dependability analysis in the EDICT Error Handling content occurs. The error modeling and analysis described in this report resides at the System Architecture layer.

The EDICT tools support an implementation of this framework and a supporting set of management tools to enable users to establish related sets of architecture models across the abstraction, to maintain inter-model references, and to define design options within those model sets. EDICT uses a pair of concepts to organize the models in an architecture framework and define the set of models that represent a design option.

The Design Effort establishes the scope/visibility of models, analyzers and generated artifacts that are contributing to the development or maintenance of a particular system. A Design Effort is composed of one or more *design options* and may contain models and analyzers that can contribute to *multiple* design options. Figure 2: EDICT Design Effort and Options Tree graphically depicts a Design Effort as a set of architecture models within the architecture framework. The models are represented by circles and the abstraction level relations indicate how the lower level models were derived from models higher in the framework.

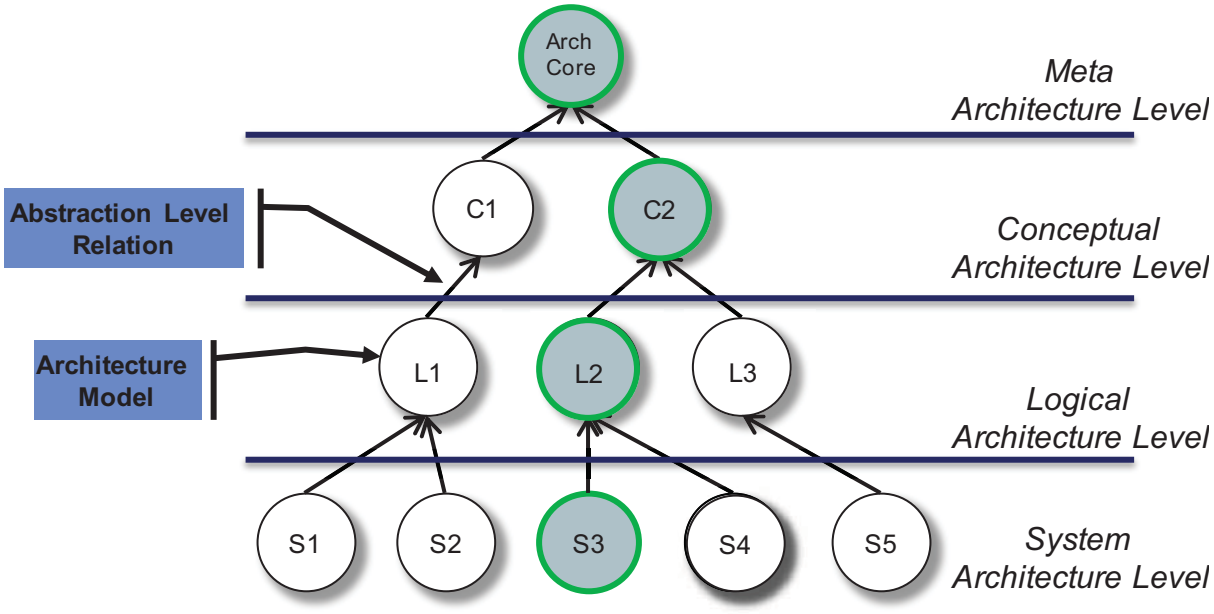

Figure 2: EDICT Design Effort and Options Tree

A Design Option defines *one* design that strives to meet the requirements of the system that is being developed. The design option is composed of a set of models, analyzers and artifacts that are related within the Design Effort. In Figure 2 one Design Option is highlighted in green. Design Options may be

used in a number of different ways; such as modeling and comparing competing designs or modeling and comparing design revisions.

1.2. Importing AADL Models

An AADL model may be a provider of system architecture information for EDICT. An AADL-specific adapter translates component and interface information from AADL to EDICT's internal representation. The resulting system architecture model is managed by EDICT's Architecture Model Services (AMS), which make the model available to the error handling content of System Composer and System Analyzer, as well as other tool clients. Architecture model generation configurations tell the AMS which EDICT architecture model is generated from which adapted model. The AMS provides the extension point for model change detection and notifications and management of the architecture model generation configurations. Figure 3 conceptually illustrates the relationship between an AADL model, the AMS, and EDICT tool clients.

In this project we have developed models of a 3x3 SPIDER architecture to use as an example in evaluating the error modeling and analysis capabilities. The Architecture Model Services are used to import the SPIDER model into EDICT for additional error modeling and propagation analysis.

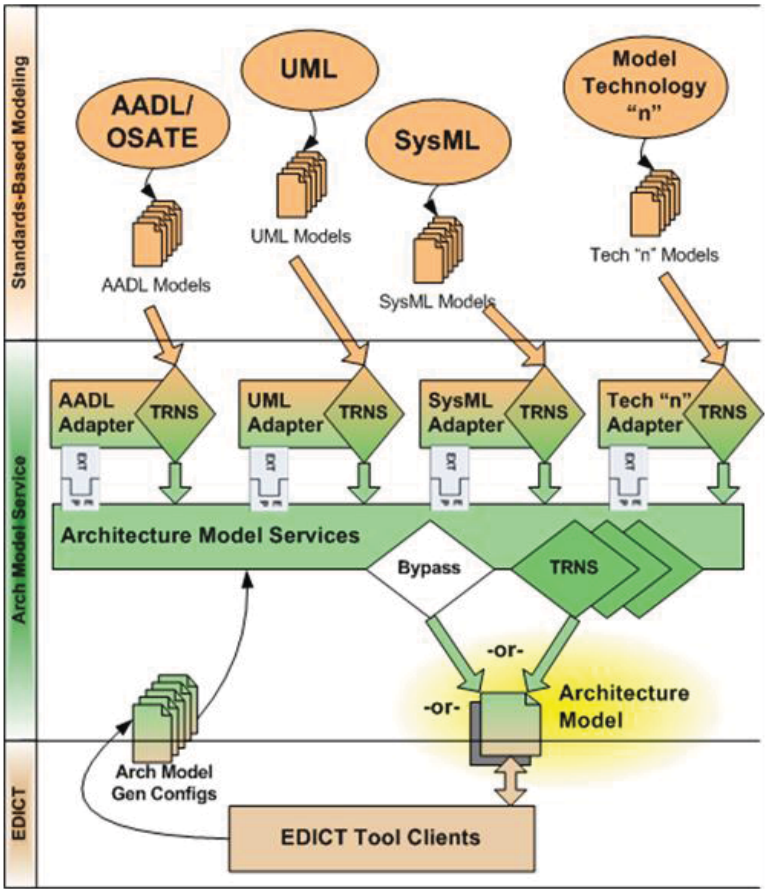

Figure 3: Conceptual View of AADL Model Translation

1.3. EDICT Error Modeling

1.3.1. Approach

The first step in error analysis of a system is to define the errors that may occur, the behavior of the system components given those errors and the error handling or mitigation capabilities of the architecture. EDICT defines the structure of this information for two aspects of error handling: errors that occur in and/or flow through components in the system, and mechanisms that serve to detect and/or mitigate the effects of errors when they occur. Specifically, six distinct models are used to augment the structural architecture model with these error characteristics:

1. ***Error Semantic***
   This model defines the classification of an error that can occur as well as the specific manifestation that the error takes. These semantic definitions are based on the error classes defined in the Customizable Fault Effects Model [5].

2. ***Component Error Descriptors***
   This model defines the set of error semantics that may originate from a given type of component in the system. Each error descriptor associates an error semantic with a model of occurrence (in terms of the probability that the error will occur), and persistence (permanent, transient, etc.).

3. ***Component Error Model***
   This model defines the error characteristics of a specific component in the system architecture model. This model has three primary elements, the first being the specification of a component in the architecture model to which the model applies. This places the error model into a context and allows deployment specific aspects to be incorporated into the error model definition. The second element is the set of "exhibited" errors that describe errors that may originate from a component. This consists of a Component Error Descriptors model (see above) augmented with a set of deployment properties that may refine the way in which errors originating at the component may propagate outward into the system. The final element is the error translation description, which models how errors that *arrive at* a component's input interfaces and influence relationships undergo transformation (if any) and subsequent propagation.

4. ***Error Mitigator***
   This model defines a mechanism that is capable of detecting and/or tolerating errors (e.g., 3-way voter, CRC checker, value range checker). The Error Mitigator model defines the set of error semantics that the mitigator can detect and a separate set of semantics that can be tolerated.

5. ***Component Error Mitigation Model***
   This model defines for a given component all of the error mitigators that operate on a specified set of error sources. These sources include input interfaces and influence relationships. The Component Error Mitigation Model is where Error Mitigators are deployed to component instances in the architecture model.

6. ***Influence Relationship Model***
   Influence relationships describe paths over which error propagations may occur as the result of the component's context (as opposed to a component's physically described interfaces). This model defines influence relationships for an entire system architecture. It has two components: A set of definitions of types of influences and the set of individual instances as they exist in the system. The definition of an influence type includes a set of component types which may exert such an influence and a set of component types which may be influenced. An influence relationship of any defined type may be established between any pair of components that match the specified source and destination component types.

These models have been implemented utilizing Eclipse Modeling Framework. The EMF facilities allow for persistent storage of the models using XML based definitions and run-time representations that are accessible though Java methods.

1.3.2. Error Characteristics Modeling

The EDICT tools supply a graphical editor for each of the error models described above. The graphical editors provide a user interface that allows the user to easily manipulate the contents of the models and to associate them with components of the architecture model that have been imported from AADL.

A screen capture of the Error Semantic Editor is depicted in Figure 4. Its content is simple, prompting the user for the two key attributes of the error semantic model (class and manifestation), as well as providing an unstructured area where the user can provide a more in-depth description of the error and how it should be applied.

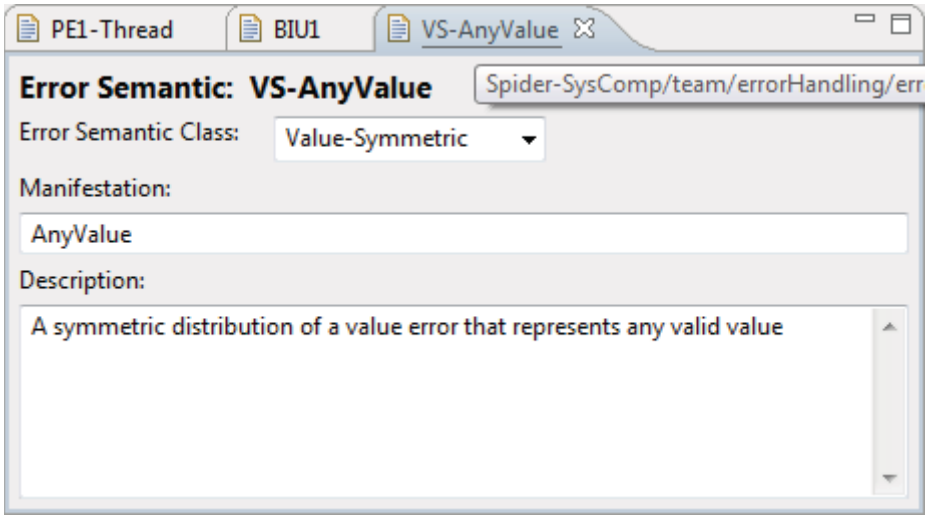

Figure 4: Error Semantic Editor

The EDICT tools provide a set of predefined error semantics that the user may employ directly. New error semantics may be defined and added by the user to address specific needs of the architecture under analysis. The combination of the default error semantics and the user defined error semantics constitutes

an error semantic *library* which is used as the basis for defining component error characteristics and error mitigation mechanism characteristics.

The Component Error Descriptors (CED) editor provides methods for defining sets of error semantics that can be assigned to architecture components. This editor is a multi-page editor, with the first page shown in Figure 5. This page provides a summary of the error descriptors that are being bundled together into the error descriptors set. From this page, new descriptors may be added to the set. When the "New Descriptor" button is pressed, a new descriptor is created and added to the set. Following this, the editor automatically navigates the user to the Error Descriptor page.

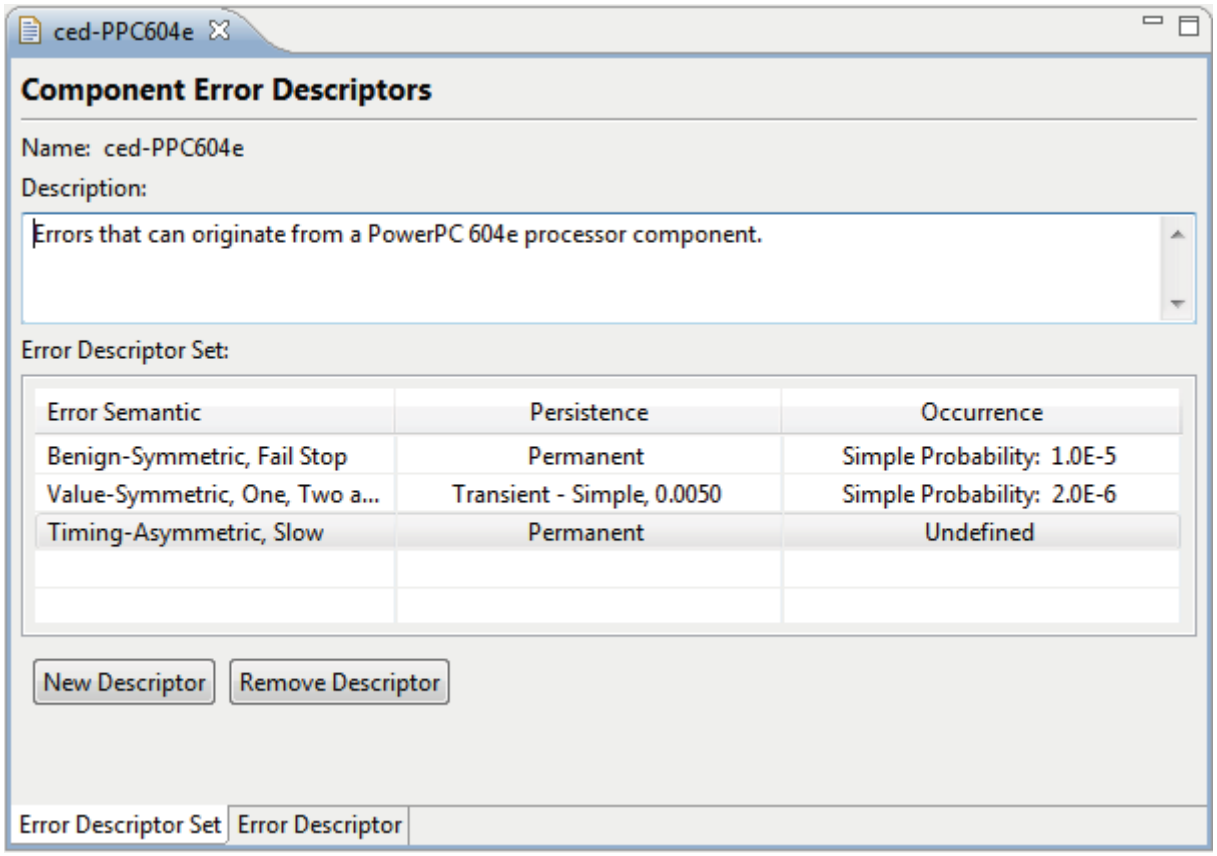

Figure 5: Error Descriptor Set page of the Component Error Descriptors Editor

A screen capture of the Error Descriptor page is presented in Figure 6. This page provides the controls that allow the user to select the error semantic for the descriptor, and then provide additional characterization of the persistence and occurrence of the error.

The CED model that is created for a specific type of component may then be reused as often as is appropriate to define the Component Error Models (discussed next). This supports the paradigm that model information should be captured exactly once where possible. Thus, if it is determined that the error characteristics of a given type of component require modification, this modification may be performed once. The change is then automatically and immediately visible to all Component Error Models that use

the modified CED model as a basis for describing the error characteristics of components of the associated type.

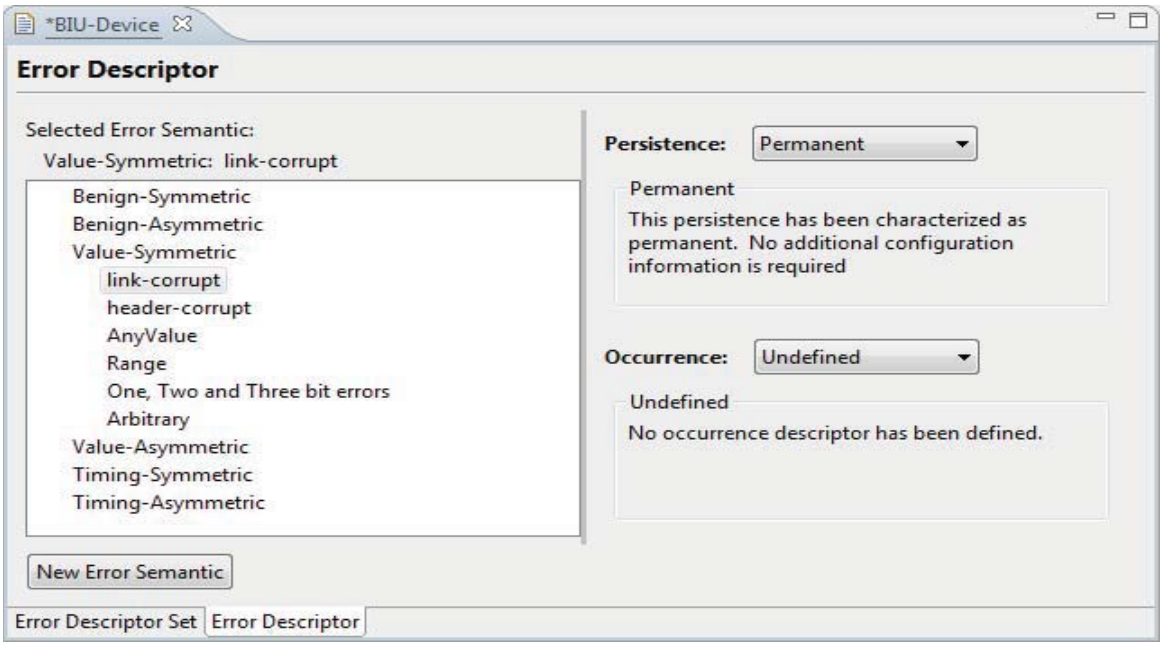

Figure 6: Error Descriptor page of the Component Error Descriptors Editor

The third editor is dedicated to the Component Error Model (CEM), defining the error characteristics of a specific component in the system architecture model. This model has three primary elements, each of which is supported by different pages of the editor. The first page (shown in Figure 7) supports the specification of a component in the architecture model to which the model applies. This places the error model into a context and allows deployment specific aspects to be incorporated into the error model definition. From within this editor page there are hyperlinks with supporting descriptive text that provide an alternate means of accessing the other two editor pages.

The second part of the CEM supported by this editor is the definition of the set of "exhibited" errors that describe errors that may originate from a component. The exhibited error specification consists of a selected Component Error Descriptors model (see Figure 8) augmented with a set of deployment properties that may refine the way in which errors originating at the component propagate outward into the system. The "Output Constraint" property allows the user to precisely define the subset of outputs over which a specific kind of error originating from the component will propagate.

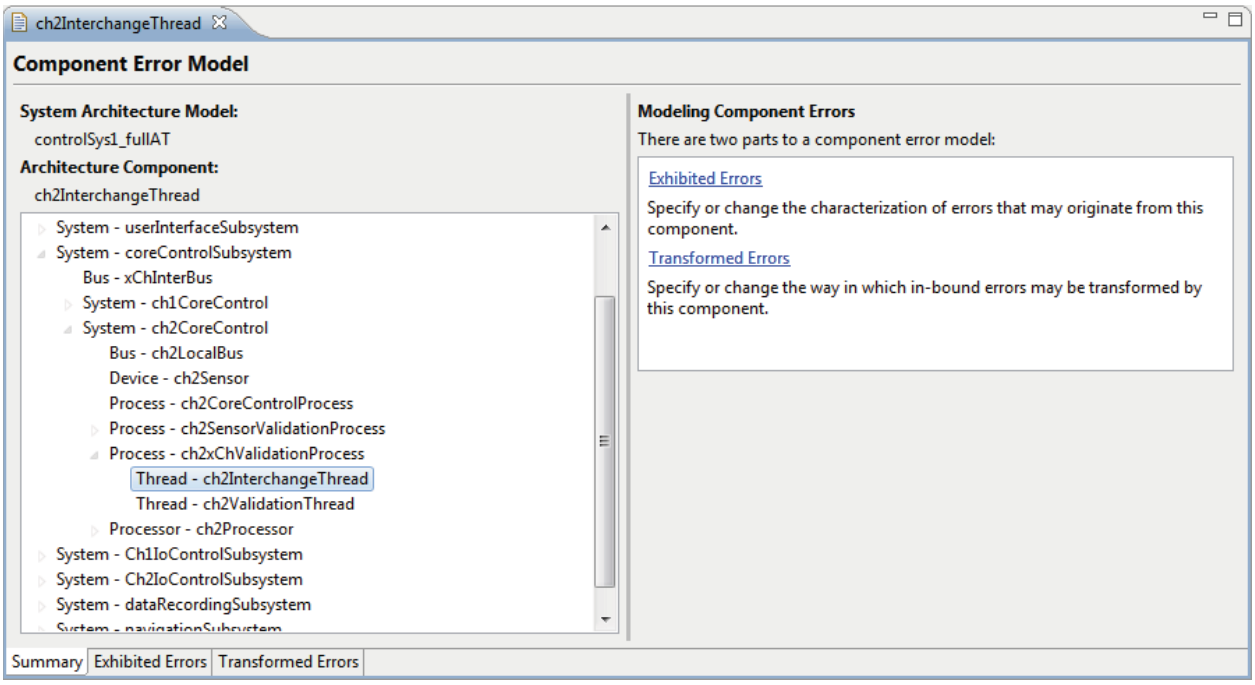

Figure 7: CEM Editor, "Summary" Page

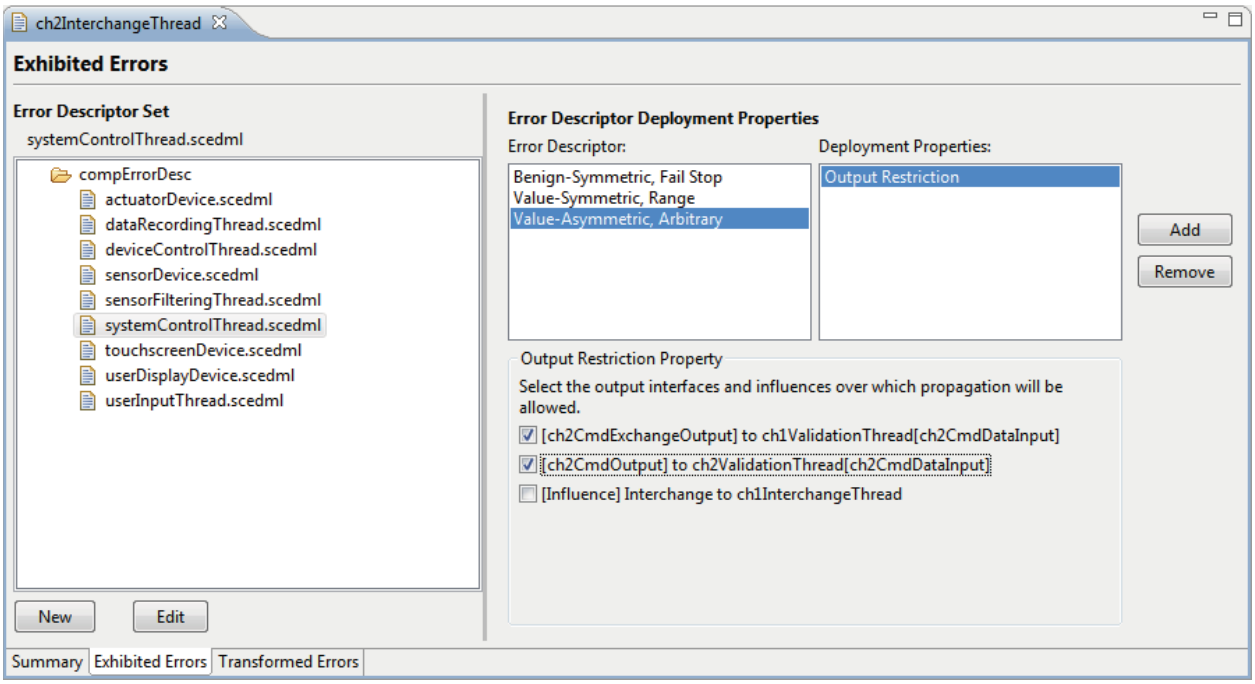

Figure 8: CEM Editor, "Exhibited Errors" Page

The final part of the CEM that is defined through this editor is the error translation description, which models how errors that *arrive at* a component (either directly via input interfaces, or indirectly through

influence relationships) undergo transformation (if any) and subsequent propagation. The "Transformed Errors" page of the CEM editor is depicted in Figure 9.

The error transformation behavior is defined through a set of rules that govern the effect of the error on the components outputs. To define a rule the user first selects a set of error sources with which to work. For those error sources, a set of one or more transformed semantics are defined. The semantics describe the kinds of errors for which a transformation is being specified. Then the resulting error semantic propagated out of each output interface or influence relationship is specified. For each output this result may be a specific semantic, the semantic that was propagated into the component, or no propagation. The user may also define a default rule that defines the transformation behavior for all error arrivals that are not specifically defined in an existing rule. In the future, this editor will also provide us the means to associate probabilities with the transformation targets so that we can model probabilistic selection of the error transformation. The addition of probabilistic data will allow for the development of stochastic analysis algorithms for error containment and propagation effects.

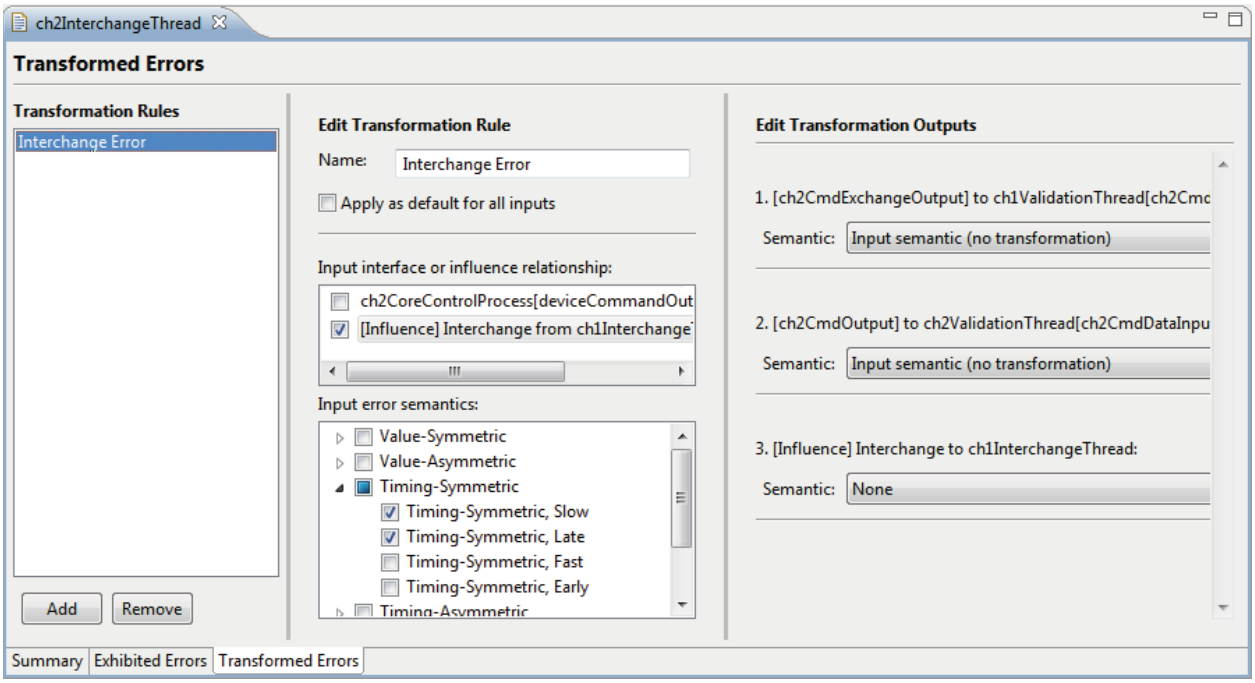

Figure 9: CEM Editor, "Transformed Errors" Page

1.3.3. Error Mitigation Modeling

The Error Mitigator Model (EMM) defines a mechanism that is capable of detecting and/or tolerating errors. A mitigation mechanism might be a 3-way data voter, a CRC checker, or a value range checker. The Component Error Mitigation Model (CEMM) defines for a given component all of the component's error mitigation mechanisms that operate on a specified set of error sources. Input interfaces and *influence relationships* are supported as inputs. Influence relationships describe paths over which error propagations may occur as the result of the component's architectural context (as opposed to a component's described interfaces).

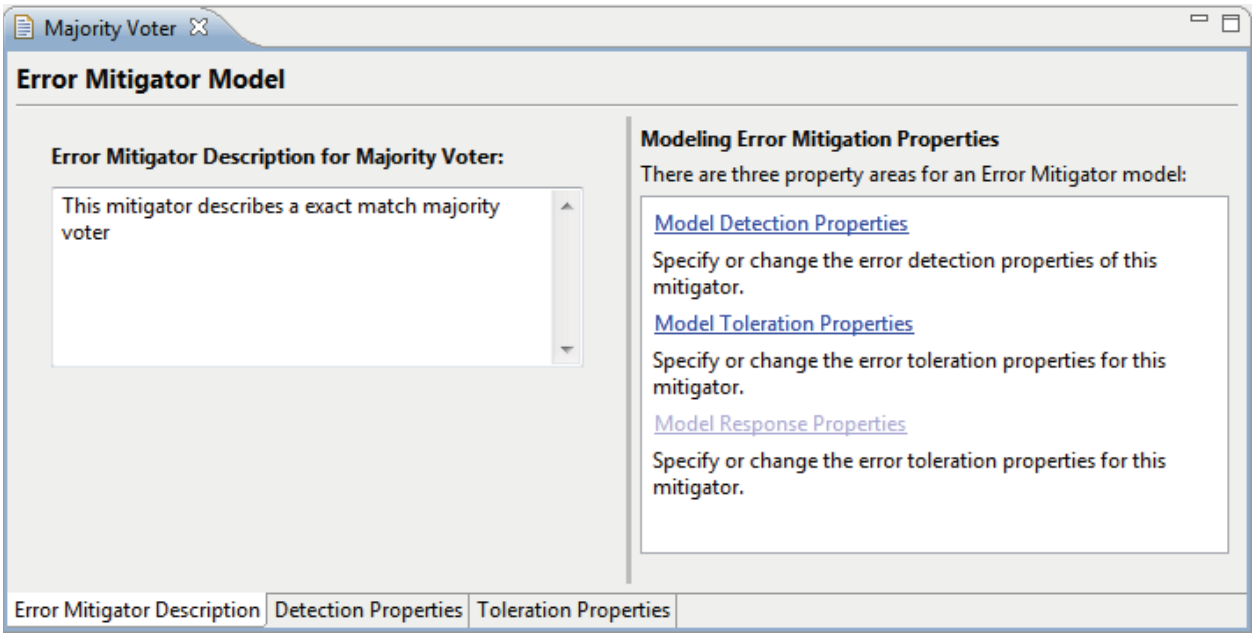

Figure 10: EMM Editor, "Error Mitigator Description" Page

The EMM editor allows the user to define specific mitigation mechanisms that represent the error detectors and error toleration mechanisms used in the architecture. Figure 10 depicts the first page of the EMM editor, which allows the user to provide a textual description of the mechanism being modeled. The Detection Properties page (see Figure 11) provides the user with the ability to define those error semantics that the mechanism is capable of detecting. The detection characteristics for a particular error semantic may be further described as being deterministic (the error is always detected), or probabilistic.

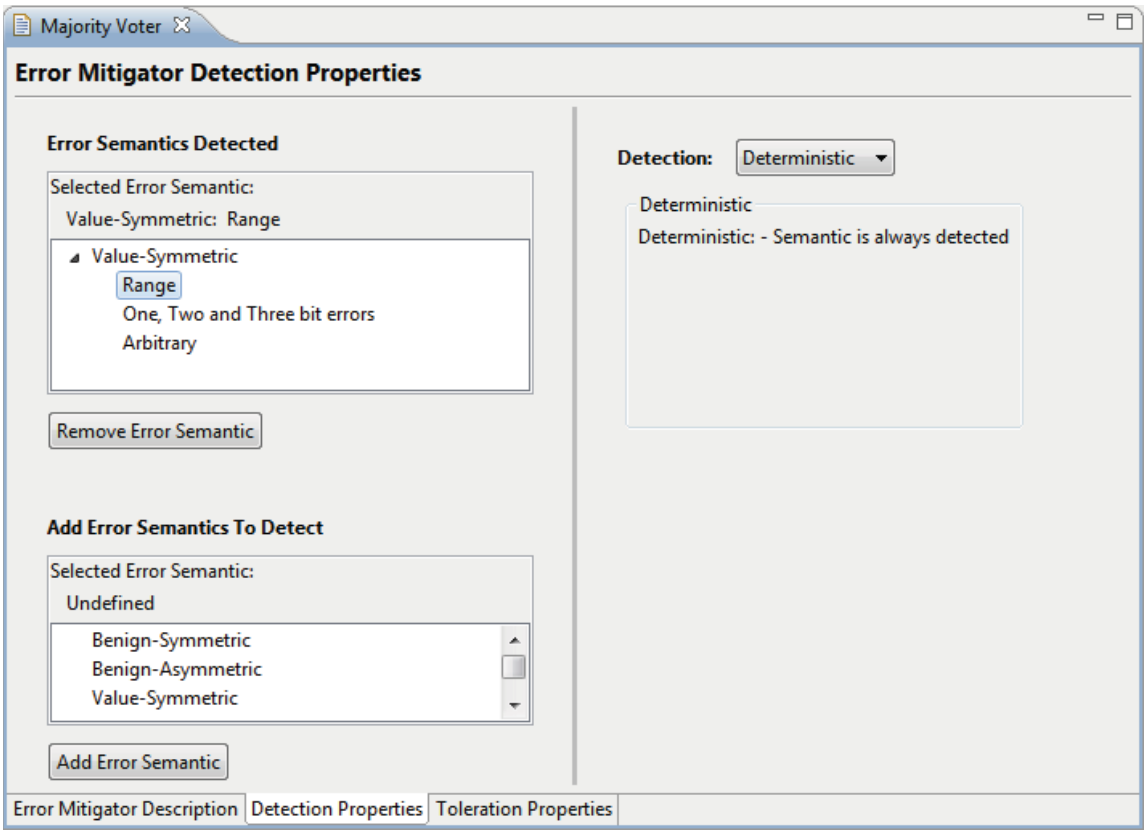

Figure 11: EMM Editor, "Detection Properties" Page

The Toleration Properties page provides the user with the ability to define the error semantics that are fully handled by the mechanism. A screen shot of this editor page is presented in Figure 12. Note that we currently assume that toleration is always deterministic. Controls are provided in the UI for the user to move error semantics between the tolerated and not-tolerated categories. (Note that, by default, all known error semantics fall into the not-tolerated category; the user must explicitly add a semantic to the tolerated category.)

As with the Component Error Descriptor model editor, the error semantics from which the user may choose are collected from the "library" of semantics that have been created. As the user adds or removes error semantics from this library, options likewise change for semantic assignment to detection and toleration properties of a given error mitigator. In addition, like the error semantics, it is the case that the group of defined error mitigation mechanisms forms a mitigation mechanism library that may be referenced in subsequent models of the error handling infrastructure.

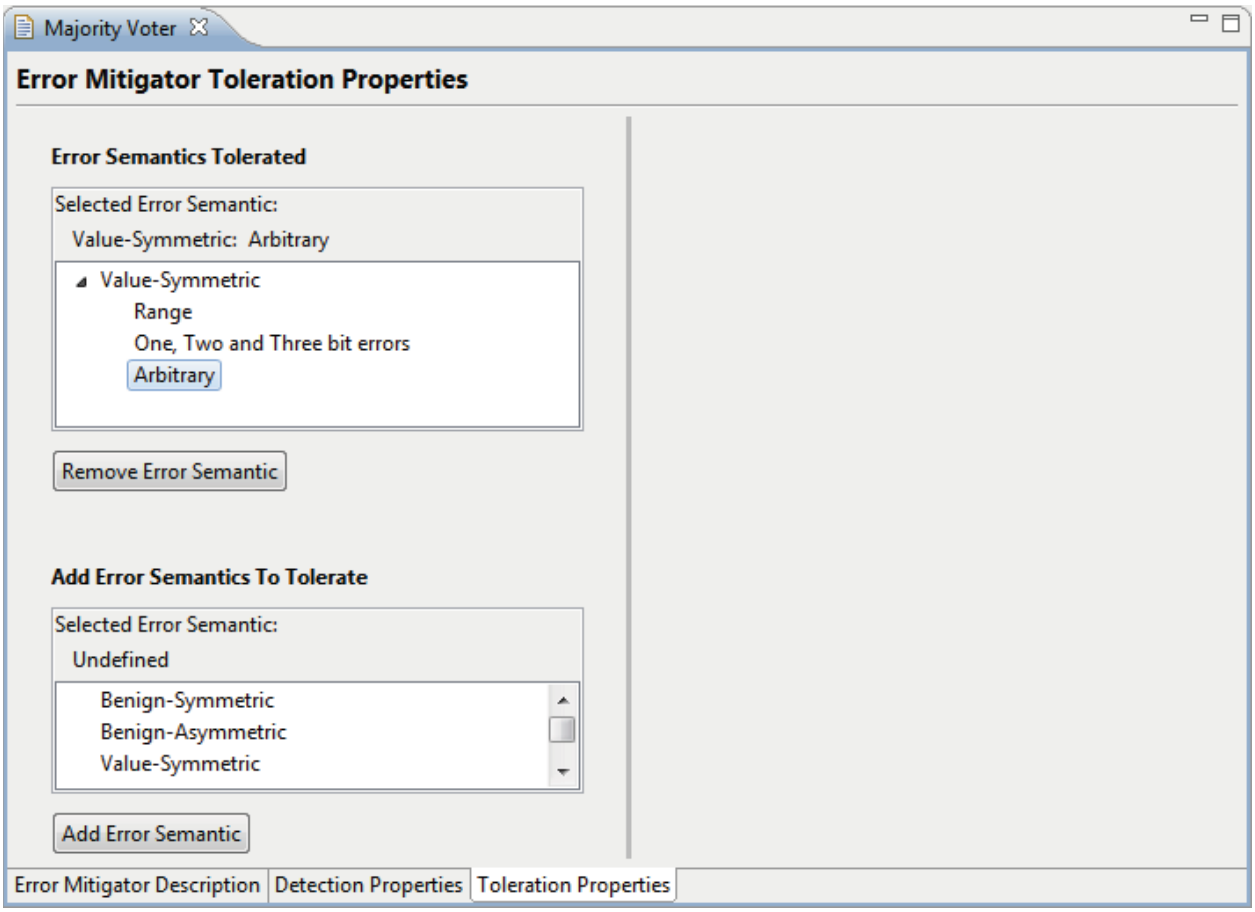

Figure 12: EMM Editor, "Toleration Properties" Page

The CEMM Editor was also implemented as a multi-page model editor. Figure 13 is a screen shot of the Component Selection page of the editor which allows the EDICT user to associate the CEMM with an architecture component and displays a summary of the error mitigation mechanisms currently deployed by the selected component.

Once a system architecture component has been selected, error mitigators may be deployed to the error sources that are inputs of the component to mitigate the effects of errors that enter the component. Error mitigator deployment is specified on the Mitigator Deployment Tab and can be accessed either through the hyperlink on the right hand side of the Component Selection area or through the tabs at the bottom of the editor.

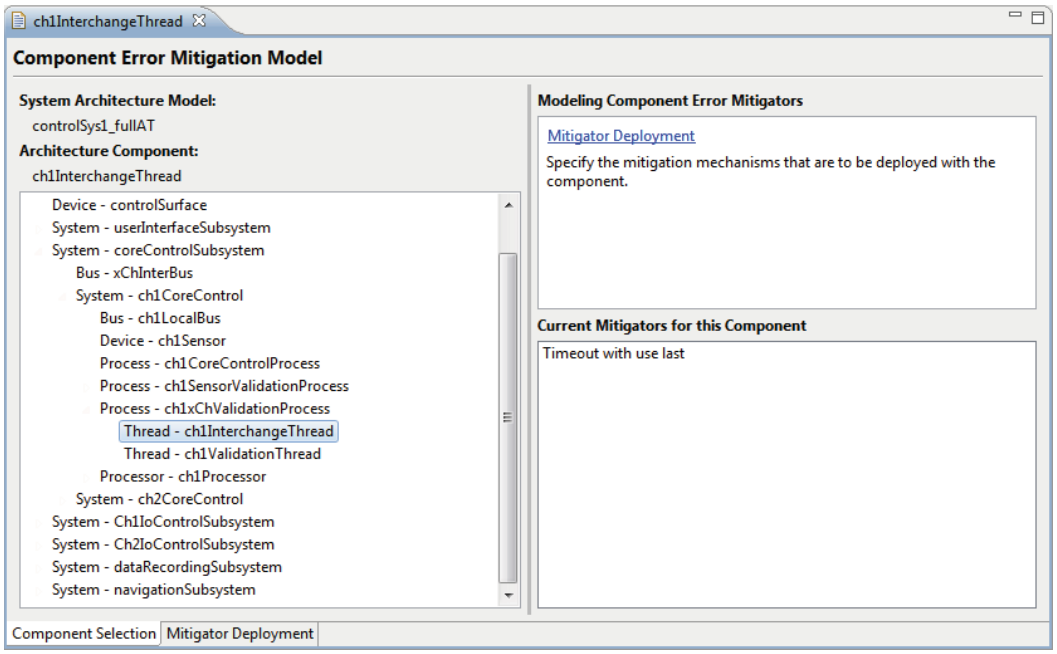

Figure 13: CEMM Editor, "Component Selection" Page

Figure 14 shows the Mitigator Deployment page of the CEMM editor. The input interfaces and other potential error sources such as influence relationships from related system components are listed on the left hand side. These error sources are determined dynamically based on the specific component that was selected, and thus will change if the user returns to the Component Selection page and chooses a different component. This context sensitivity ensures that the CEMM model is consistent with the structure of the system's architecture.

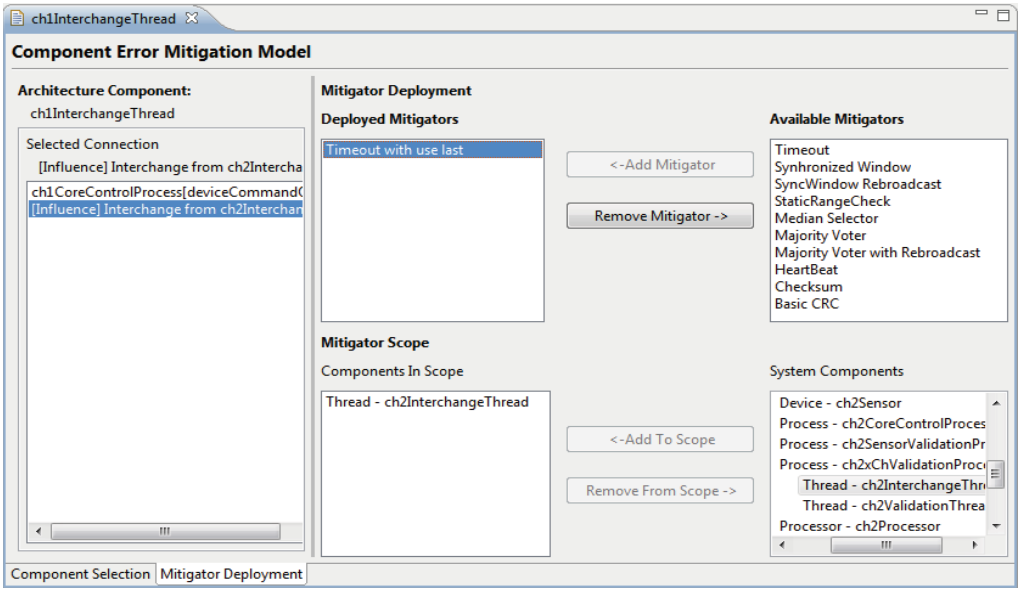

Figure 14: CEMM Editor, "Mitigator Deployment" Page

The user is then able to select an error source and add/remove error mitigators using the top set of controls on the right side. The set of widgets in the Mitigator Deployment area list the current mitigators that are deployed for the selected error source and the available set of mitigators that are contained in the error mitigator library for the project. The editor provides a set of buttons to add/remove mitigators from the error source.

The scope of the mitigator can be set once a mitigator is deployed to a component. The *mitigator scope* is used to describe the set of components for which the mitigator is able to detect errors. For instance, if a cyclic-redundancy-check (CRC) mitigator is deployed, it is only able to detect and correct errors that occur in components that lie on the path between where the CRC code was originated and the location of the CRC checking mechanism. System components may be added or removed from the mitigation mechanism's scope using the associated controls. The mitigation scope may then be used to provide additional accuracy and specificity to analyzers that are responsible for determining whether or not a mitigator is capable of detecting and/or tolerating a given error.

1.3.4. Error Modeling with Influences

The Influence Relationship Model (IRM) allows the specification of influence relationships by which an error at one component may induce an error in another component by means other than physical interfaces between the two components. This sort of relationship may reflect existing architecture constructs (for example, bindings between processes and threads or processors and memory) or environmental influences between hardware components or from a component that represents the operational environment.

The IRM editor is implemented as a multi-page editor with two pages: A page for the definition of types of influences and a page for specifying individual instances as they exist in the architecture under analysis. Influences and influence types for an entire system architecture may be specified in a single editor. Figure 15 illustrates the Influence Relationship Types page, in which all of the existing influence types are listed and may be edited or removed, and new types may be created. A type may be annotated with a name and a description, and the sets of component types that are eligible to be source and destination components are selected from lists of available types.

Figure 16 illustrates the Influence Relationships page, in which influence relationships of any defined type may be established between eligible components. When an influence relationship type is selected on this page, only components whose types are included in the list of valid source component types for that influence type are available for selection. When an influence type and a source component have been selected, the available destination components are filtered according to the list of valid destination component types for the influence type. A set of destination components may be selected, signifying that influence relationships exist between the source component and each of the destination components.

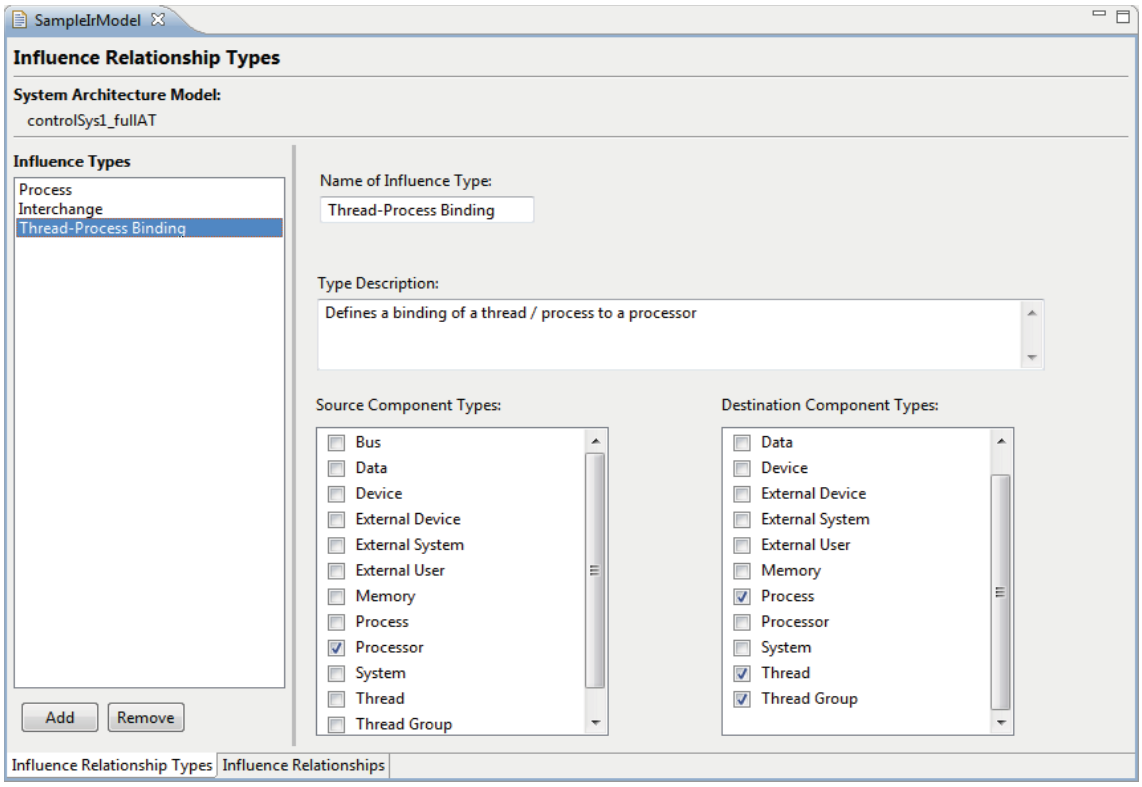

Figure 15: IRM Editor, "Influence Relationship Types" Page

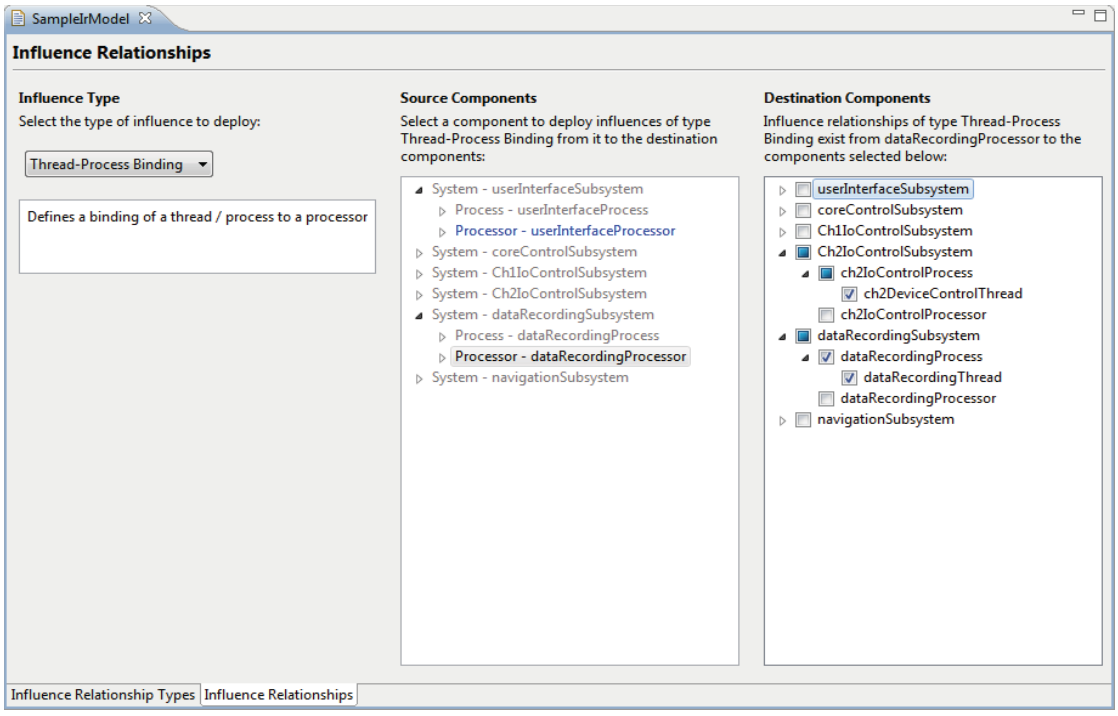

Figure 16: IRM Editor, "Influence Relationships" Page

1.3.5. Error Model Management

The EDICT Tool suite provides a method for managing all of the error and error mitigation models that are created to describe the error behavior of the system. The Error Handling Aspect Editor provides a single place where all of the models that are associated with architecture components can be managed from. This editor provides functions for creating, editing and configuring all of the model types along with integrated status reporting based on the model verifiers.

The Error Handling Aspect Editor is implemented as a multi-page editor with three pages: A page for associating Component Error Models (CEM) with system components, a page for associating Component Error Mitigation Models (CEMM) with system components, and a page for associating an Influence Relationship Model with the selected system architecture model.

Figure 17 illustrates the page for CEM selection. A component may be selected from the component tree on the left to view its aspect status. If one or more CEMs exist for the selected component, one may be selected using the status panel on the right. The selected CEM will be associated with the architecture model as augmenting information available to the error handling analyzers. A component may be designated to have no CEM associated with it. Analyzer handling of this designation is determined by analyzer configuration. The status panel also permits creation of a new CEM or editing of the selected CEM. Icons in the tree view indicate the verification status of a component's CEM association or that of its children. Each model type has an associated verifier that checks for the structural integrity of the model when the aspect editor is launched or models are saved. Icons status is propagated up the tree view so that sub tree elements with errors or warnings are not hidden when the branch is collapsed.

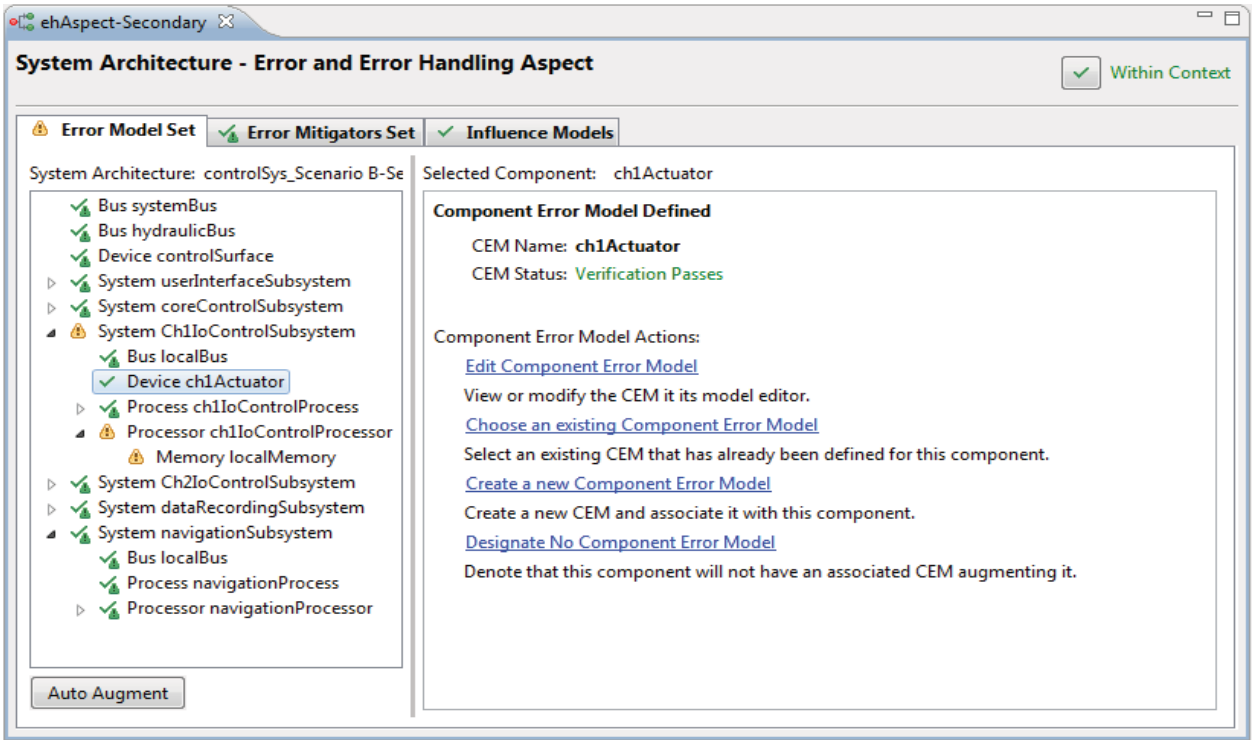

Figure 17: Aspect Editor, "Error Model Set" Page

The auto-augment feature shown in Figure 18 automates the process of associating CEMs with components. The user may select two options: 1) to designate that there is “No CEM” for all components that do not currently reference a CEM, or 2) the tool will find any existing CEMs that could apply and complete the architecture references where possible. The auto-augment features provides a way to generate an analyzable architecture model without requiring the user to fully specify the error models for all of the architecture components. This is useful when performing incremental development and analysis. The features and interface of the page for CEMM selection are identical to those of the page for CEM selection.

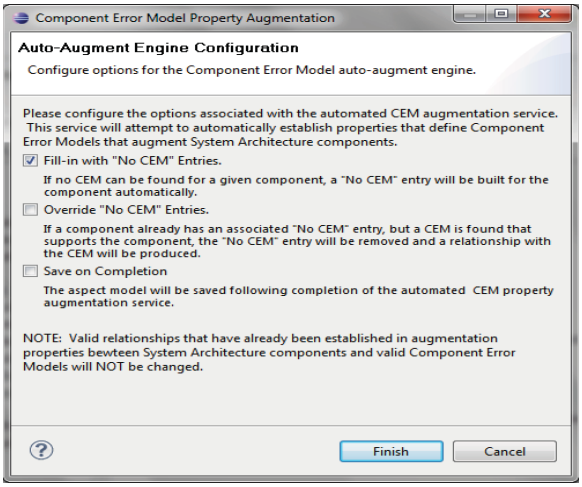

Figure 18: Aspect Editor, "Auto-Augment Engine Configuration" Dialogue

Figure 19 shows the final Error Handling Aspect Editor page for Influence Relationship Model (IRM) selection. This page identifies the IRM currently associated with the system architecture model and reports the verification status of the IRM. From this panel the current IRM may be edited, a new IRM may be created, or an existing IRM may be selected.

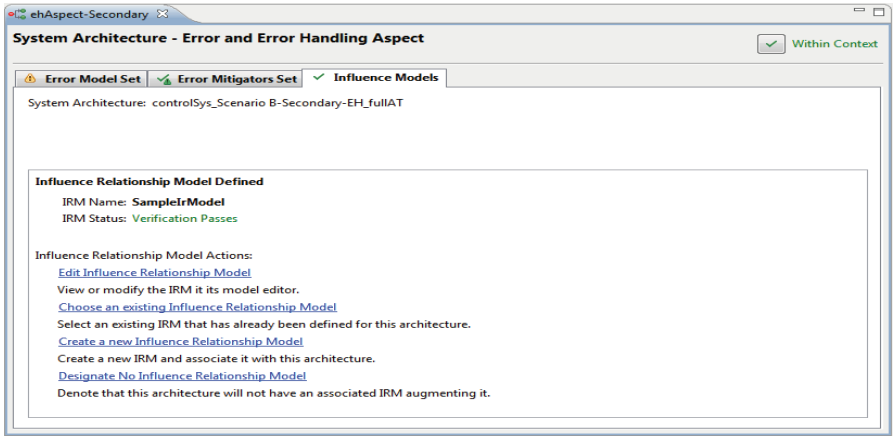

Figure 19: Aspect Editor, "Influence Models" Page

1.4. EDICT Error Propagation Analysis

1.4.1. Approach

The EDICT error handling analysis tools provide the capability to trace potential error propagation paths and determine architecture components exposure to different error semantics based on information in the component error and error mitigation models. This analysis is performed on a specified set of errors and components exhibiting the errors, and follows each error's propagation over interfaces or influence relationships, its mitigations (toleration or detection), and its transformations at each component.

An error propagation scenario consists of a specific set of components, a set of exhibited errors for each component (that must be a subset of the exhibited errors specified in its Component Error Model), and a set of outgoing connections (a subset of the outputs permitted by the output restrictions in the Component Error Model, if any) for each component from which the errors will propagate. If an error semantic reaches a connection that is an input for a component, any mitigator for that semantic and input is applied, and if the semantic is not tolerated, error transformations are applied. The semantic resulting from any applicable transformations is propagated out of each output connection. This process continues, accounting for cycles in the propagation, until all resulting paths have been followed for the appropriate error semantics.

The result of this computation is a worst-case perspective on the potential error exposure of each component given the initial scenario, since probabilities of error exhibition, transmission, transformation, and mitigation are not yet accounted for. The analysis constructs a summary for each component, a propagation trace for the entire system architecture, and a tree of the paths by which each error semantic reached each affected component.

1.4.2. Error Propagation Analyzer

The Error Propagation Analyzer (EPA) consists of a page for configuring an error propagation scenario and a page for performing the analysis and examining the results. Figure 20 shows the configuration page. Any number of components that have Component Error Models defined may be selected from the system architecture diagram as error sources. When a component is selected, the exhibited errors listed in its Component Error Model may be refined by selection of a subset of the errors. For each of these errors, the outgoing connections of the component (interfaces or influence relationships) are presented, and may be selected if there is no output restriction defined in the Component Error Model for that particular error or if the output restriction permits propagation on that connection.

The Error Propagation Analyzer has two models of operation: Normal and Pessimistic. If Normal mode is selected, propagation will halt if Component Error Models are missing or incomplete for components encountered in error propagation. If Pessimistic mode is selected, such errors will be reported, but propagation will continue using a worst case assumption that the error semantic propagates out of any unhandled outgoing connections without transformation.

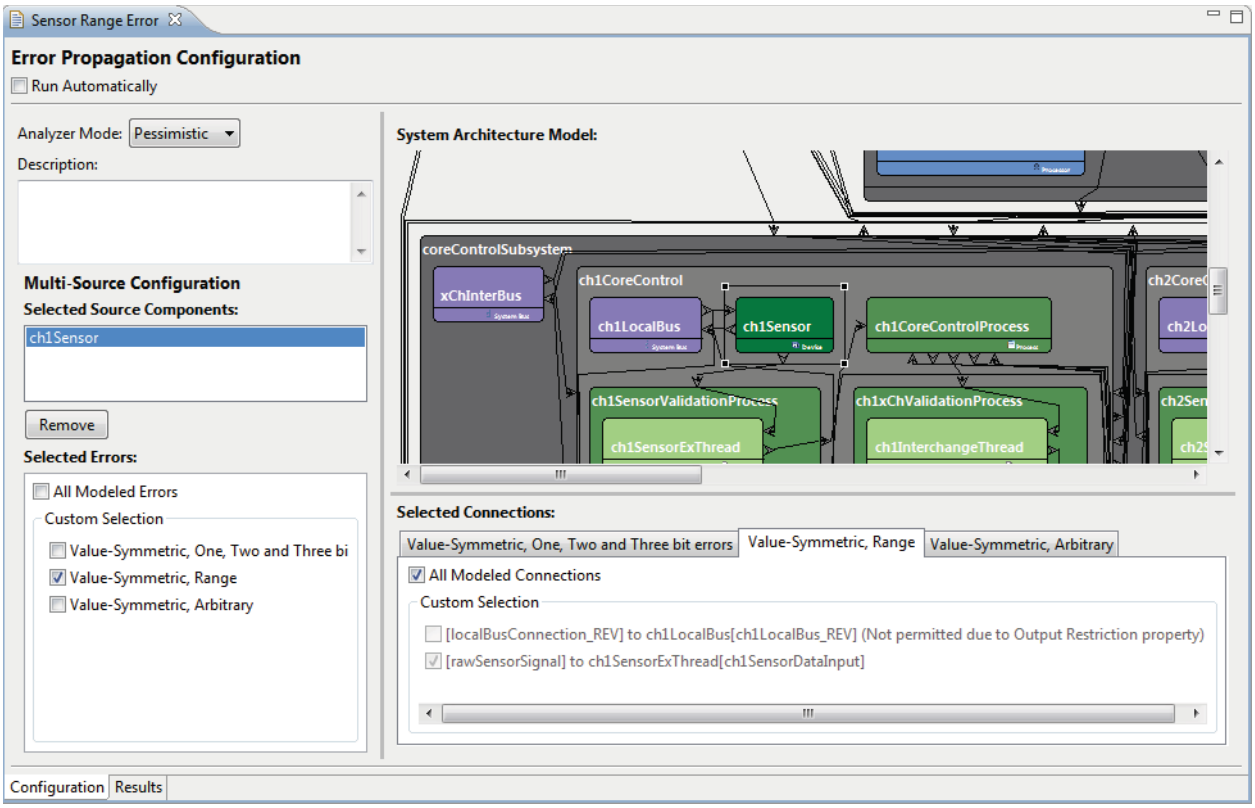

Figure 20: EPA Editor, "Configuration" Page

Figure 21 shows the Analysis Results portion of the Results page. This view presents a trace of all error paths throughout the system architecture for the current analysis scenario. Components and interfaces are color-coded to indicate whether an interface has transmitted any errors (red lines) and whether a component has been affected by (red), tolerated (green), or detected (yellow) any errors. This highlighting shows the paths that the errors take from the source of the errors until they are tolerated or reach the system boundary. If multiple error semantics are selected at the source component then the traces for each of the semantics are overlaid on the diagram.

When a component in the diagram is selected the error exposure summary (on the right hand side) presents a list of error semantics that arrived at the component. For each error semantic the immediate source of that error is shown. When a semantic is select from this list the Error Details displays (bottom right) displays propagation related details such as if the semantic was detected or tolerated and if so by what mechanism.

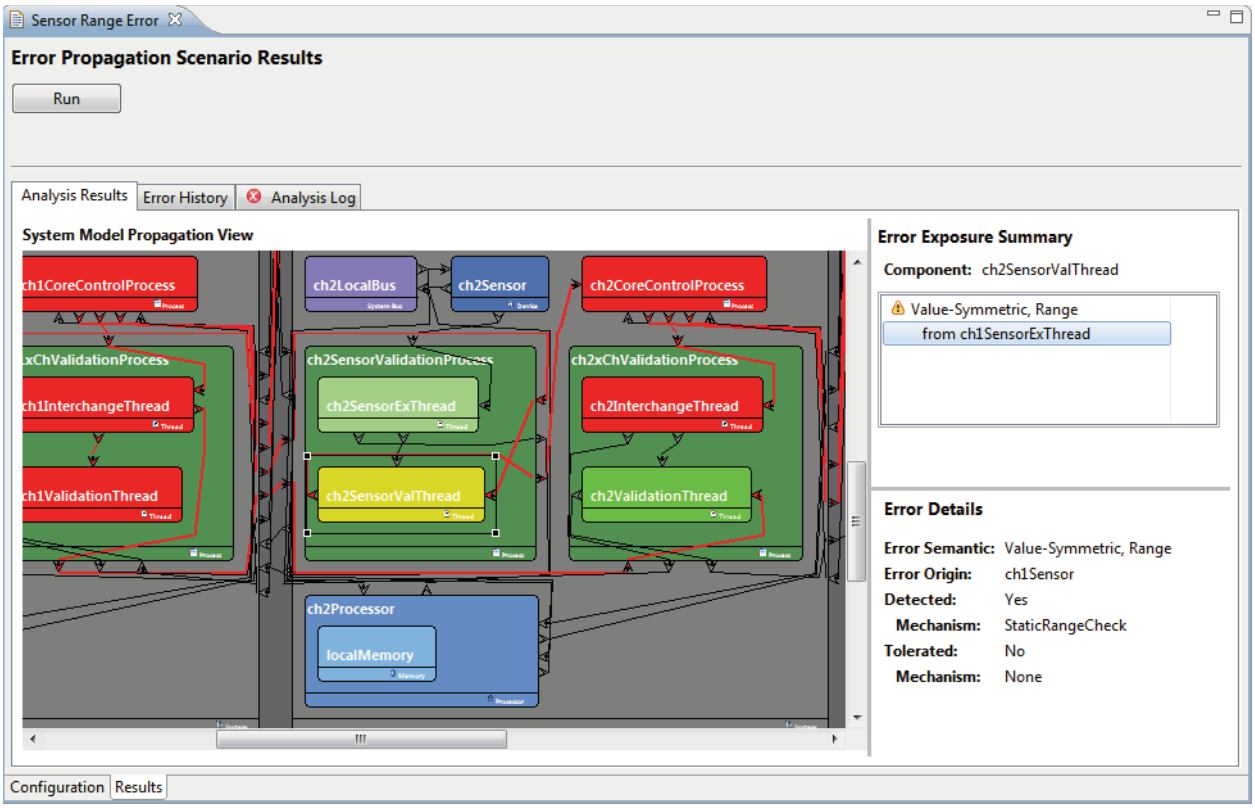

Figure 21: EPA Editor, "Results" Page, “Analysis Results” Tab

Figure 22 illustrates the Error History portion of the Results page. This view shows all paths by which a selected error semantic reached the selected component. This permits a quick and thorough diagnosis of a specific threat to a particular component. The diagram is color-coded like the system architecture diagram in the Analysis Results view and additionally depicts influence relationships in the propagation paths and provides visual indication of error transformations. Mouse-over text for the arcs and nodes in the diagrams provides quick access to information on the semantics involved in a path and any mitigation that occurred. This example depicts a case where there are two propagation paths from the Ch1Sensor to the Ch2InterchangeThread for the Fail Stop error semantic. On one path the error is detected but not tolerated (yellow component) and on both paths the error is transformed on its way to the Ch2InterchangeThread (components that contain the gears icon). The dynamic mouse-over displays provide more details on each step of the propagation.

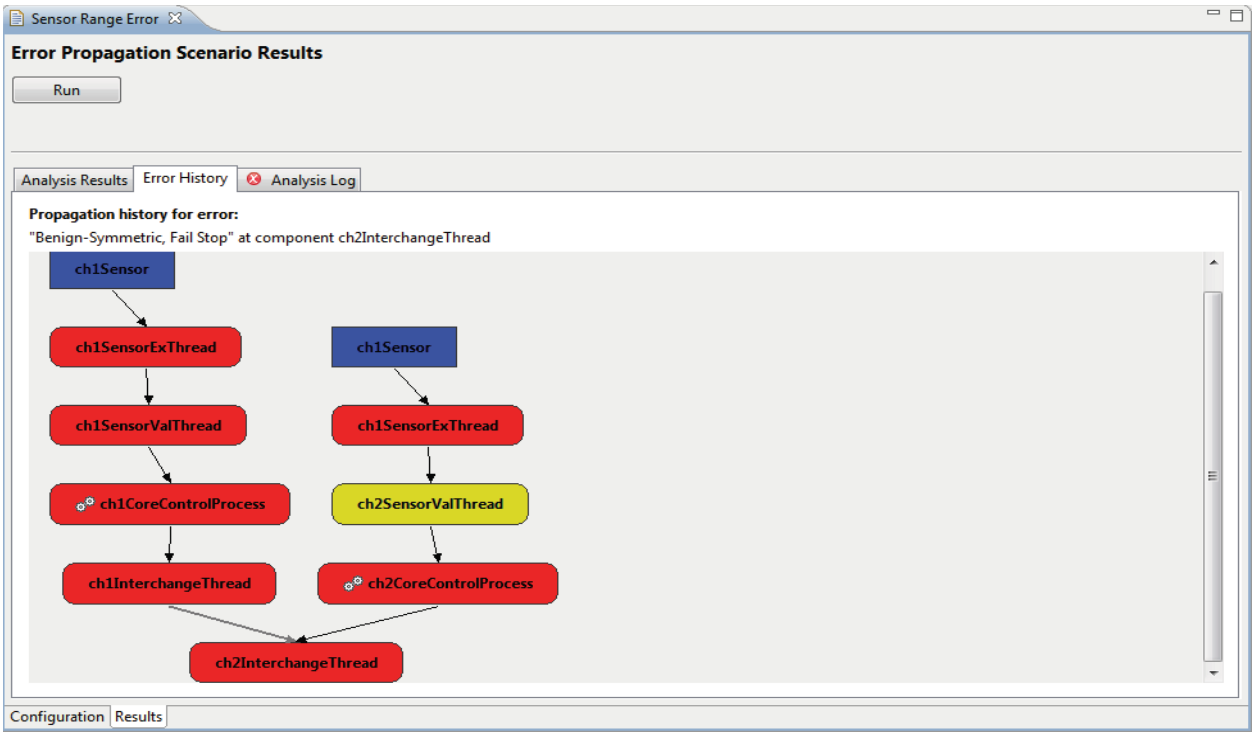

Figure 22: EPA Editor, "Results" Page, “Error History” Tab

Figure 23 shows the Analysis Log portion of the Results page. This log reports any warnings or errors pertaining to problems encountered with the execution of the analyzer. Problems reported include the absence of expected Component Error or Error Mitigation Models or data that is missing within the models.

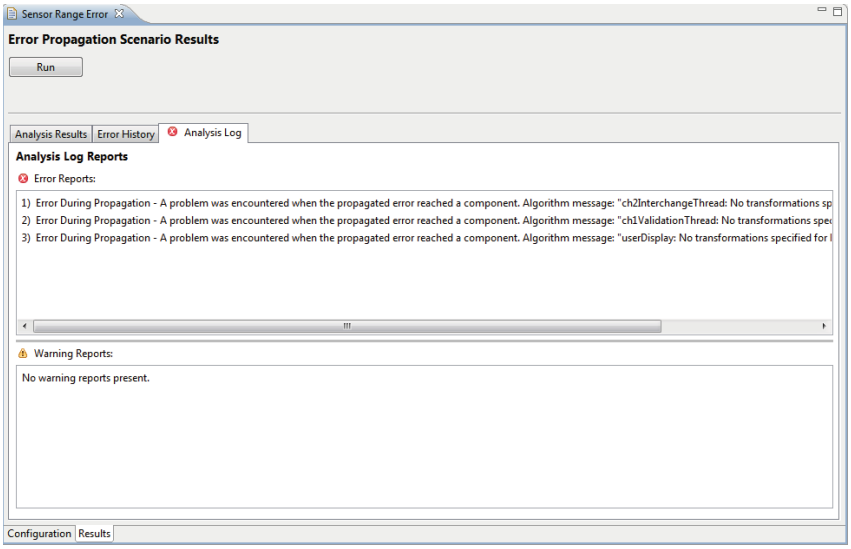

Figure 23: EPA Editor, "Results" Page, “Analysis Log” Tab

2. Analysis of the SPIDER Architecture

In order to evaluate the error modeling and analysis capabilities of the EDICT tools we have constructed error models and performed propagation analysis on a 3x3 Spider architecture. This effort used the AADL model developed by the team under this effort. Our analysis concentrated on the PE Broadcast Protocol and the error detection and mitigation that is applied to data flows through the ROBUS architecture. These models of mitigators and data flow assume that the ROBUS elements are synchronized and in a steady state.

We developed error models for the Bus Interface Units (BIUs), Redundancy Management Units (RMUs) and the Processing Elements (PEs). These models were used as sources of errors that were then propagated through the architecture based on the PE Broadcast data flows. We were able to evaluate the ability of the EDICT tools to represent the expected error propagation behavior and error mitigation behavior as defined in the protocol. The error sources considered representations of errors in the value and time domains, along with both symmetric and asymmetric error distributions.

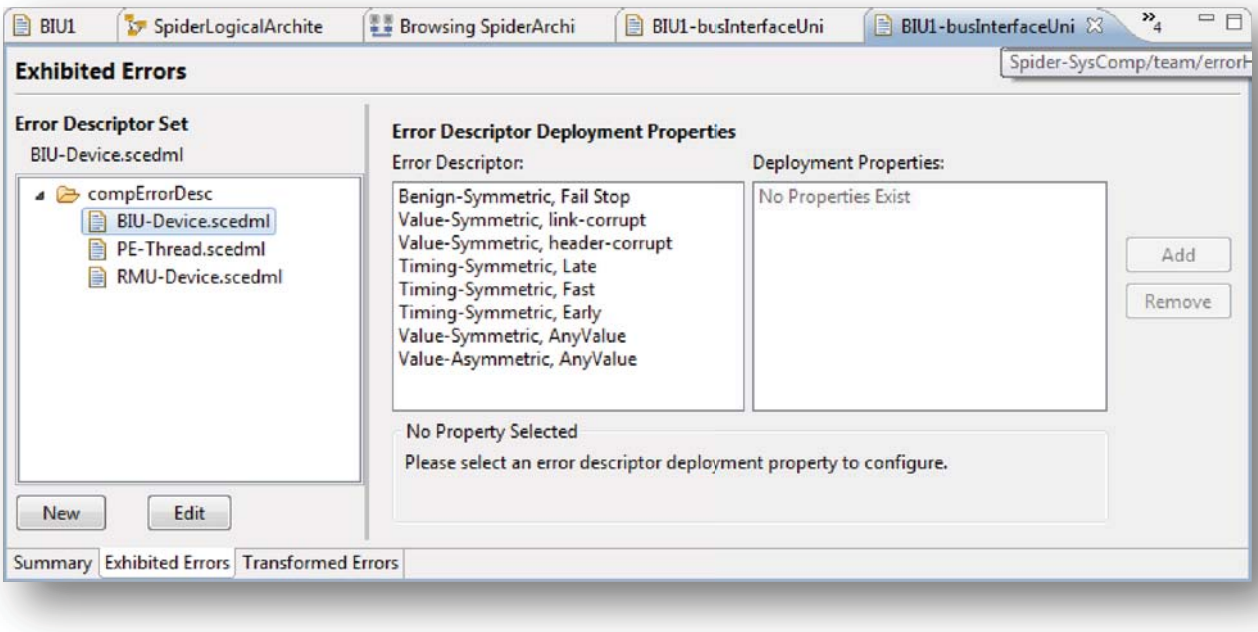

Figure 24 - BIU Error Model

Figure shows the error model for the BIU that was developed in the EDICT Component Error Model Editor. The semantics cover both simple failure modes such as fail stop and more complex errors such as the Any-Value symmetric and asymmetric errors. The error set shown in Figure 24 represents the error semantics that may be originated from the BIU on the data paths that are associated with the PE Broadcast Protocol. They will be refined and expanded over the development period of the project. Similar models were developed for the PEs and the RMUs.

The component error model is also capable of modeling error transformations that happen in the component when errors arrive or when they are processed by error mitigators that are deployed to the component. Figure 25 is a screen capture from the error transformation tab on the Component Error Model editor for the BIU. This editor tab allows the definition of transformation rules that describe the relations between error semantics that arrive at the component inputs and the semantics that are propagated out of the component.

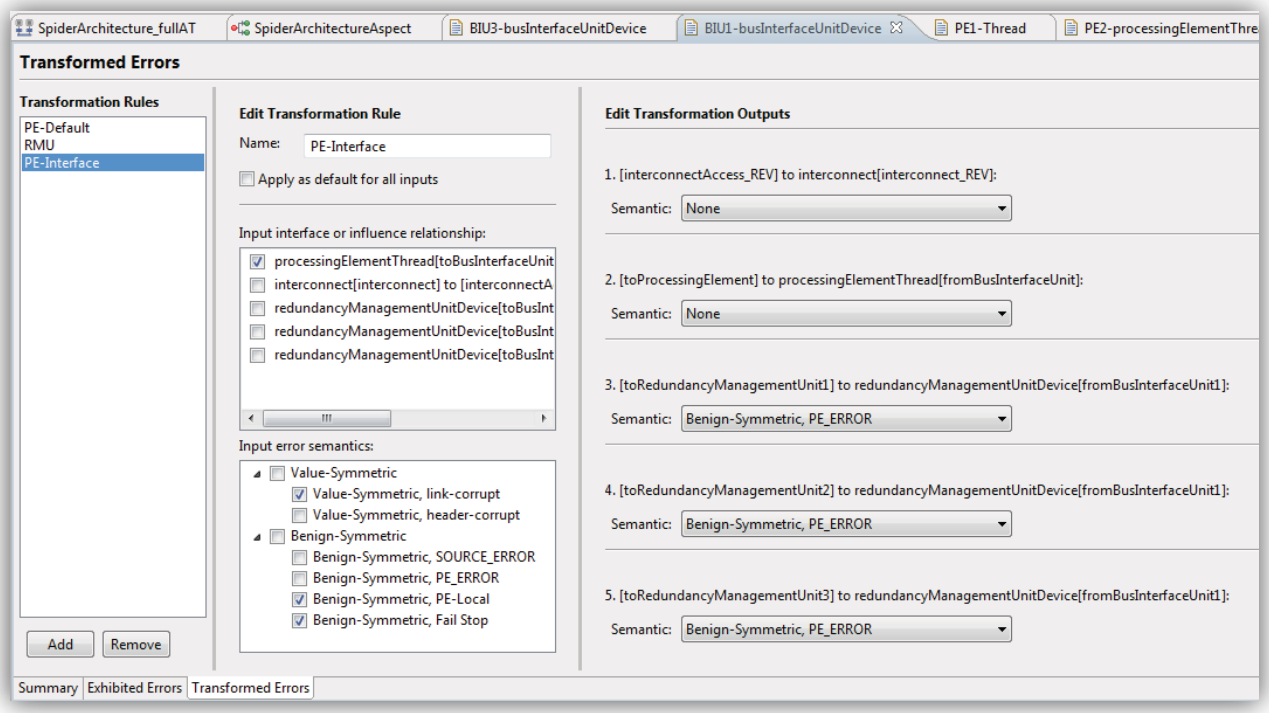

Figure 25 - BIU Error Transformations

These rules describe the transformation behavior and the routing of error flow through the component. We have developed transformation models for the BIU and RMU components that address the data and error flow in the PE Broadcast protocol.

We used these techniques to model the error messages that ROBUS uses to inform downstream nodes of the results of error detectors. EDICT does not provide a distinct capability for representing these error reports, so we have defined additional error semantics that are used to represent the error reports that ROBUS introduces to the normal data flow when detectable error conditions are encountered. This technique is useful because it represents the continued flow of the effect of the source error and allows downstream error detectors and mitigators to properly respond to the error messages.

Error mitigators were developed that represent the error detection and toleration characteristics of the Communication, In-line and Cross-lane checks that are performed in the ROBUS for the PE Broadcast protocol. We have developed an initial set of mitigators and deployed them to the BIU and RMU

components using the Component Error Mitigation Model capabilities in the EDICT tools. Figure 26 shows the error mitigators that have been deployed to the BIU. They are able to represent the Communications Checks, the In-line Checks and the Cross-Lane checks that the BIU performs.

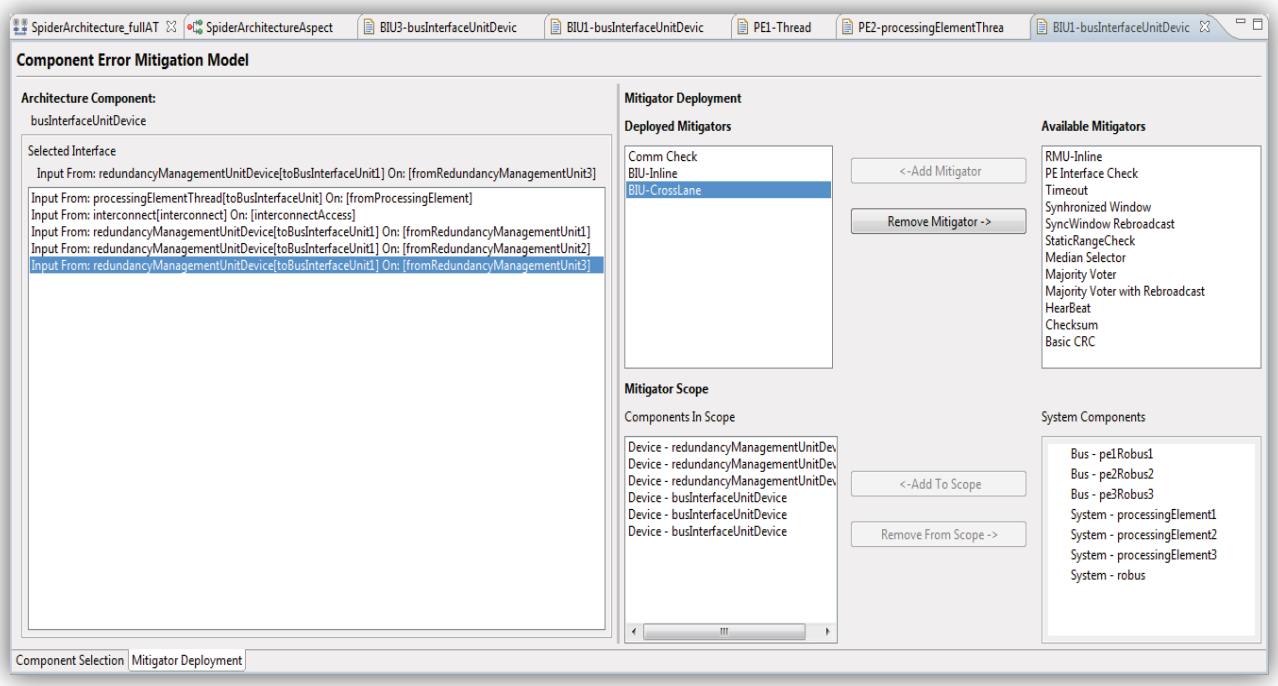

Figure 26 - BIU Error Mitigation

With these models in place we conducted an initial set of error propagation analyses using the PEs, BIU and RMUs as error sources. The propagation analysis evaluates single error manifestations and determines the error flow path according to the PE Broadcast data flows and the error mitigation mechanisms that have been deployed in the ROBUS model.

Figure 27 is a screen shot from the propagation experiment that uses BIU1 as the source of error. This analysis shows a varied set of error semantics propagated with varying impact to the system. Some errors are detected and then passed on with benign semantics in the form of error messages, some are tolerated by the receiving BIUs while others are propagated though the entire architecture to all receivers. The error propagation analyzer also supports the visualization of the history of specific error semantics through the propagation.

By constructing several of these scenarios with the various ROBUS components and Processing Elements as error sources we were able to develop an error profile for the PE Broadcast Protocol that represents the error behavior of the ROBUS when exposed to the error set we have defined. This error profile is useful when representing the ROBUS as a component in architecture models at higher levels of abstraction.

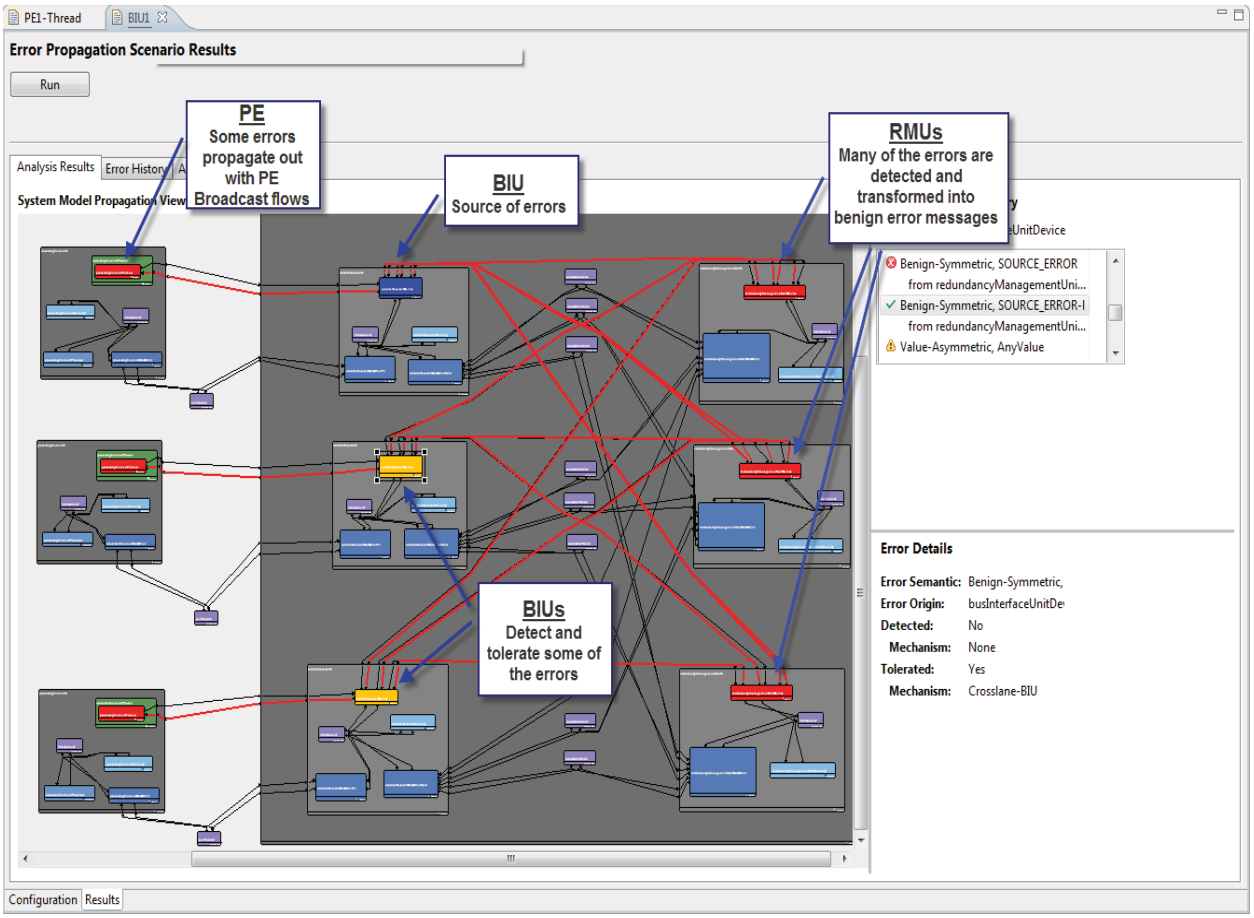

Figure 27 - Error Propagation from a PE

3. Layered Error Modeling

Based on the results of our ROBUS architecture analysis we began to explore concepts for layered modeling and analysis of architectures at multiple levels of abstraction. Our initial efforts concentrated on using the ROBUS element in the Spider model, condensing all of the error propagation analysis on the detailed model into a single set of Component Error and Component Error Mitigation models that represents the characteristics of the bus as a whole. This model was then integrated with a simple application architecture that exchanges sensor data across the ROBUS. The goal of this work is to develop approaches for representing errors and semantics that are consistent at many levels of abstraction.

A diagram that represents the layers of abstraction that were modeled is shown in Figure 28. The lower level shows the detailed architecture of the ROBUS portion of the SPIDER architecture. Error and mitigation models where developed and error propagation analysis was performed on the network layer architecture model. The results of the error propagation analysis were used to build error and mitigation models for the ROBUS component in the Application Layer Architecture Model.

The process for collecting the results at the Network Architecture level and composing higher level architecture models that represent these results at the Application layer is a manual process that relies on separate architecture models with current technology. Consistency between the models is enforced by the user and requires active management of the models to obtain consistency. This exercise shows the potential for integrating models at multiple levels of abstraction to provide for consistency between error models at the various levels. This capability allows for analysis to be performed on a given level of abstraction that is consistent with supporting assumptions and more detailed modeling. As the project progresses we will evaluate the modeling techniques and tools required to automate this process.

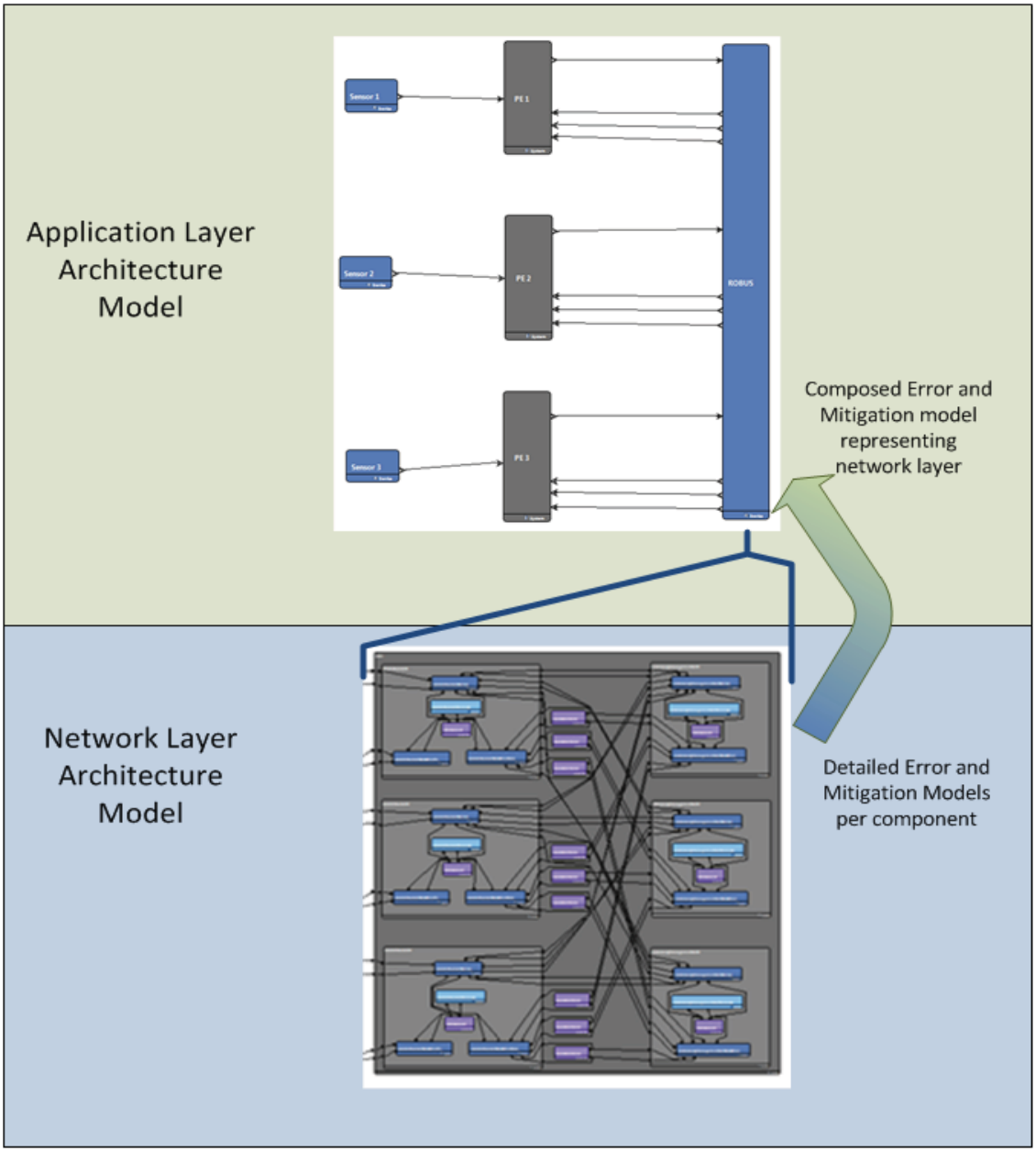

Figure 28 - Layered Error Modeling

A sensor broadcast exchange was modeled at the Application Layer that uses the ROBUS PE Broadcast protocol to exchange sensor values between all of the PEs. The addition of voting and error mitigation mechanisms to the PEs to handle errors sourced to the PEs, the sensors and the ROBUS allowed for the evaluation of error propagation at this level of abstraction. Error propagation analysis was performed to evaluate the effectiveness of the application error mitigation strategy when combined with the error mitigation capabilities of the ROBUS. Figure 29 is a screen shot from the error propagation analysis that shows errors in a single sensor being tolerated at the receiving processing elements.

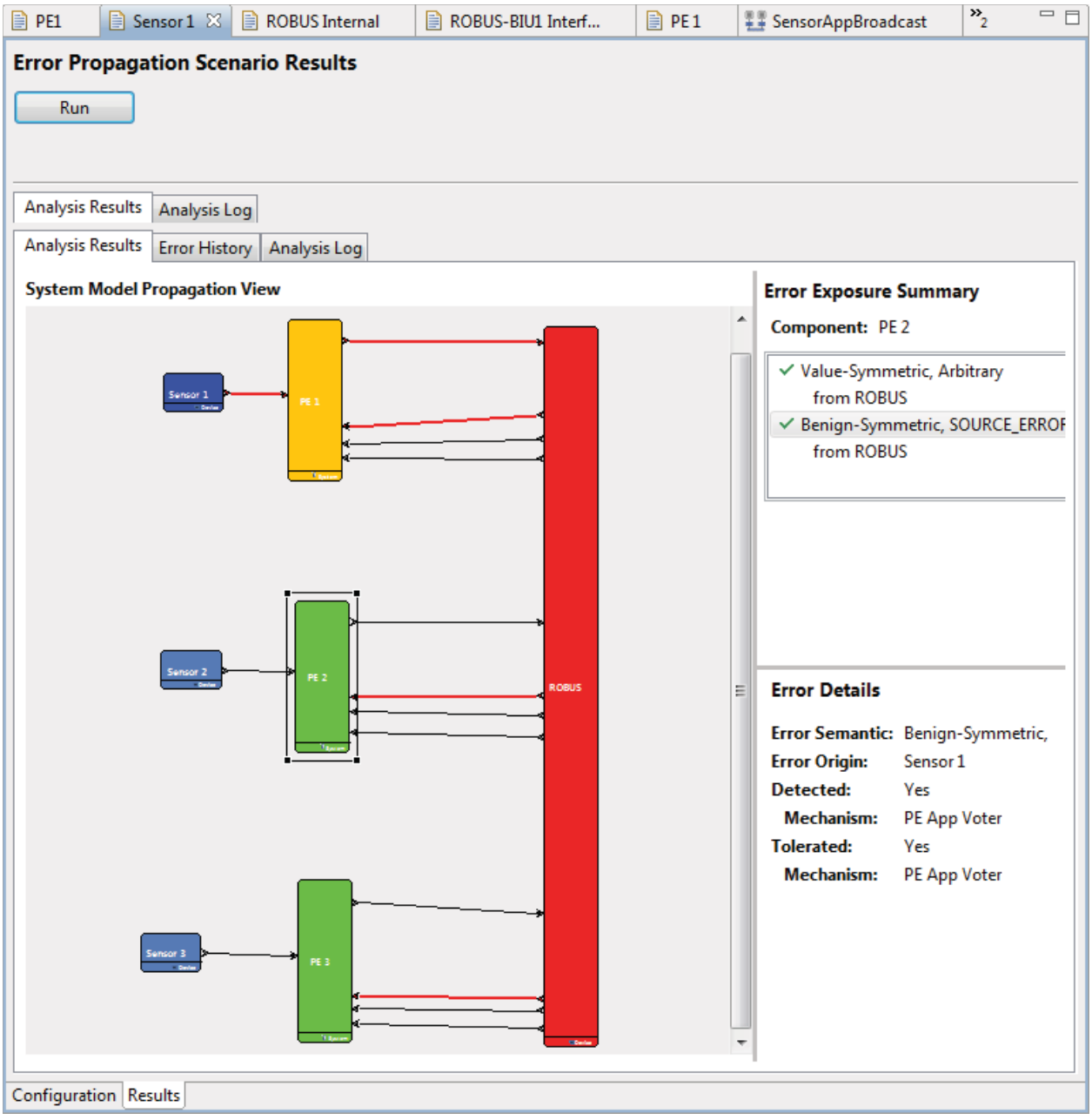

Figure 29 - Application Layer Error Propagation

4. Conclusions

Our work on error modeling and propagation analysis with the AADL and the EDICT tool suite has shown the utility of error modeling and propagation analysis in the design of high reliability systems and revealed several areas where the modeling and analysis techniques could be improved to play an even greater role in system analysis. Our work used the AADL to specify the structure of the computing architecture under analysis and then used EDICT models to augment the structure with models of error semantics, error transformation, error detection and error mitigation.

Error propagation is useful as an exploratory tool for modeling and evaluating the spread of errors from a source in the system. The error propagation behavior results can be seen as the interaction and composition of individual error behaviors of the system components and the connectivity in the system architecture. This bottom up approach is useful for discovering error propagation paths and combinations of error events that may occur. It is complementary to top down specifications of expected error behavior (such as sets of known error states and predefined events that cause state transition) and provides analysts with views of error coverage from a deterministic perspective. The determinism is brought by the nature of the propagation analysis; it is independent of error arrival rates or other probabilistic assumptions but is able to show error effects from all error sources.

Hierarchical modeling and abstraction methods will need to be developed to address the required levels of complexity in modern systems. The overlapping nature of protocol specifications and the resultant data flows in the system architecture require that separate views and representations be developed in order to manage complexity of the models. These models must be composable such that dependencies between the representations are explicit and can be made consistent. Analysis must be used to verify the characteristics at an abstraction level and methods must be developed to trace and make consistent representations at the various levels.

During the course of the work there were several other analysis and modeling features identified that would facilitate the analysis of dependable architectures. These features focus on the ability to combine the behavior specifications for a system with the error specifications of the system. Current modeling and analysis methods are disjoint, often leading to replication of behavior specifications in each of these aspects. This was found to be true in both the AADL modeling capabilities and in the error modeling capabilities of the EDICT tools. The following items highlight the areas of concern that we will address as the project moves forward:

- Enhanced modeling of error mitigator behavior in the presence of multiple failures.
- Enhanced error propagation analysis that considers data flow behavior of the architecture.
- Methods for the representation of protocol behavior and methods to connect the events discovered through error propagation with protocol behavior and overall system state.
- Tighter integration between system behavior specifications and error behavior specifications.

5. References

[1] Torres-Pomales, W., Malekpour, M., and Miner, P., “ROBUS-2: A Fault Tolerant Broadcast Communication System”, NASA/TM-2005-213540, Langley Research Center, Hampton, VA. March 2005.

[2] Torres-Pomales, W., Malekpour, M., and Miner, P., “Design of the Protocol Processor for the ROBUS-2 Communication System”, NASA/TM-2005-213934, Langley Research Center, Hampton, VA. November 2005.

[3] Feiler, P., Glutch, D., Hudak, J., “The Architecture Analysis & Design Language (AADL): An Introduction”, CMU/SEI-2006-TN-011, February 2006.

[4] Feiler, P., Seibel, J., Wrage, L., “What’s New in V2 of the Architecture Analysis & Design Language Standard”, CMU/SEI-2010-SR-008, March 2010.

[5] C.J. Walter, and N. Suri, "The Customizable Fault/Error Model for Dependable Distributed Systems," Journal of Theoretical Computer Science, Volume 290 , Issue 2 (January 2003).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01- 05 - 2011	Contractor Report	

4. TITLE AND SUBTITLE

Error Propagation Analysis in the SAE Architecture Analysis and Design Language (AADL) and the EDICT Tool Framework

5a. CONTRACT NUMBER

NNL10AB32T

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

LaValley, Brian W.; Little, Phillip D.; Walter, Chris J.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

534723.02.02.07.30

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)

NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

NASA/CR-2011-217149

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 62
Availability: NASA CASI (443) 757-5802

13. SUPPLEMENTARY NOTES

This report was prepared by WW Technology Group under subcontract to Honeywell International, Inc., Minneapolis, MN, underNASA contract NNL10AB32T.
Langley Technical Monitor: Paul S. Miner

14. ABSTRACT

This report documents the capabilities of the EDICT tools for error modeling and error propagation analysis when operating with models defined in the Architecture Analysis & Design Language (AADL). We discuss our experience using the EDICT error analysis capabilities on a model of the Scalable Processor-Independent Design for Enhanced Reliability (SPIDER) architecture that uses the Reliable Optical Bus (ROBUS). Based on these experiences we draw some initial conclusions about model based design techniques for error modeling and analysis of highly reliable computing architectures.

15. SUBJECT TERMS

AADL, Error Modeling, Error Propagation Analysis, Model-Based Design, SPIDER

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	35	(443) 757-5802