International Journal of Automation and Computing X(X), X X, X-X
DOI: XXX

An Approach to Modelling and Analysing Reliability of Breeze/ADL-based Software Architecture

Chen Li[1] Hongji Yang[2] Huaxiao Liu[3]
[12]Centre for Creative Computing, Bath Spa University, UK
[3]College of Computer Science and Technology, Jilin University, Jilin, PRC

Abstract: Breeze/ADL is an XML based architecture description language which is used to model software systems at the architecture level. Though Breeze/ADL provides an appropriate basis for architecture modelling, it can neither analyse nor evaluate the architecture reliability. In this paper, we propose a Breeze/ADL based strategy which, by combining generalized stochastic Petri Net (GSPN) and tools for reliability analysis, supports architecture system reliability modelling and evaluation. This work expands the idea in three directions: Firstly, we give a Breeze/ADL reliability model in which we add error attributes to Breeze/ADL error model for capturing architecture error information, and at the same time performing the system error state transition through the Breeze/ADL production. Secondly, we present how to map a Breeze/ADL reliability model to a GSPN model, which in turn can be used for reliability analysis. The other task is to develop a Breeze/ADL reliability analysis modelling tool - EXGSPN (Breeze/ADL reliability analysis modelling tool), and combines PIPE2 (Platform Independent Petri Net Editor 2) to carry out a reliability assessment.

Keywords: Software Architecture, Reliability, Breeze/ADL, GSPN, Breeze Graph Grammar.

1 Introduction

With the rapid development of Internet technology, the size and the complexity of system components and their interconnections continue to grow, which makes software development, integration and evolution more complex. Besides, the related technology of dependability modelling and evaluation for software systems is still lacking, which leads to software products containing a lot of known or unknown defects during the system running time, makes software systems operation facing significant threats and brings new challenges to systems reliability.

System reliability refers to a system that has an ability to execute the required functions without failures for a given time under the specified environment [1−3]. In general, there are two methods for reliability analysis, that is, code based method and component based method, and the software reliability relies on three main technical approaches, faults prevention and elimination, fault prediction [4] and fault tolerance mechanisms [5], and system reliability mainly focuses on the system quality (e.g., online payment requires the reliability of the network server reaching 99.99%). While in the Internet environment, how to analyse and evaluate the software system reliability and select an appropriate security mechanism according to the user's current environment (e.g., time, space and network bandwidth) and conditions (e.g., through the PC, mobile phones or notebooks) is still a hot issue.

Due to the software system becoming complex, the code based analysis is not only inefficient but also lacking accuracy. Thus, the developer transforms their focus to the system component and extracts the core information of the software system in the high-level, which could be used for pre-evaluation to detect the system faults before the system implementation.

As one of the popular model-driven architecture (MDA) [6] development methods, Breeze/ADL [7] has been used to design and develop the software system at high level, i.e., software architecture. Breeze/ADL is an architecture language which adopts XML as the meta-language and has the ability of describing the software architecture. Breeze/ADL specifies the software architecture in a XML format and captures the change during both initial development and subsequent evolution by performing production according to the Breeze Graph Grammars (BGG) [8] productions. However, Breeze/ADL does not support software systems reliability modelling and evaluation, and that is the motivation behind this paper.

In this paper, we borrow some ideas from the Architecture Analysis & Design language (AADL) [9] which has been been widely used in real-time and embedded system development. Combining with the error model, AADL implements the dependability modelling for the software system [10]. Our strategy partially adopts the AADL error model principle and extends the Breeze/ADL by adding error attributes, e.g., state, event and probability of the event, and defining the error production for error transition. In this way, the software systems' reliability model is generated and the system error information can be captured. Then, we present the model transformation rules for mapping the Breeze/ADL reliability model to the generalized stochastic petri net (GSPN) model [11] which is used for modelling the qualitative behaviour of software systems. Thus, the Breeze/ADL reliability model is evaluated by analysing the corresponding GSPN model.

The rest of the paper is organized as follows. Section two introduces background concepts about Breeze/ADL, GSPN and reliability concerns. Section three extends Breeze/ADL

Regular paper
Manuscript received date; revised date
This work is supported by Jilin Province Science Foundation for Youths under Grants No.20150520060JH.
Recommended by Associate Editor xxxx

and gives the Breeze/ADL reliability model. The principle of mapping the error transition of the Breeze/ADL reliability model to GSPN will be illustrated in section four. Section five uses a case study through Breeze and PIPE2 tools to verify our strategy. Section six presents the conclusions of this work.

2 Background

In this section, we give some background concepts about our strategy which include Breeze/ADL, GSPN and reliability concerns.

2.1 Breeze/ADL

Breeze/ADL, as one of our former work, was developed as an architecture language which adopts XML as the meta-language and has the ability of describing the software architecture. Breeze/ADL specifies the software architecture in an XML format and captures the change during both initial development and subsequent evolution by performing production according to the Breeze Graph Grammar productions.

In general, there are three core elements of the software architecture, that is, meta-model layer (style), definition layer and configuration layer. Style layer provides two types of definition, template and style constraints. Like the Class, Breeze/ADL provides the template for component, connector and interface according to the style requirement. Since each architecture has its own style, the developer can design the style constraints in Breeze/ADL. Architecture style depicts the basic information of software architecture and guides the subsequent development. Definition layer implements the elements definition according to the corresponding template. The whole configuration of software architecture is achieved by the configuration layer which defines the instances of the architecture element, like component instance. The above three layers specify the basic static aspect of software architecture. In Breeze/ADL, we add another layer, reconfiguration layer, which gives the user the right to define the production, known as graph rewriting rules in BGG, for dynamic change. In Breeze/ADL, a production is divided into two parts, that is, left hand side (LHS) and right hand side (RHS). The LHS represents the precondition and RHS means the results, i.e., if there is a part of architecture model satisfying the LHS then this part should be replaced by the RHS. Leveraging the production, some operations, like addition and removal, can be applied to the software architecture, and implement the architecture reconfiguration.

In order to support Breeze/ADL, we also developed Breeze, a modelling tool based on Breeze/ADL for software architecture design. Breeze can either take an architecture model file which is specified by Breeze/ADL as inputs or build an architecture through a set of graphic architecture elements. Breeze also supports user defined productions, corresponding to the graph rewriting rules of BGG, to realize the dynamic changes for the architecture reconfiguration. Breeze outputs architecture model in both a graphical and an XML based file. Breeze in addition provides architecture style definition, e.g., Client/Server style, C2 style.

2.2 GSPN

GSPN is used to specify and simulate software systems for quantitative analysis. To be specific, it has been widely used in the reliability [12−13] and performance analysis [14−15]. It not only helps to describe the dynamic structure and behaviour of the system but also consider the time factor during the system running time [16]. Besides, it also supports graphic view to describe a system's behaviour.

A GSPN model has several core elements, e.g., place (drawn by black circle), arc (drawn by directed arrow), transition (drawn by rectangle). A place may stands for a state, position, etc. For example, a place represents a component state in our approach. A transition may lead to place changing. There are two types of transitions in GSPN, timed transition and immediate transition. The timed transitions start after a random time. The immediate transitions are different from timed transitions, it starts right away if the condition satisfied. In our approach, a timed transition follows a Poisson distribution with a failure rate λ and a immediate transition follows a fix failure rate r. The interconnection between the place and transition is through the directed arc.

2.3 Reliability Concerns

According to the definition in GB/T 11457-95, system reliability refers to the system component behaviour not causing the system failure in the fixed time or the system component can provide correct service whenever it gets interference.

In order to improve the system reliability, various approaches have been proposed. Event Tree Analysis [17] uses the boolean logic to analyse a series of sequential events which cause the system failure, and then calculates the probability of these events to evaluate the system reliability. Fault tree analysis (FTA), developed in 1962 at Bell Laboratories by H.A. Watson [18], helps to analyse and evaluate system reliability. FTA, as a top down analysis method, uses binary (i.e., boolean logic) to decompose the system failure to a set of error events. It has been widely used in quantitative and qualitative analysis for software systems. Reliability Block Diagrams (RBD) [19], first proposed as a hardware reliability analysis method, defines the system in two types of structure, i.e., sequential and parallel, and uses a series block diagrams to describe the system function. Cause and effect diagrams [20], i.e., Ishikawa diagrams, were developed by Kaoru Ishikawa in 1943. The Cause and effect diagrams focus on the causes of a specific event and it groups the causes into categories to identify the source. The categories mainly include people, methods, machines, materials, measurements and environment.

Most of the above approaches either only focus on the static structure and behaviour of the software system or not concern the time factor. While GSPN not only specifies the complex software system's dynamic behaviour but also able to describe asynchronous concurrent systems by taking the time as a key factor. Using the GSPN model, we can analyse the undesired system's failure which might be caused by the system's behaviour during the running time. Besides, we can evaluate the system reliability through calculating the probability of an error event.

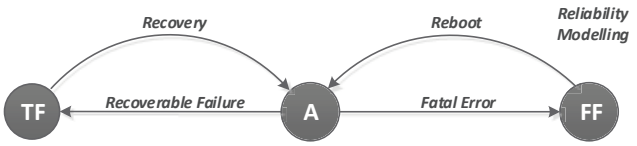

Fig. 1 Component state transformation

3 Breeze/ADL Reliability Model

By introducing the *Error Model* into the architecture specification, the extended software architecture model may support various analysis methods, for example, building a *Fault Tree* for safety analysis or creating the *Markov Model* for reliability calculation.

Error model captures failure of the component and focuses the interaction among components, e.g., invocation, data transmission. The internal failure of each component and external error propagation between components are obtained by leveraging error model at architecture level.

To construct a Breeze/ADL reliability model, this paper extends the meta model of Breeze/ADL and also uses a reliability calculation model through GSPN. In this section, we first give the definitions of error model in Breeze/ADL. Then, we explain how to extend our Breeze/ADL model to build a reliability model.

3.1 Error Model

Definition 3.1 Error Model:= (ErrorState, ErrorEvent, Probability) is a Breeze/ADL error model, where

- *ErrorState.* It represents the state which system's component might be running in. This paper concerns three states which are related to system failure, i.e., *Active* (meaning the component works normally without failure), *Temporary Failure* (describing why the component can not work at this moment because of some temporary failure, e.g., network delay) and *Fatal Failure* (representing the component stop working due to some fatal error, e.g., system bugs). Fig. 1 shows the corresponding component states transformation, *TF* stands for Temporary Failure, *A* represents Active and *FF* means Fatal Failure.

- *ErrorEvent.* The system component may change its state according to the event it received. Events are divided into internal events (i.e., component triggers the event by itself) and external events (i.e., component receives an event which is triggered by other component). Internal events include *Recoverable Failure Event*, *Recovery Event*, *Fatal Event* and *Reboot Event.* The recoverable failure event stands for the temporary failure event. The component uses the internal recovery method to eliminate this failure with a certain probability. For example, when the number of requests reaches to the upper bound of a primary server, the primary server will not respond to the further requests from clients and then a temporary failure event is triggered. This temporary failure can be eliminated by transferring the requests to other backup servers. Recovery event is triggered by the component itself, like the above transferring the requests, and it helps component to back to active state from the temporary failure state. When a fatal failure happens, like deadlock, components can not continue to work and it comes into fatal failure state. This will trigger a fatal event and the component is not able to leverage the internal recovery method to eliminate this problem. Thus, a reboot or manual operation has to be done to make the component back to active state, and this is called reboot event. When a component comes into temporary or fatal failure state, it may propagate error to other components with a certain failure rate. Since this event is triggered by other components, it is called external error event.

- *Probability.* When a temporary or fatal failure happens, the component might propagate the error with a certain probability. In general, there are two types of the probability, i.e., Poisson and Fixed.

According to the above definition of the error model, the error transition within or among the component(s) can be specified due to the error events or error propagation. In order to support the error model, we extend the meta model of Breeze/ADL. To be specific, the above elements are introduced into Breeze/ADL to support error model construction.

Based on the analysis of the error model, a new element - *ErrorModel* is added to the meta model of Breeze/ADL. Two types of error transition, i.e., internal error transition (*InternalTransition*) and external error transition (*ExternalTransition*), are defined in the *ErrorModel.* The following are corresponding meta elements which are added to meta model of Breeze/ADL.

3.1.1 Internal Error Transition

- *Event.* It represents internal events, i.e., Recoverable Failure Event, Recovery Event, Fatal Failure Event and Reboot Event.

- *OrginalState.* It stands for the component initial state which could be Active, Temporary Failure or Fatal Failure.

- *TargetState.* After received an event, component might change from the initial state to the target state which could also be Active, Temporary Failure or Fatal Failure.

- *Lambda.* The probability of internal event follows the Poisson distribution with failure parameter *Lambda.*

3.1.2 External Error Transition

- *Event.* A component may send an external event to the other components when an error propagation happens.

- *TargetComID.* It is the *ID* of target component which received the external event.

- *Fixed.* An external error propagation usually happens immediately with a fixed probability.

Fig. 2, 3 and 4 describe the element tree of node with error model, element tree of error model and extended Breeze.xsd.

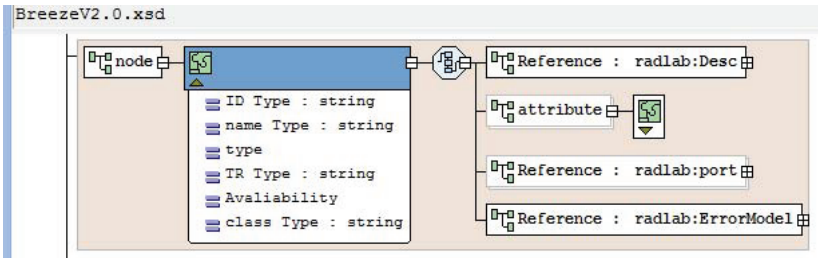

Fig. 2 Element tree of node with error model

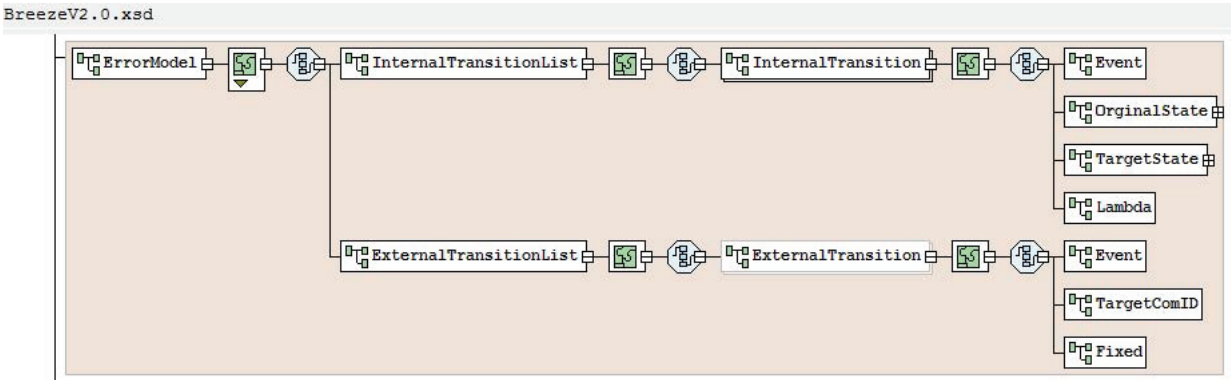

Fig. 3 Element tree of error model

```
<!-- Define the Error Model -->
<element name="ErrorModel">
  <complexType>
    <sequence>
      <element name="InternalTransitionList">
        <complexType>
          <sequence>
            <element name="InternalTransition" minOccurs="4" maxOccurs="4">
              <complexType>
                <sequence>
                  <element name="Event" type="String"/>
                  <element name="OrginalState" type="String" >
                    <simpleType>
                      <restriction base="string">
                        <enumeration value="Active"/>
                        <enumeration value="Temporary Failure"/>
                        <enumeration value="Fatal Failure"/>
                      </restriction>
                    </simpleType>
                  </element>
                  <element name="TargetState" type="String">
                    <simpleType>
                      <restriction base="string">
                        <enumeration value="Active"/>
                        <enumeration value="Temporary Failure"/>
                        <enumeration value="Fatal Failure"/>
                      </restriction>
                    </simpleType>
                  </element>
                  <element name="Lambda" type="float" />
                </sequence>
              </complexType>
            </element>
          </sequence>
        </complexType>
      </element>
      <element name="ExternalTransitionList">
        <complexType>
          <sequence>
            <element name="ExternalTransition" minOccurs="0" maxOccurs="unbounded">
              <complexType>
                <sequence>
                  <element name="Event" type="String"/>
                  <element name="TargetComID" type="String" />
                  <element name="Fixed" type="float" />
                </sequence>
              </complexType>
            </element>
          </sequence>
        </complexType>
      </element>
    </sequence>
    <attribute name="ID" type="string" use="required"/>
    <attribute name="name" type="string" />
  </complexType>
</element>
```

Fig. 4 Definition of error model in Breeze.xsd

3.2 Reliability Modelling

The Breeze/ADL reliability model is divided into software architecture model and error model. Breeze/ADL architecture model specifies the component, connector, connection, etc. of a software system at architecture level, and it provides the constraints of its topology. On the other hand, Breeze/ADL error model captures the reliability information, i.e., error event, error state and error transition. Thus, we combine the architecture model and error model to build an architecture reliability model to implement the reliability analysis at architecture level based on Breeze/ADL.

There are many reliability model analysis methods proposed in recent years, for example, GSPN, Fault Tree, Markov Chain. Due to the GSPN not only describing the dynamic structure and system behaviour but also concerns the time factor during the system running time. In general, GSPN is used to specify and simulate the software system and provide the graphical form to illustrate the system behaviour. GSPN has been widely used in reliability and performance evaluation. Thus, we choose GSPN to model and analyse the reliability, and present rules for mapping the Breeze/ADL reliability model to GSPN model to implement the architecture reliability analysis.

4 Model Mapping

In this section, we use the GSPN as reliability calculation model and build mapping rules between the Breeze/ADL reliability model and GSPN model. Fig. 5 shows the corresponding reliability analysis process.

In Fig. 5, a Breeze/ADL reliability model is constructed

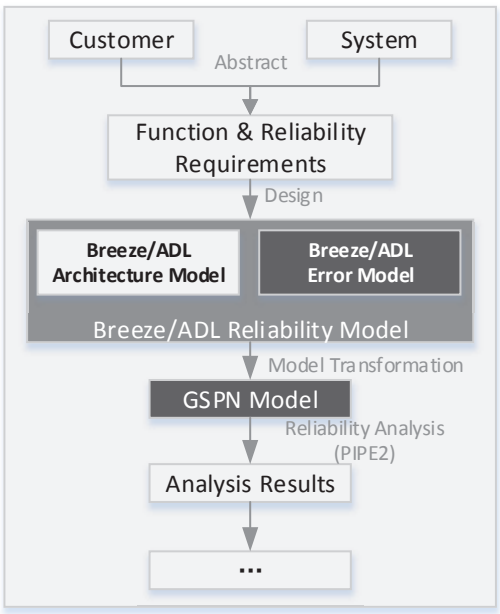

Fig. 5 Reliability analysis with Breeze/ADL

based on functional and nonfunctional (reliability) requirements of customers and system. By using mapping rules, a Breeze/ADL reliability model is transformed to a GSPN model. PIPE2 [21] is leveraged to analyse and evaluate the reliability model and return the results to the customers and system.

Before we present mapping rules, it is necessary to give the definition of GSPN. GSPN is developed based on the SPN and it is widely used in the systems performance and reliability analysis. The following is the corresponding definition.

Definition 3.2 GSPN Model:= (S, T, F, W, M, λ) is a six tuple, where

- S represents the set of the position;
- T means the set of transition, i.e., a transition from one position to another. It is divided into immediate transition and timed transition;
- F stands for directed arc which connects position and transition;
- W represents the weight on the transition;
- M means the initial state;
- λ stands for the timed transition follows the Poisson distribution with a failure rate λ.

By using the above GSPN model, a Breeze/ADL reliability model can be analysed and evaluated. The model transformation processing can be divided into two parts. The first are rules for mapping the error attributes of Breeze/ADL to the corresponding elements of GSPN. Secondly, we give the rules for mapping error production in Breeze/ADL to position transition in GSPN.

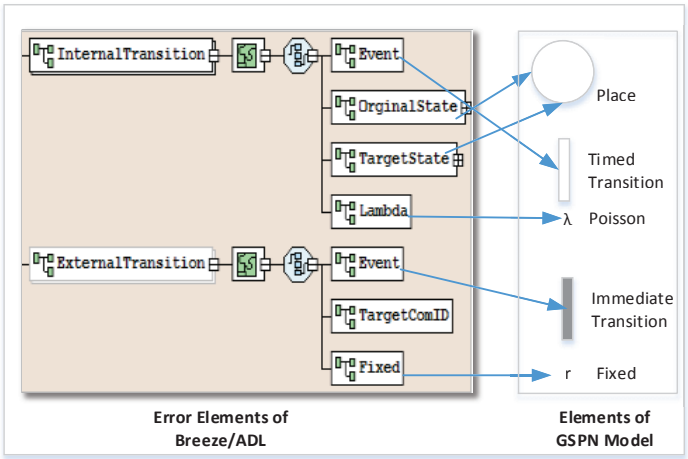

Fig. 6 Element Mapping from Breeze/ADL error model to GSPN

4.1 Breeze/ADL error attributes and GSPN elements

4.1.1 Error state - Position

An error state (ErrorState) of Breeze/ADL is transformed to a position in GSPN.

4.1.2 Error event - Transition

An error event of Breeze/ADL is transformed to a transition in GSPN. Error events include internal events (i.e., Recoverable Failure Event, Recovery event, Fatal Event and Reboot Event) and external event (i.e, External Error Event).

4.1.3 Probability - Weight and λ

In Breeze/ADL reliability model, an internal error propagation follows the Poisson distribution with a failure rate λ. In theory, a component state transition is triggered immediately when it receives an external error propagation. However, the external error propagation needs through connectors or a transmission medium which might lead to deviation. Thus, we assume that the external error propagation follows a fix rate.

Fig. 6 shows the corresponding relationship between the Breeze/ADL error attributes and GSPN elements. The left side of Fig. 6 is error attributes of error model in Breeze/ADL and the right side is core elements of the GSPN.

4.2 Breeze/ADL error production and GSPN transition

When a failure happens, a component will change its current state to error state. If the component can not fix this failure, it will propagate the error to the other components. In order to implement the transformation between the error propagation of Breeze/ADL reliability model to position transition in GSPN model, we leverage the *Production* element of Breeze/ADL. To be specific, pre-condition of a error production describes the initial state, error event and probability of transition and the result of the transition can be captured by the post-condition of the error production. Connector is used as a transition channel to keep the intermediate state.

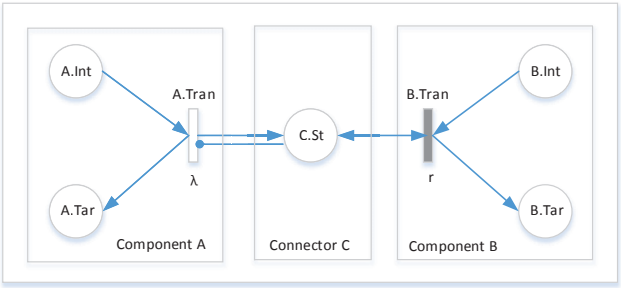

Fig. 7 Transition in GSPN

```
<Production ID="001" name="Active to TemFailure">
  <desc>Error Transiition</desc>
  <Arch ID="Left01" name="beforeTransition" type="ArchGeneral">
    <node ID="N01" name="A" Avaliability="A.Int">
      <ErrorModell>
        <InternalTransitionList>
          <InternalTransition>
            <Event>A.Tran</Event>
            <OrginalState>A.Int</OrginalState>
            <TargetState>A.Tar</TargetState>
            <Lambda>lambda</Lambda>
          </InternalTransition>
        </InternalTransitionList>
        <ExternalTransitionList>
          <ExternalTransition>
            <Event>B.Tran</Event>
            <TargetComID>N03</TargetComID>
            <Fixed>0.8</Fixed>
          </ExternalTransition>
        </ExternalTransitionList>
      </ErrorModell>
    </node>
    <node ID="N02" name="C"></node>
    <node ID="N03" name="B" Avaliability="B.Int"></node>
  </Arch>
  <Arch ID="Right01" name="afterTransition" type="ArchGeneral">
    <node ID="N04" name="A" Avaliability="A.Tar"></node>
    <node ID="N05" name="C" Avaliability="C.St"></node>
    <node ID="N06" name="B" Avaliability="B.Tar"></node>
  </Arch>
</Production>
```

Fig. 8 Error propagation in Breeze/ADL

In Fig. 7, after an error event happens, the state of component A transforms from A.Int to A.Tar following the timed transition with a failure rate λ. Connector propagates the error (C.St) to the component B. After receiving the error, the state of component B transforms from B.Int to B.Tar following the immediate transition with a failure rate r (Bi-direction arc between C.St and B.Tran represents the propagation out and in relationship). Fig. 8 shows the corresponding error propagation in *Production* of Breeze/ADL.

In the above Breeze/ADL definition, the internal error and external error propagation of component A are specified by *InternalTransition* and *ExternalTransition* respectively. Since the component A is the source component, the internal error transition and external error propagation are both initialed by component A. Thus, we define both transitions inside component A. The connector state is changed after the error transition and the corresponding state C.st can be defined in the post-condition of a production. The initial state and the target state of component B is defined in pre-condition and post-condition of the production.

5 Case Study

In this section, we use a Medical Analysis Systems (MAS) as a running example to demonstrate how to build the reliability model in Breeze/ADL and map it to GSPN for reliability analysis. In MAS, there are five Medical Query Terminals (MQT) for five different hospitals, one Government query terminal (Gov) and one Medical Data Center (MDC), that is, one server (MDC) and six clients (MQTs and Gov).

The server may propagate the errors to the clients when the failures happen which could lead to clients failure too. Thus, we need to aware of the error propagation at server side. Due to the function of six medical query terminals being similar, we use MDC and query terminal of hospital A as an example to demonstration the reliability modelling, model transformation and reliability calculation.

5.1 Error elements of MDC and MQT_A

The basic error elements of the MDC is specified as following:

- TemFailure. Due to the Internet delay, the MDC might enter into temporary failure. We assume the temporary failure event follows Poisson distribution with a failure rate 0.006.
- Recovery. When a temporary failure happens to the MDC, the MDC will invoke the recovery method to continue the communication with clients. We assume the event happens following Poisson distribution with a failure rate 0.0005.
- FatalFailure. The disk damage will lead to the fatal failure to the MDC. We assume that this failure happens following Poisson distribution with a failure rate 0.002.
- Restart. By using the backup disk, the fatal failure can be fixed. We assume that this event happens following Poisson distribution with a failure rate 0.0006.
- ExTemFailure. When temporary failures happens to MDC, the MDC will propagate the temporary failures to clients (MQTs). We assume that this external error propagation happens following a fixed rate 0.8.
- ExFatalFailure. When fatal failures happens to MDC, the MDC will propagate the fatal failures to the clients (MQTs). We assume that this external error propagation happens also following a fixed rate 0.8.

The basic error elements of the MQT_A is specified as following:

- TemFailure. Due to the Internet error or received an error propagation from MDC, the MQT_A might enter into temporary failure state. We assume the temporary failure event follows Poisson distribution with a failure rate 0.0002.
- Recovery. If the temporary failure is caused by MDC, the MQT_A will send request to the MDC periodically

Fig. 9 Reliability modelling of MDC in Breeze/ADL

Fig. 10 Reliability modelling of MQT_A in Breeze/ADL

to see if the MDC back to active state. If the temporary failure is triggered by internal error, the MQT_A will invoke the recovery method to fix the problem. We assume the the event happens following Poisson distribution with a failure rate 0.0006.

- FatalFailure. Fatal failure may happen to the MQT_A due to the internal error, and MQT_A could not continue to work. We assume that this failure happens following Poisson distribution with a failure rate 0.0004.

- Restart. Restarting the MQT_A manually may fix the fatal failure. We assume that this event happens following Poisson distribution with a failure rate 0.0007.

Fig. 9 and 10 show the corresponding error elements definition of MDC and MQT_A in Breeze/ADL.

5.2 Error transition of MDC and MQT_A

Fig. 11 shows the error production which defines the internal and external error transition of MDC.

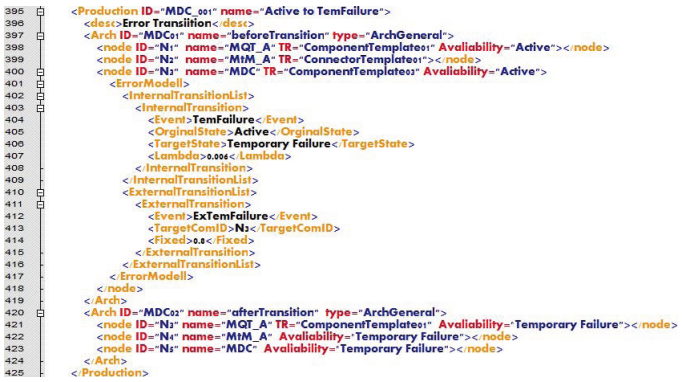

Fig. 11 Modelling the internal and external error propagation with Production in Breeze/ADL

Fig. 12 Modelling the internal error propagation with Production in Breeze/ADL

In Fig. 11, the pre-condition defines the state of MDC changes from the Active to Temporary Failure. The transition follows Poisson distribution with a failure rate 0.006, and MDC propagates the error to MQT_A with a fix failure rate 0.8. Thus, the state of MQT_A changes from the Active to the Temporary Failure. At the same time, the connector - MtM_A will propagate this error and change its state to Temporary Failure too.

We assume that the state of MQT_A changes from Fatal Failure to Active, and Fig. 12 shows the corresponding transition definition in Breeze/ADL.

Due to the client's failure will not influence the server, thus, MQT_A will not propagate the error to the MDC and we only need to define the internal error transition for MQT_A. Similar to MDC, the pre-condition specifies the MQT_A need to change its state from Fatal Failure to Active following Poisson distribution with a failure rate 0.0007. The post-condition defines the state of MQT_A changes to Active.

In order to transform the Breeze/ADL reliability model to GSPN model automatically, we developed a plug-in - EXGSPN to map the error production of Breeze/ADL to GSPN transition processing. Fig. 13 and 14 show the error production for MQT_A in Breeze and the corresponding source code of the GSPN respectively.

In Fig. 14, Breeze transforms the Breeze/ADL reliability model to a file - 1_GSPN_.gspn_model. The right side of the Fig. 14 shows the source code of the Restart transition of MQT_A. *Value* represents the transition name, *rate* means transition rate and *timed* stands for timed transition.

By importing the file - *1_GSPN_.gspn_model* into PIPE2,

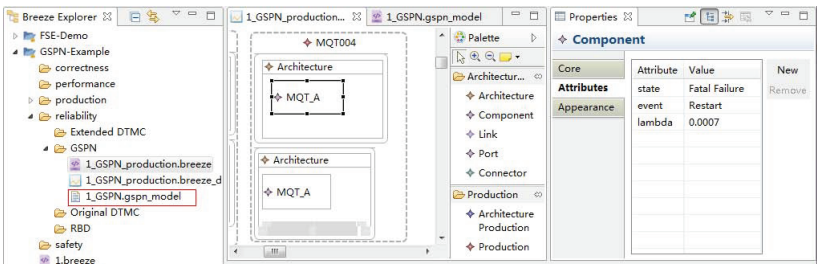

Fig. 13 Defining the error propagation in Breeze Tool

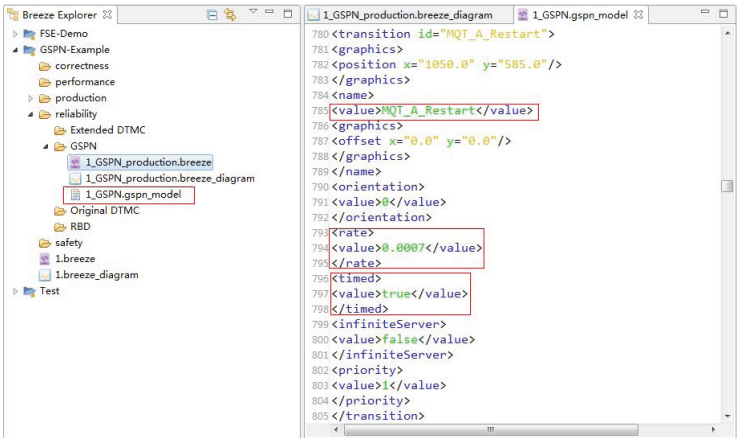

Fig. 14 Source code of GSPN generated by Breeze Tool

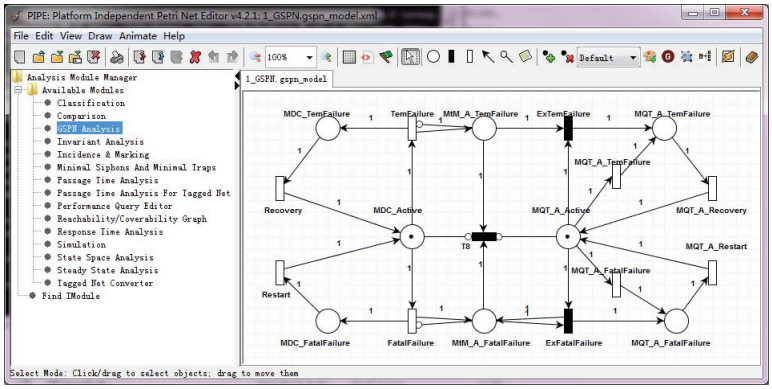

Fig. 15 Importing the transformed GSPN model into PIPE2 Tool

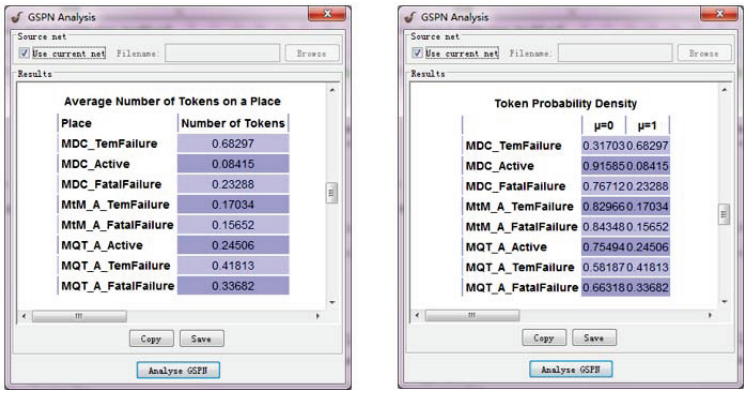

Fig. 16 Reliability analysis by PIPE2 Tool

a GSPN model is obtained. Fig. 15 shows the corresponding GSPN diagram.

Thus, a Breeze/ADL reliability model can be transformed to a GSPN model and a reliability analysis can be implemented by using the PIPE2. Fig. 16 shows the corresponding reliability analysis results by using PIPE2.

According to the above analysis results, the component failure rate, failure distribution, etc. can be obtained. For example, the left side of Fig. 16 shows that failure rate of MDC and MQT_A are 0.916 and 0.755 respectively. In this way, we can evaluate the reliability of the Breeze/ADL reliability model indirectly and adjust the software system, e.g., replacing the low reliability component, or reconfiguring the system.

6 Conclusion

Software architecture reflects the early decision for system development. If the quality issues (e.g. system reliability) can be found at this stage, it will reduce the cost following bugs fixing and software maintaining. However, most of architecture description languages are not able to support the quality modelling and analysis. Thus, we proposed a strategy for modelling and evaluating the reliability for Breeze/ADL based software architecture, which extends the basic Breeze/ADL model by adding the error attribute (i.e., state, error event and probability of event) and giving the error productions for error transition, that is, internal transition and external transition, and present the Breeze/ADL reliability model. Then, we gave the rules for mapping the Breeze/ADL reliability model to the GSPN model which can be analysed by the PIPE2 tool.

According to the analysis result of the GSPN model, the system architecture and its component reliability can be obtained, and we may adjust the architecture structure or behaviour to meet the reliability requirements which improves the software architecture reliability.

References

[1] ISO9126:Information technology - Software product evaluation - Quality characteristics and guidelines for their use. *International Organization for Standardization*, 1992.

[2] ISO8402:Quality management and quality assurance - Vocabulary. *Australian/New Zealand Standards*, 1994.

[3] BS4778:Quality vocabulary. Quality concepts and related definitions. *British Standards Institution*, 1991.

[4] J.Moubray. RCM II: reliability-centered maintenance. *Industrial Press Inc.*, 2001.

[5] A.Avizienis, J.C.Laprie, B.Randell, C.Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[6] Model Driven Architeture (MDA). *OMG*, 2010.

[7] C. Li, L. P. Huang, L. X. Chen, C. Y. Yu. Breeze/ADL: Graph Grammar Support for an XML-Based Software Architecture Description Language. In *Proceedings of the 37th IEEE Computer Software and Applications Conference*, IEEE, Kyoto, Japan, pp. 800–805, 2013.

[8] C. Li, L. P. Huang, L. X. Chen, C. Y. Yu. BGG: A graph grammar approach for software architecture verification and reconfiguration. In *Proceedings of the 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, Taichung, Taiwan, pp. 291–298, 2013.

[9] P.H.Feiler, D.P.Gluch, J.J.Hudak. The architecture analysis & design language (AADL): An introduction, Technical Report, DTIC Document, 2006.

[10] P.H.Feiler, A.Rugina. Dependability Modeling with the Architecture Analysis & Design Language (AADL), Technical Report, DTIC Document, 2007.

[11] G.Chiola, M.A.Marsan, G.Balbo, G.Conte. Generalized stochastic Petri nets: A definition at the net level and its implications. *IEEE Transactions on Software Engineering*, vol. 19, no. 2, pp. 89–107, 1993.

[12] Y.W.Dong, G.R.Wang, F.Zhang, L.Gao. Reliability Analysis and Assessment Tool for AADL Model. *Journal of Software*, vol. 6, pp. 0–14, 2011.

[13] A.E.Rugina, K.Kanoun, M.Kaâniche. A system dependability modeling framework using AADL and GSPNs. *Architecting Dependable Systems IV*, pp. 14–38, 2007.

[14] J.Y.choi, S.A.Reveliotis. A generalized stochastic Petri net model for performance analysis and control of capacitated reentrant lines. *IEEE Transactions on Robotics and Automation*, vol. 19, no. 3, pp. 474–480, 2003.

[15] M.Ajmone Marsan, G.Conte, G.Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 2, pp. 93–122, 1984.

[16] H.H.Ammar, S.M.R.Islam. Time scale decomposition of a class of generalized stochastic Petri net models. *IEEE Transactions on Software Engineering*, vol. 15, no. 6, pp. 809–820, 1989.

[17] C.A.Ericson. Event Tree Analysis. *Hazard Analysis Techniques for System Safety*, pp. 223–234, 2013.

[18] W.S.Lee, D.Grosh, F.A.Tillman, C.H.Lie. Fault Tree Analysis, Methods, and Applications - A Review. *IEEE Transactions on Reliability*, vol. 34, no. 3, pp. 194–203, 1985.

[19] R.Kolar, E.Koh. Reliability Block Diagrams. 2006.

[20] K.Ishikawa. Guide to quality control. *Asian Productivity Organization Tokyo*, vol. 2, 1982.

[21] N.J.Dingle, W.J.Knottenbelt, T.Suto. PIPE2: a tool for the performance evaluation of generalised stochastic Petri Nets. *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 34–39, 2009.

Chen Li received his B. Sc. degree in Computer Science and Technology from University of Science and Technology of China, China, in 2003, the M. Sc. degree in Computer Applications Technology from the University of Shanghai for Science and Technology, China, in 2010, and the Ph. D. degree in Computer Science and Technology from Shanghai Jiao Tong University, China in 2015. Currently, he is a postdoctoral research assistant in School Of Humanities And Cultural Industries at Bath Spa University, UK.

He has published about 28 refereed journal and conference papers. His research interest covers software architecture, software reliability and formal methods.

Dr Li is a member of CCF and IEEE.

E-mail: c.li2@bathspa.ac.uk

ORCID iD:0000-0001-6249-8957

Hongji Yang received his B. Sc. and M. Sc. degrees in computer from the Jilin University, China, in 1982 and 1985, respectively, and the Ph. D. degree in computing from Durham University, UK in 1994. In 1985, he was a faculty member at Jilin University, China, in 1989 at Durham University, UK, in 1993 at De Montfort University, UK and in 2013 at Bath Spa University, UK. Currently, he is a professor in School Of Humanities And Cultural Industries at Bath Spa University, UK.

He has published about 400 refereed journal and conference papers. His research interest covers Software Engineering, Creative Computing, Web and Distributed Computing.

Prof. Yang has become IEEE Computer Society Golden Core Member since 2010, also, he is a member of EPSRC Peer Review College since 2003. He is the Editor in Chief of International Journal of Creative Computing, InderScience.

E-mail: h.yang@bathspa.ac.uk

Huaxiao Liu received his B. Sc. and Ph. D. degrees in Computer Science from Jilin University, China, in 2009 and 2013. Currently, he is a lecturer in College of Computer Science and Technology Jilin University, China.

He has published about 12 refereed journal and conference papers. The central theme of his research is improving software quality, and his recent research concerns the software requirements engineering, software cybernetics and formal methods of software development. More specifically, he develops techniques to verify Aspect-oriented requirements model based on ontology.

Dr Liu is a member of CCF and IEEE.

E-mail:liuhuaxiao@jlu.edu.cn