IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received August 3, 2020, accepted September 2, 2020, date of publication September 7, 2020,
date of current version September 24, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3022016

System Modeling and Fault Tree Analysis Based on AltaRica

ZHEN LI[1,2], **ZHENGQI JIANG**[1,2], **DONGSHENG WANG**[2,3], **(Member, IEEE), AND ZHAOBIN WANG**[1,2]

[1]School of Electronics and Information, Jiangsu University of Science and Technology, Zhenjiang 212003, China
[2]Reliability and System Engineering Open Group, Jiangsu University of Science and Technology, Zhenjiang 212003, China
[3]School of Computer Science and Engineering, Jiangsu University of Science and Technology, Zhenjiang 212003, China

Corresponding author: Zhen Li (justlz@just.edu.cn)

This work was supported in part by the project collaborated with China Aero Poly Technology Establishment under Grant 2014DX078F, in part by the National Natural Science Foundation of China under Grant 61702234 and Grant 51977101, and in part by the Jiangsu University of Science and Technology, Reliability and System Engineering Open Group (JRSOG) Open Fund under Grant 2019003.

ABSTRACT With the increasing scale and complexity of system, it is very necessary to analyze the safety of complex system. Fault tree is an effective method to safety analysis. However, traditional fault tree relies on manual construction and analysis. When fault nodes and systems are complex, the efficiency and correctness of manual analysis can hardly be guaranteed. To the varied understanding of analysts, it is difficult to ensure the consistency of failure mode and system architecture due to the different understanding from safety analysts and system designers. The same node needs fault analysis again in different systems, which has poor reusability and low efficiency. AltaRica is a fault-oriented Safety Modeling Language. It takes the guard transformation system(GTS) as its core, describes nodes and faults with a style of reusable object-oriented language, and describes information of interaction between nodes and systems through interface connections between nodes and nested systems. Therefore, this paper proposed an automatic system modeling and fault analysis method and its detailed computer algorithm on single class node, multiple nodes and nodes with subsystems based on AltaRica. Finally we developed a software prototype and carry out the automatic modeling and fault analysis in a detailed example. The results showed that the proposed method, algorithm and software prototype can realize automatic graphical modeling of the system on AltaRica, the automatic fault analysis is correct and efficient, has reusability of modeling and fault analysis, and greatly improve the accuracy, objectivity and efficiency of fault modeling and analysis.

INDEX TERMS Safety, reliability engineering, system analysis and design, AltaRica, fault tree analysis.

I. INTRODUCTION

With the rapid development of science and technology, the integration of system is getting very high, the scale is getting larger, and the function is getting more complex. A small fault may cause the whole system failed, leading to serious safety accidents. Therefore, more and more attentions have been paid on the system safety.

In safety-critical systems, researchers widely used system safety modeling and analysis technology [1]. Traditional system safety analysis techniques mainly include Markov process analysis, FMEA(failure mode effect analysis) [2], and FTA (fault tree analysis) [3]. These analysis techniques can provide efficient algorithms and tools. However, the models designed using these forms are far from the specifications of the system. Therefore, they are difficult, expensive and inefficient to design and maintain throughout the life cycle of the system. Traditional fault analysis techniques rely too much on the analysts, and the understanding of system structure by analysts plays a great role in the construction of fault systems, which is highly subjective. Nowadays, safety-critical systems and equipment are more and more complex, so it is difficult to achieve complete, continuous and error-free analysis process and results by brains and hands of analysts. Once a change in the system, it often forces the reconstruction of the fault model, resulting in a large waste of human and time resource.

At the same time, with the continuous development of model driven and formal technology, researchers combine safety analysis model with system design model, and many model-based safety analysis (MBSA) and evaluation

The associate editor coordinating the review of this manuscript and approving it for publication was Zhaojun Li.

methods emerged with a characters of related modeling language and automation [4]. MBSA techniques include Failure Propagation and Transformation Notation (FPTN) [5], Hierarchically Performed Hazard Origin and Propagation Studies (HiPHOPS) [6] and AltaRica language [7].

Jean-Yves Choleya and etc derived from previous MBSE and MBSA integration studies, performed mainly on M2M and M2T transformations, and appropriate SysML metamodeling, proposed CPS safety analysis methodology with an aeronautic industrial case study [8].

Model Checking language and tool such as NuSMV, Spin and AADL are integrated to MBSA due to their powerful abilities of searching the model space. Lu Chen, Jian Jiao and etc put forward a failure analysis method using NuSMV to manually transform the counterexample into FMEA or FTA result [9]. There are some limitations of model checking in MBSA, one as only by dynamic operation, partial information of FT or FM can get, because when counterexample occurs, the operation will stop the analysis which is ongoing. The other limitation is there still has no mature tool to support all this process automatically which is mainly divided into relatively independent parts of system and safety modeling, translating into model checking language, model checking, safety analysis and FT generation based on counterexample manually.

Petri Nets (PNs) are a formal graphical and mathematical modelling tool which is appropriate for specifying and analyzing the behavior of complex, distributed and concurrent systems. Many papers related to PNs and its extensions are summarized in [10] including Bayesian network and extensions of PNs. PNs originated from a mathematic model and method to analyze system with characteristics of resource production and consuming, condition trigger, concurrence and confliction, and etc. It is not a specific model to deal with fault and safety, and when be used and developed with computer science PNs have some extensions as Stochastic PNs, Time PNs, Colored PNs, Temporal PNs, PNs related to model checking language, and etc. Now PNs is not an independent modeling method, it is tightly combined with computer science and the core is formal automata. So the PNs is suitable to model system behavior and not specific in safety, it can be used in safety analysis if the system is safety-critical.

Altarica is a formal language developed by the computer science laboratory of Bordeaux jointly with industries partners. The purpose of creating AltaRica language is to overcome the shortcomings of traditional formal methods which deviate from the research system, such as fault tree, Markov chain, and Petri net. AltaRica language is a formal, object-oriented modeling language. It can describe the functional behavior of the system under normal conditions and the failure behavior of the system.

Later with the cooperation of academia and industry, AltaRica's capability of system safety assessment has been well strengthened. Using AltaRica model can truly reflect the structure of the system, the operating mechanism of the system, and has a good reusability on its style of object-oriented language. Many companies have used AltaRica for safety assessment analysis in their important projects, including Alston Railway, Total, Schneider Electric, and France Telecom and so on. AltaRica has became the standard of model-based safety assessment in European industry.

The structure of the AltaRica model has a strict grammatical definition, which can describe the system structure and generates the safety model according to the fault propagation logic of the system [11]. Based on this, the FTA method can be combined with AltaRica to implement safety analysis and generate fault tree.

As the inventors of AltaRica, Batteux M, Prosvirnova T and Rauzy A illustrated the whys and wherefores of the fixpoint assertion mechanism introduced in AltaRica 3.0 to perform changes of states in literature [12]. Based on the key GTS(Guarded Transition System) of AltaRica, it can increase expressive power and in particular the ability to handle looped systems is obtained without any significant overhead for data-flow models by using the fixpoint mechanism introduced in AltaRica 3.0.

In system modeling and analysis using AltaRica, there are some researches on it. Issad M, Kloul L and Rauzy A proposed a scene-oriented modeling approach which relies on semi-formal representation to describe the scene and relies on the formal execution model described in the AltaRica 3.0 modeling language [13]. In literature [14], a method was described that extends the AltaRica model developed for safety analysis with time information and provides a tool to interpret the correctness of time safety conditions. Michael Lipacewski, Frank Ortmeier, and et al compared the SAML and AltaRica and pointed out that AltaRica is more convenient to large system modeling and easier to be reused due to the object-oriented style [15]. Brunel *et al.* [16] introduced the principles of the AADL and AltaRica languages and the connections between them, proposed a conversion procedure from AADL to AltaRica, and applied its prototype to the simplified flight control system of the drone. Literature [17] proposed the use of AltaRica language to build electrical and electronic system models protected by first-order and second-order safety mechanisms, and clarified that the model is helpful to analyze the behavior of the system and determine the validity areas of simpler models. Duo and Li [18] proposed a practical safety modeling methodology based on Altarica, which contains three phases like information collection, model construction and model V&V, with an example of a hydraulic system. Long G and Liang A proposed a fault modeling and analysis method for complex systems. Taking the heating and cooling backup system and the reconstruction system as examples, the accuracy of the fault logic conversion process is verified in [19].

In fault analysis and fault tree generation from AltaRica, there are some researches on it. Humbert S, Seguin C, Castel C, and et al took a part of the control system of a helicopter turboshaft engine as the research object, extracted requirement from the system fault propagation model by using AltaRica for modeling and analysis [20]. Batteux M, Prosvirnova T and Rauzy A, et al mentioned the transform GTS model into fault tree, which is, to transform state/transition model into a set of Boolean formulas, and then verifies the safety of AltaRica system with relevant safety assessment tools [21]. In literature [22], Prosvirnova T and Rauzy A briefly introduced fault tree generation from independent GTS and simply illustrated the process of getting one single fault tree from connected nodes without mentioning the generation from subsystems and nodes. XiaoXun Li, ShaoJun Li extracted failure logic relations from Altarica component models and system models to generate fault trees, and discussed the fault tree generation method from multiple nodes in AltaRica [23]. They showed this method in example of graphical modeling to connect Altarica nodes in figures drawn in hand without details of computer algorithm to implement it [24].

OCAS [25] is the MBSA tool that was accepted by regulatory agencies as a basis for certification of safety-critical systems. Bozzano *et al.* [26] considered the limitations of OCAS as no ability to perform an exhaustive space examination and limitations in in presence of industrial-sized systems hindering the generation of important artifacts such as Fault Trees.

Simfia is a software package that, based on the knowledge and functional analysis of an equipment, product or system, can be used to analyse and simulate its overall behaviour and automate R.A.M.S. studies [27]. Fukai Zhang used Simfia to make the AltaRica model of HUDS visually to achieve assessing formal safety [28]. In [29], Event—B language is used to model application layer to check the integrity of operations modes and AltaRica is used to model dySfunction of system to solve the problem of dynamic failuret The efficiency and practice of the method are illustrated by analyzing safety of auto pilot system thmugh Rodin t()01 which is used for analyzing operational modes of application and Simfia tool which is used for safety analysis.

In those existed literatures, we have found AltaRica has been successfully used in system and fault modeling and analysis [13]–[19], and most of those literatures only proposed the principle and framework to generate fault tree from assertions in single or multiple class [20]–[24]. So there are relatively absence in method to automatically generate and programming realize the safety analysis as FT from complex nested systems including interaction between nodes and subsystems. Some papers used OCAS [25], [26] and Simfia [28], [29] to implement MBSA, but there are some limitations and no details of how the computer algorithms implemented and no expansions to nodes and subsystems. The details of realization of automatic computer algorithm and tool developing are absent and not open enough.

This paper studied the grammar and semantics of AltaRica language, proposed NAltaSys(Nested AltaRica System) to well support complex system modeling, detailed illustrated the automatic algorithms including analyzing assertion and link information in AltaRica models and fault tree generation from single node, multiple nodes and subsystems, combined visual modeling techniques with fault logic description ability of AltaRica language to improve the safety description and analysis ability of the model, and developed an automated system fault modeling and analysis tool named SSMA based on AltaRica to system modeling and automatically generate system fault tree. The fault tree is convenient to modify and maintain according to the change of class in AltaRica fault model. This paper realized the synchronous work of fault modeling, analysis, modification and maintenance based on AltaRica, and greatly improved the efficiency of fault modeling and analysis of safety-critical systems.

The contents of this paper are arranged as following. Section I is an introduction of system modeling and analysis, and the AltaRica with researches related to it. Section II makes a detailed description of AltaRica language includes all elements and GTS mechanism in it. Section III introduces the NAltaSys(Nested AltaRica System) proposed by this paper which will support the system modeling, algorithm design and code programing to visual software development. Section IV introduces the algorithm of fault tree generation from single class, multiple nodes and nodes with subsystem, and then makes examples for each situation. In Section V, a detailed example is put forward to prove the correctness of the algorithm proposed and the tool developed in this paper. Finally, there comes to the conclusion.

II. AltaRica MODELING LANGUAGE

AltaRica is a high-level modeling language for safety analysis. It is based on GTS and its mathematical model [12]. The basic component of the AltaRica model is node which is an instance of Class. The class and its important parts are described as:

1) Class: In order to use a box with certain given characteristics (encapsulating a GTS), you need to declare a class, the type of the box. The AltaRica model is a series of declarations of classes. Classes can be nested into other instances of other classes to achieve hierarchical description of the research system.

2) Domain: Domain has domain name and its states which normally represent the different state of class.

3) Boolean: The logical type that represents the state of variables in class.

4) Reset: Represents a reset value of variables in class.

5) Variable: Two types of variables called domain variable and class variable are in the class. Domain variable represents the external state of class not limited to Boolean value. Class variable represent the state of internal variable in class.

6) Flow: Flow is like interface variables, are divided into input and output flow, which are used to describe the state of interface variables between class nodes or class node with subsystem.

7) Trans: Trans is a triple $< e, \mathrm{G}, \mathrm{P} >$, which is defined as: where e is the event marking the transition, G is a Boolean condition about the state and flow variables, called the Guard of transformation, P is the action performed by the new state

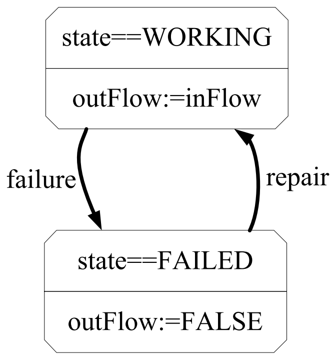


FIGURE 1. State transition diagram.

```
domain PumpState { WORKING, FAILED }
class Pump
PumpState state (init = WORKING);
Boolean inFlow, outFlow (reset = false);
event failure, repair;
transition
failure: state==WORKING -> state:= FAILED;
repair: state==false -> state := WORKING;
assertion
outFlow := if state==WORKING then inflow else false;
end
```

FIGURE 2. AltaRica Pump model.

calculation of the state variables. When the Guard is satisfied, trans $e : G \rightarrow P$ will be triggered.

8) Assertion: A set of constraints on outflows values after transfer functions occurred.

The state transition diagram of a pump is shown in FIGURE 1, and its AltaRica model is shown in FIGURE 2.

The model first needs to declare the domain (“PumpState”), namely two symbol constants ‘WORKING’ and ‘FAILED’, of course we can define more than two constants as you need. Then, declare a class (the type of box), in that case the GTS is called “Pump”.

The GTS includes:

(1) a state variable “state”, which takes its value in the definition domain “PumpState” and its initial state is “WORKING”
(2) two Boolean flow variables “inFlow” and “outFlow”, whose default values are false
(3) two events “failure” and “repair”, each event corresponds to a transition and an assertion that constrains the value of the variable.
(4) When one event occurs, the state will change in transition and result in the change of outflow through related assertion
(5) After change of outflow, it will pass out if connects to other node or subsystem. When it is false, that means it is a fault will pass out and the reason is the assertion related to this outflow.

All state variables are initialized once. Their initial values are given by the attribute “init”. This attribute describes the characteristics of state variables. After each conversion is triggered, the flow variables are updated. Its value is determined by the assertion or by the default value given by the attribute “reset”. This attribute describes the characteristics of flow variables. If the box “Pump” is used alone, “inFlow” will be reset to the default value after each conversion trigger. If it is connected to another box through the “inFlow” and the pump is working properly, the assertion will be used to pass the value. The transition marked by event “failure” triggers only when the state variable “state” takes the value of “WORKING”. After the conversion triggers, the “state” value is first set to “FAILED”, and then the value of the flow variable is updated.

III. NESTED AltaRica SYSTEM MODELING

Complex system has many nodes and subsystems, the fault information will pass into or come from subsystem. In complex nested system, faults reason hides in the single node or subsystem and it is very hard to uncover the reasons of top system faults manually that originate from single node and subsystem passing by interface. So it is significantly meaningful to propose a nested system model which can be transacted by computer, and based on that the algorithm of automated fault analysis of nested system with related tool can be developed.

This paper put forward a nested system model based on AltaRica to well support system and fault modeling with analysis.

This model called NAltaSys(Nested AltaRica System) is formally described as:

$$< InNode, Class, Node, SubSys, Connection, OutNode >,$$

where:

(1) InNode: InNode is a virtual entry of all input, that means all input passed to nodes through InNode.
(2) Class: The definition of Class is the same as in the section II. It have domain, variable, flow, trans, assert and etc.
(3) Node: Node is an instance of Class which inherits all properties in Class defined in the section II.
(4) SubSys: SubSys has a recursive definition as < InNode, Class, Node, SubSys, Connection, OutNode >. It is a part of whole system which has some nodes or even subsys in it, and connections among nodes with subsys.
(5) Connection: Connection has the interface information between nodes or node to SubSys. The fault will pass or backtrack through connections and obey the mechanism defined in flow and assertion in Class.
(6) OutNode: OutNode is a virtual exit of all output, which means all output from nodes should pass through OutNode.

The NAltaSys model is described as in Fig3.

This paper models nested system based on AltaRica and implemented automatic safety analysis (Fault Tree Analysis).

The Fault Tree Analysis can be done in three situations:

(1) Single Class Node

The basic Fault Tree Analysis is implemented in single class node. This paper used the information of assert in class,

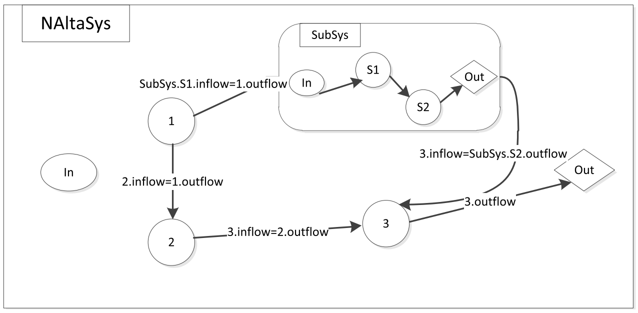


FIGURE 3. Nested AltaRica System.

```
class Class1
Boolean A1,B1// variable
Boolean Y1,X1// inflow
Boolean U1,V1;//outflow
event eA1,eA2,eB1,eB2;
transition
eA1: A1==true ->A1:=false;
eA2: A1==false -> A1 := true;
eB1: B1==true ->B1:=false;
eB2: B1==false -> B1 := true;
assertion
U1=if ((A1==false)and(B1==false))or(Y1==false)
then false else true;
V1 = if (((A1=false) or (X1=false))) then false else true;
end
```

FIGURE 4. Details of Class1.

then scan, cut it and calculate the position of FTA node with algorithm detailed described in Part A of Section IV.

(2) Multiple Class Nodes

This paper firstly gets the nodes link to OutNode, then gets the connected interface information bind in links to the OutNode, and trace back from the inflows to outflows with assertions bind with them, then recursively trace back to generate the whole fault tree of multiple class nodes.

(3) Nodes with SubSystem

When implement FTA in nodes with subsystem connected, this paper firstly gets the nodes link between node and subsystem, then analysis connected interface information between node and subsystem, and trace back the inflow and outflow to get the fault information based on the algorithm in (1) and (2) to generate the whole fault tree of nodes with subsystem.

IV. FAULT TREE GENERATION ALGORITHM

We have known the outflow will pass out if connects to other node or subsystem. The fault reason is in description of assertion related to outflow.

In a class, one fault tree to one outflow can be got through the analysis of the related assertion. The assertion will transacted to construct fault tree. The assertion will be scanned and cut into fault node automatically by algorithm proposed by this paper. And then this algorithm will also get the level that is the position of each fault node. By having got fault tree node and its position, we will get the whole fault tree of outflow from GTS in AltaRica.

When an outflow connected to other node or subsystem, it will pass out the related fault. The information in the link of node to node or node to subsystem is as an interface. Because we have known the link information of each outflow, so we can known how the fault pass out by analysis of the interface. If we can correctly get the path that outflow which bind the fault, we will get the whole path that the fault pass out in the whole system.

So the most important work is to design the computer algorithm and make the computer automatically implement the system modeling, assertion analysis of single node, analysis of link information, automated fault tree analysis and generation between nodes and subsystems.

Using AltaRica to define a class, we can make the instance of it repeatedly. We can just connect those instances in anywhere in the system, we will automatically get the analysis result of whole system because we have known the fault result of assertion in single node and how it connect by system modeling and the analysis of the link information between node and subsystem using the algorithm proposed in this paper. That is we only need to define a class in which the variable, inflow, outflow and assertion have been described. After that, we can use those class to define instance repeatedly and to construct the whole system. And we need not do any additional system analysis but can automatically get the fault tree analysis of whole system, for we may get the fault tree in each node and the link information of connected interface by the algorithm proposed in this paper. Comparing to traditional fault tree analysis, it is no need to implement fault tree analysis in node, subsystem and whole system. Once the system is modeled, it will construct the whole system fault tree automatically without any additional analysis as in traditional fault tree analysis.

This paper designed the fault tree generation algorithms from AltaRica system models include single class node, multiple nodes, and nodes with subsystems.

A. SINGLE CLASS NODE

Here we make Class1 as an example of fault tree generation from single class node which is described by AltaRica in Figure 4.

We get the fault information from assertion "U1: if ((A1==false) and (B1==false)) or (Y1==false)

then false else true" where:

U1: an output flow

A1, B1: class variable

Y1: inflow

We find that if the expression

"((A1==false) and (B1==false)) or (Y1==false)" is satisfied, the U will come into false that means it causes the fault. So we can get the reasons of the fault by the expression of assertion and can also get the position and preposition of nodes to be drawn in fault tree.

There are two main steps in Assertion Analysis Algorithm of Single Class Node.

```
// MakeAssertionToAssertionArray()
int length=m_strAssertion.GetLength();
char *pAssertion=new char[length+1];
strcpy(pAssertion,(LPCTSTR)m_strAssertion);
//scan the assertion
int level;//record level of embrace
CString strTemp;// record valid Character String
While(i<=length)
{
  if (pAssertion[i]=='(')    level++;//embrace level
  else if (pAssertion[i]==')')   level--;//embrace level
 //if valid character, insert it to strTemp
 if(isalpha(pAssertion[i])||isdigit(pAssertion[i])||(pAssertion[i]=='_'
)||(pAssertion[i]=='=')||(pAssertion[i]=='.'))
strTemp.Insert(strTemp.GetLength(),pAssertion[i]);}
//when '(', ')', ' ', '\0', insert as an AssertionNode
else if((pAssertion[i]==')')||(pAssertion[i]=='(')
||(pAssertion[i]==' ')||(pAssertion[i]=='\0'))
{
  if (!strTemp.IsEmpty())
  {  //New Node
     AssertionNode node;
     node.strElement=strTemp;
     node.level=level;
     node.isCut=FALSE;
     //Adjust Embrace Level
     if (pAssertion[i]==')')
          node.level=level+1;
     else if (pAssertion[i]=='(')
          node.level=level-1;
     AssertionNodeArray.Add(node);//Add Node
  }
}
i++;}
```

FIGURE 5. Assertion Analysis Algorithm-Step 1.

TABLE 1. Result of assertion scanned, cut with levels.

String Cut	Level
A1==false	2
and	1
B1==false	2
or	0
Y1==false	1

The first step is to record the valid character string and calculate its level which is scanned and cut from Assertion which includes:

(1) Scan, cut and record valid character string of assertion in single class node
(2) Record the embrace level of each character string
(3) Add to AssertionNodeArray realized by function MakeAssertionToAssertionArray() in class CAltaRica-Analyser.

The details of the first step in algorithm is shown in Fig5.

Here are the result of assertion scanned, cut with levels of each valid string in the Table 1.

The second step is to recursively generate the prepositions of Expression and Logic Operation nodes will be drawn in Fault Tree which includes:

(1) Scan the AssertionNodeArray, get the position of first root node and set its preposition to -1

```
// MakeAssertionArrayToFaultTree
int level,minLevelPos;
CString strNode; preposition=-1;
//Get the position whose level is minimum
minLevelPos=GetMinLevelPos(NodeArray);
//cut into left and right part, set preposition, recursively operation
if (minLevelPos>=0)
{//get and set current node's preposition, insert to AssertionNodeTreeArray
AssertionNode node=NodeArray[minLevelPos];
node.prePosition=preposition;
AssertionNodeTreeArray.Add(node);
//label the position ready to use recursive algorithm
int posInsert=AssertionNodeTreeArray.GetSize()-1;}
if((minLevelPos==1)&&(!IsLogicalOperation(NodeArray[0].strEle
ment)))
//if only one expression in left part, insert to AssertionNodeTreeArray
{node=NodeArray[0];
 node.prePosition=posInsert;
 AssertionNodeTreeArray.Add(node);}
else//recursive algorithm in left part
{ Carray<AssertionNode,AssertionNode> AssertionNodeTreeArrayLeft;
  AssertionNodeTreeArrayLeft.SetSize(0,-1);
  //Generate NodeArray of left part
  for (i=0;i<minLevelPos;i++)
AssertionNodeTreeArrayLeft.Add(NodeArray[i]);
// recursive algorithm run on left part
MakeAssertionArrayToFaultTree(AssertionNodeTreeArrayLeft,posInsert);}
//if only one expression in right part, insert to AssertionNodeTreeArray
if((minLevelPos==(size-
2))&&(!IsLogicalOperation(NodeArray[size-1].strElement)))
{node=NodeArray[size-1];
 node.prePosition=posInsert;
 AssertionNodeTreeArray.Add(node);}
else{
Carray <AssertionNode,AssertionNode> AssertionNodeTreeArrayRight;
AssertionNodeTreeArrayRight.SetSize(0,-1);
int sizeRight=NodeArray.GetSize();
for (i=(minLevelPos+1);i<sizeRight;i++)
AssertionNodeTreeArrayRight.Add(AssertionNodeArray[i]);
// recursive algorithm run on right part
MakeAssertionArrayToFaultTree(AssertionNodeTreeArrayRig
ht,posInsert);}
}
```

FIGURE 6. Assertion Analysis Algorithm-Step 2.

(2) Cut the AssertionNodeArray into left part and right part with the center position of the first root node

(3) If only one expression or logic operation node in the left, get the position of this node, and set the position of the first root node to its preposition, else recursively run this algorithm on the left part as step (1)

(4) If only one expression node or logic operation node in the right, get the position of this node, and set the position of the first root node to its preposition, else recursively run this algorithm on the right part as step (1)

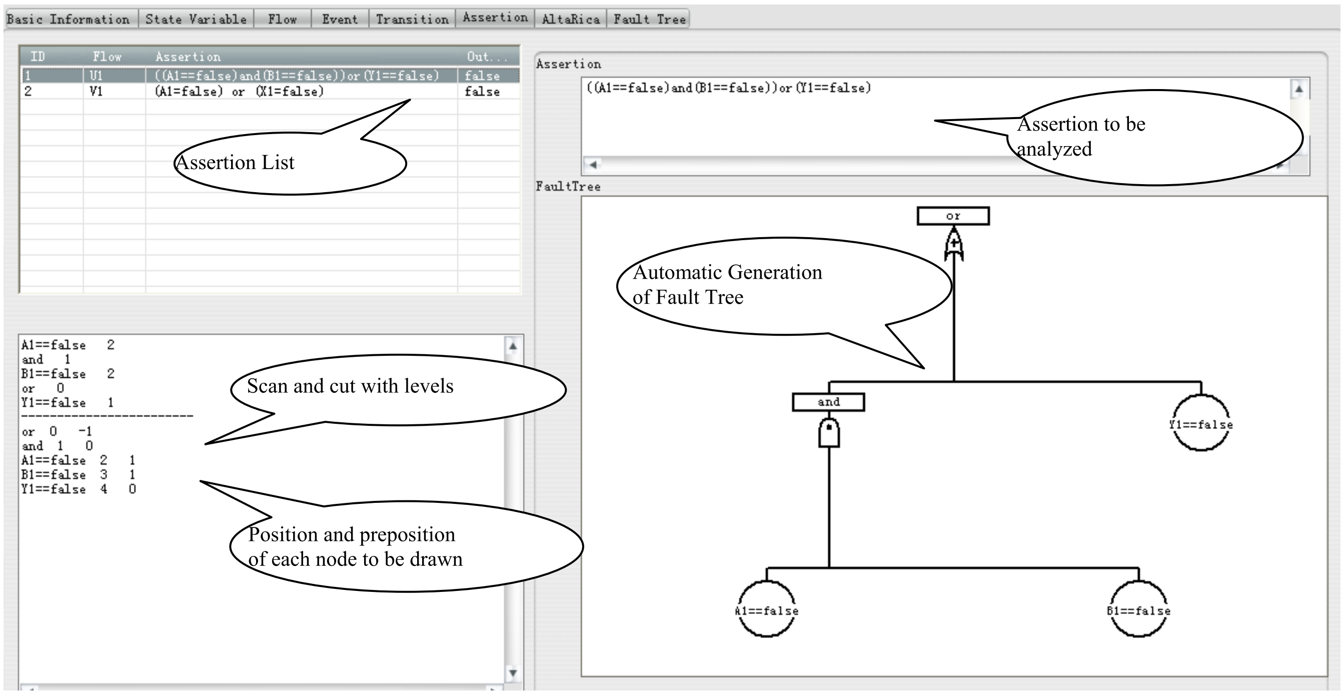
The details of the second step in algorithm is shown in Fig 6.

Here are the example results of Assertion U1 scanned, cut and recursively run in the Table 2.

Then this paper will draw the fault tree by the position and preposition of each node in AssertionNodeArray as in Fig 7.

Here are Class2 and Class3 that will be related to multiple nodes and nodes to subsystem.

Class2 as detailed in Fig 8 has an inner variable C2, an inflow V2, two outflows W2 and X2, and two assertions



ID	Flow	Assertion	Out...
1	U1	((A1==false)and(B1==false))or(Y1==false)	false
2	V1	(A1=false) or (X1=false)	false

```
((A1==false)and(B1==false))or(Y1==false)
```

```
A1==false   2
and   1
B1==false   2
or   0
Y1==false   1
-------------------------
or  0   -1
and  1   0
A1==false  2   1
B1==false  3   1
Y1==false  4   0
```

FIGURE 7. Final Result of Automatic Fault Tree Generation and Drawing of Class1.

TABLE 2. Result of position and preposition of assertion node.

String Cut	position	preposition
or	0	-1
and	1	0
A1==false	2	1
B1==false	3	1
Y1==false	4	0

```
class Class2
Boolean C2// variable
Boolean V2// inflow
Boolean W2,X2;//outflow
……
……
……
assertion
W2=if ((A2==false)and(B2==false))or(Y2==false)
then false else true;
X2 = if ((C2==false)) then false else true;
end
```

FIGURE 8. Details of Class2.

of W2 and X2. The final Result of Automatic Fault Tree Generation and Drawing of assertion related to outflow W2 in Class2 is shown in Fig 10.

Class3 as detailed in Fig 9 has an inner variable D3, two inflows U3 and W3, two outflows Y3 and Z3, and two assertions of Y3 and Z3. The final Result of Automatic Fault Tree Generation and Drawing of assertion related to Z3 in Class3 is shown in Fig 11.

```
class Class3
Boolean D3// variable
Boolean U3,W3// inflow
Boolean Y3,Z3;//outflow
……
……
……
assertion
Y3=if (D3==false) then false else true;
Z3= if ((U3==false)and (W3==false)) then false else true;
end
```

FIGURE 9. Details of Class3.

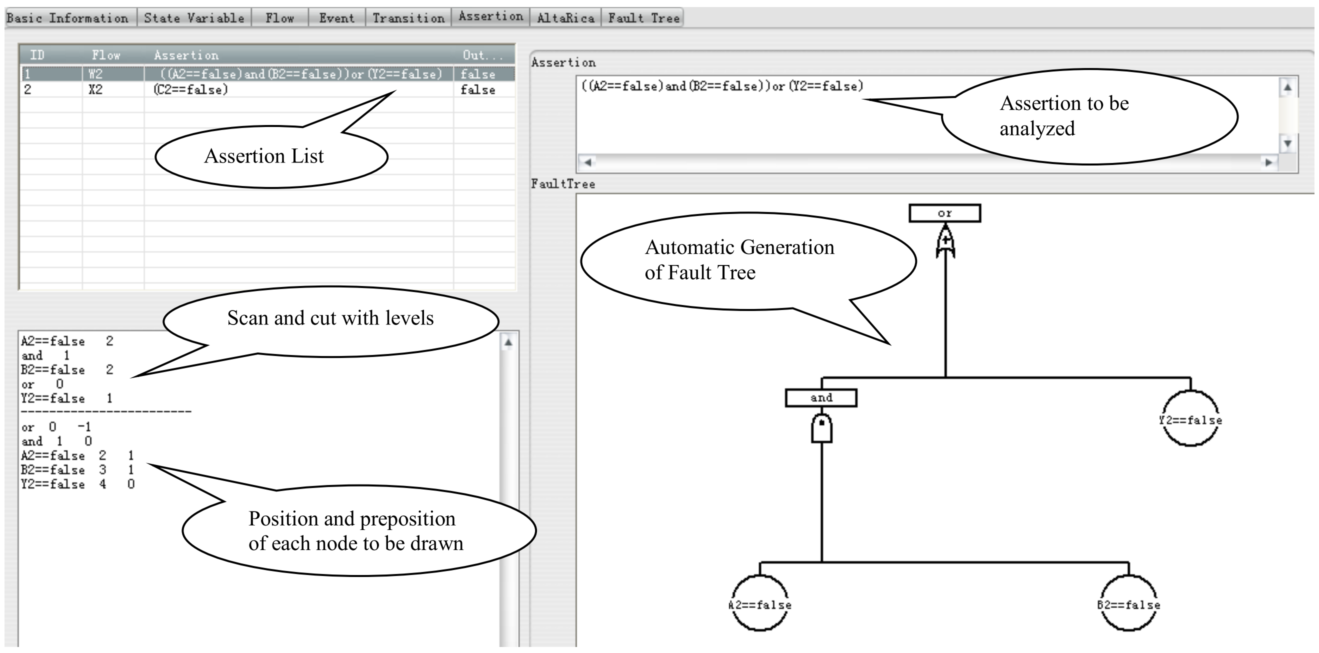
B. MULTIPLE NODES

This paper put forth the fault tree generation algorithm from multiple nodes connected which include five steps:

(1) Find the 'Out' node in current view
(2) Initialize the Analyser which has functions described in Part A of Section IV
(3) Traverse the LinkInfoArray to get and cut the assertion bind with the links connected to the 'Out' node in current view
(4) Analyze the link and assertion cut, get the class of variable in assertion
(5) Recursively trackback and generate the final unfolded assertion to be analyzed and get the position and preposition of each valid string or operation node in the final unfolded assertion

The first three steps in the algorithm are described in Fig 12.

The fourth and fifth steps are to recursively run an algorithm described in Fig 13 to get the position and preposition of each node need to be drawn in fault tree.

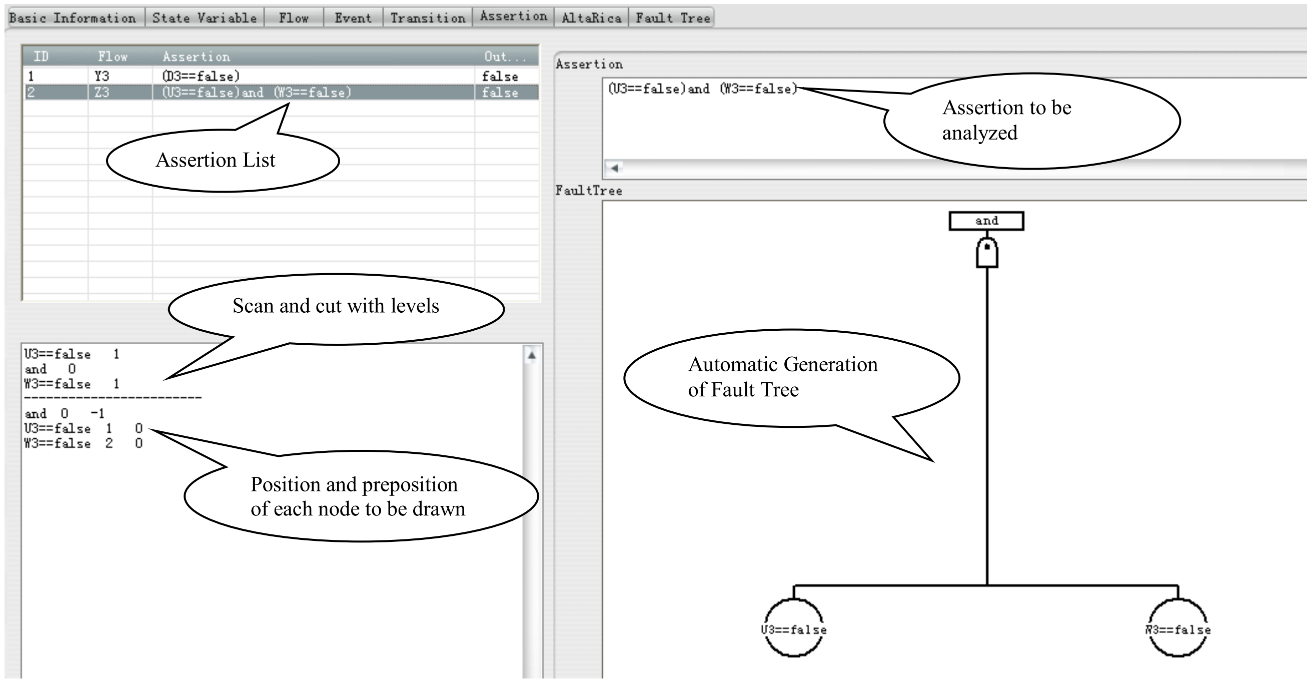


ID	Flow	Assertion	Out...
1	W2	((A2==false)and(B2==false))or(Y2==false)	false
2	X2	(C2==false)	false

```
A2==false   2
and   1
B2==false   2
or   0
Y2==false   1
-------------------------
or  0   -1
and  1   0
A2==false  2   1
B2==false  3   1
Y2==false  4   0
```

Assertion: ((A2==false)and(B2==false))or(Y2==false)

FIGURE 10. Final Result of Automatic Fault Tree Generation and Drawing of Class2.



ID	Flow	Assertion	Out...
1	Y3	(D3==false)	false
2	Z3	(U3==false)and (W3==false)	false

```
U3==false   1
and   0
W3==false   1
-------------------------
and  0   -1
U3==false  1   0
W3==false  2   0
```

Assertion: (U3==false)and (W3==false)

FIGURE 11. Final Result of Automatic Fault Tree Generation and Drawing of Class3.

Then, this paper makes three instances comp1, comp2 and comp3 of Class1, Class2 and Class3 as in Fig 14, and connect them to hand over fault by flow bind in the link between the three nodes.

The process, which is detailed described as following and in the related information is shown in Fig 15, includes:

(1) Get the linkinfo connects to "Out" node, which is "comp3.Z3"
(2) Find the comp3 is an instance of Class3 and Z3 is an outflow of Class3 by traversing all outflows in Class3
(3) The fault assertion of Z3 in Class3 is "(U3==false) and (W3==false)", so the linkinfo is expanded to "(U3==false) and (W3==false)"
(4) Find the U3 is an outflow, and comes from comp1 by analysis of the linkinfo "comp3.U3= comp1.U1"

```
 // OnSysFaultTreeGen
int i,j,size; CNode nodeOver;
CString strKeyOver=strViewKey+"_Out";// The key of 'Out' node
CString strLinkInfo,strAssertion,strClass;
size=ClassNodeToClassNodeLinkInfoArray.GetSize();
ClassNodeToClassNodeLinkInfo linkInfo;
//Initialize  the Analyser
 Analyser.m_strAssertion.Empty();
Analyser.AssertionNodeArray.RemoveAll();
Analyser.AssertionNodeTreeArray.RemoveAll();
//traverse the ClassNodeToClassNodeLinkInfoArray
for (i=0;i<size;i++)
{linkInfo= ClassNodeToClassNodeLinkInfoArray[i];
if (linkInfo.strViewKey==strViewKey)// link in current view
  {if (linkInfo.strOutKey==strKeyOver)//link connect to 'Out' node
   {strLinkInfo=linkInfo.strInAssertion;
    strLinkInfo.TrimLeft();strLinkInfo.TrimRight();
    //get Assertion bind to links connect to 'Out' node
    strAssertion=GetOverAssertion(strLinkInfo,strViewKey);
    //the link is from a node, not from subsystem in current view
    if(!strAssertionIsFromSubSys())
   {Analyser.m_strAssertion=strAssertion;
     //scan, cut and label the level
      Analyser.MakeAssertionToAssertionArray();
    for (j=0;j<Analyser.AssertionNodeArray.GetSize();j++)
    //set the key same to link's viewkey
    Analyser.AssertionNodeArray[j].strKey=linkInfo.strViewKey;
    //generating fault tree information of multiple nodes
     GenerateSysFaultTreeInfoArray(strAssertion,linkInfo);}
      }
   }}
```

FIGURE 12. Multiple Nodes Algorithm-Step 1-3.

(5) Find the comp1 is an instance of Class1 and U1 is an outflow by traversing all outflows in Class1

(6) The fault assertion of U1 in Class1 is "((A1==false) and (B1==false)) or (Y1==false)", so the linkinfo is expanded to "(((A1==false) and (B1==false)) or (Y1==false)) and (W3==false)"

(7) Recursive backtracking to analysis the "(W3==false)" in the linkinfo "(U3==false) and (W3==false)"

(8) Find the W3 is an outflow by traversing all outflows in Class3

(9) Find there is no inflow related to W3 from other node, then skip and return

(10) For there is no more expression to be analyzed, the process and recursive algorithm runs over

The result of recursive analysis on the linkinfo by flows and assertions in nodes are described in Table 3. The operation result of tool developed by this paper and all explanations are shown in Fig 16.

C. NODES WITH SUBSYSTEM

This paper put forward the fault tree generation algorithm among nodes with subsystem which includes steps partly same as multiple nodes, and the difference is in the analysis of the link connected to subsystem.

The differences include two situations are subsystem to node and node to subsystem which will be firstly analyzed of the information in the links between the node with subsystem.

```
    // GenerateSysFaultTreeInfoArray
    int i,j;
    CString strLinkInfo;
    CString str=Analyser.m_strAssertion;
    ClassNodeToClassNodeLinkInfo linkInfoPre;
    int size=Analyser.AssertionNodeArray.GetSize();
    for (j=0;j<Analyser.AssertionNodeArray.GetSize();j++)
    {//Get one node
    AssertionNode node=Analyser.AssertionNodeArray[j]
    If(!LogicOperation(node))// not logic operation
    {//Get and Analysis the interface information in LinkInfo
    strLinkInfo=node.strElement;
    //Get the flow information in node
    CString strOutFlow=GetOutFlowFromLinkInfo(strLinkInfo);
    CString strInFlow=GetInFlowFromLinkInfo(strLinkInfo);
    CString strNodeKey=linkInfo.strInKey;
    CString strViewKeyNow=linkInfo.strViewKey;
    //Get the class of node
    ClassNodeInfo
 infoNode=GetClassNodeInfoByClassNode(strViewKeyNow,strNodeKey);
    //Get Begin Part of linked assertion
    CString strPartBegin=infoNode.strName+"."+strOutFlow;
    //Get End Part of linked assertion
    CString strPartEnd=GetLinkInfoPartEnd(strPartBegin,strNodeKey,strViewKeyNow);
    //Get  the subsystem from Begin Part of linked assertion
    CString strSubNodeNameB=GetSubNodeFromPartBegin(strPartBegin);
    //Get  the subsystem from End Part of linked assertion
    CString strSubNodeNameE=GetSubNodeFromPartEnd(strPartEnd);
   CString strClassNodeNamePre=GetClassNodeNameFromPartEnd(strPartEnd);
   CString strOutFlow=GetFlowNameFromPartEnd(strPartEnd);
   CString strInFlow=GetFlowNameFromPartBegin(strPartBegin);
   CString strAssertionPre;
Lable1:
   //not from subsystem
if (strSubNodeNameB.IsEmpty()&&(strSubNodeNameE.IsEmpty()))
{strAssertionPre=GetAssertionPre(strFlow,strClassNodeNamePre,strViewKeyNow);
linkInfoPre=GetLinkPre(strPartBegin,strNodeKey,strViewKeyNow);
Lable2:
    if (!strAssertionPre.IsEmpty())
    {//Scan, cut and get the preposition of node
    Analyser.AssertionNodeArray[j].strElement=strAssertionPre;
    Analyser.ResetAssertion();
    Analyser.MakeAssertionToAssertionArray();
    //if has prenode, recursively run GenerateSysFaultTreeInfoArray()
    if(!linkInfoPre.strViewKey.IsEmpty())
    GenerateSysFaultTreeInfoArray(Analyser.AssertionNodeArray[j].str
Element,linkInfoPre);
    }}}
```

FIGURE 13. Multiple Nodes Algorithm-Step 4-5.

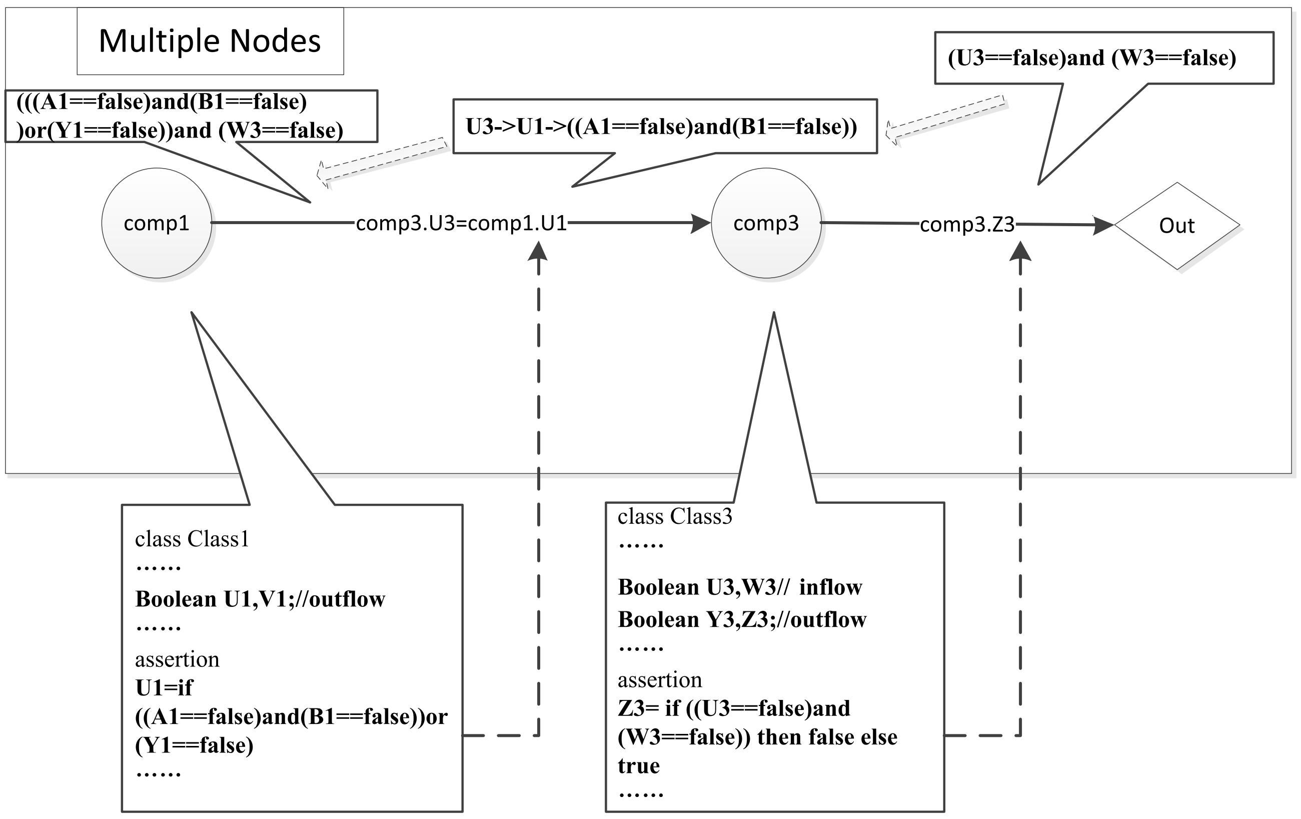
```
class Class1
class Class3
……
comp1:Class1
comp3:Class3
comp3.U3=Comp1.U1//connection
……
```

FIGURE 14. Description of the connection between multiple nodes.

For a simple example: compA.X=sub1.compA.Y, which is an information in the link from subsystem to node. We should analysis it as following:

(1) Cut the whole strings into two parts at "="

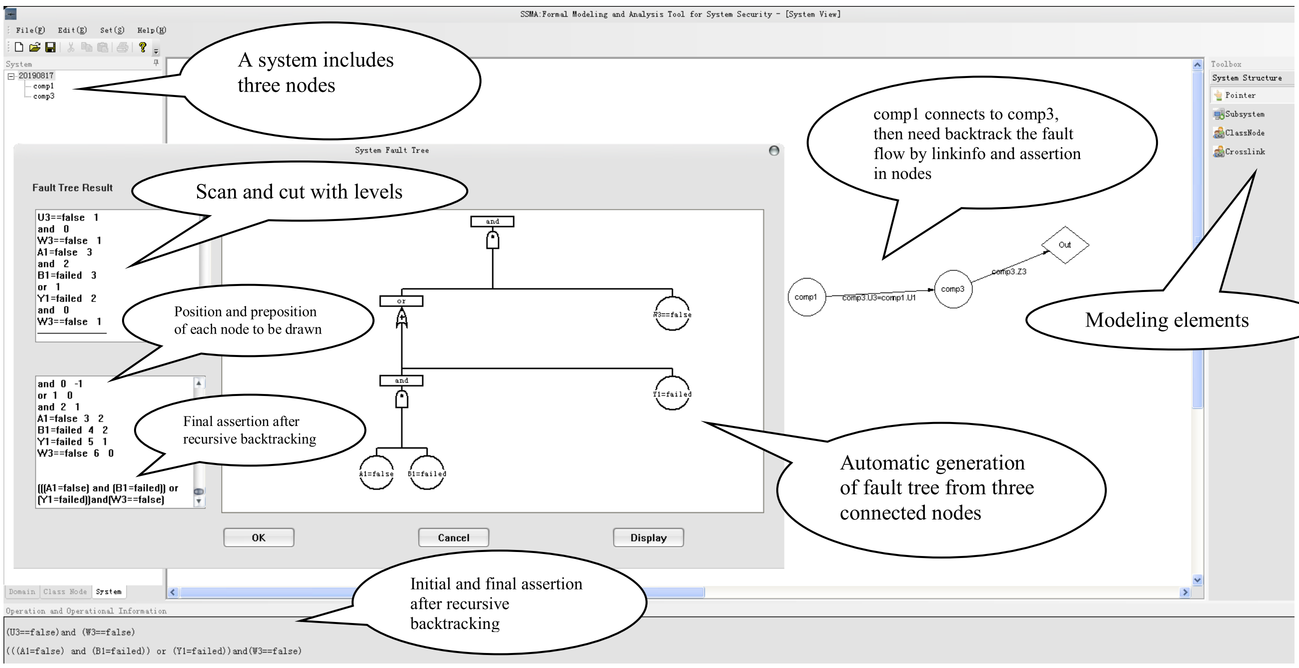
(2) By analysis of the strings in the left of equation, it is "compA.X" which only has one "."


Multiple Nodes
(((A1==false)and(B1==false))or(Y1==false))and (W3==false)
U3->U1->((A1==false)and(B1==false))
(U3==false)and (W3==false)
comp1
comp3.U3=comp1.U1
comp3
comp3.Z3
Out

```
class Class1
……
Boolean U1,V1;//outflow
……
assertion
U1=if
((A1==false)and(B1==false))or
(Y1==false)
……
```

```
class Class3
……
Boolean U3,W3// inflow
Boolean Y3,Z3;//outflow
……
assertion
Z3= if ((U3==false)and
(W3==false)) then false else
true
……
```

FIGURE 15. Dataflow and assertions run in multiple nodes.


SSMA:Formal Modeling and Analysis Tool for System Security - [System View]
File(F) Edit(E) Set(S) Help(H)
System
20190817
comp1
comp3
A system includes three nodes
System Fault Tree
Fault Tree Result
Scan and cut with levels
Position and preposition of each node to be drawn
Final assertion after recursive backtracking
and
or
and
W3==false
Y1=failed
A1=false
B1=failed
OK
Cancel
Display
comp1 connects to comp3, then need backtrack the fault flow by linkinfo and assertion in nodes
comp1
comp3.U3=comp1.U1
comp3
comp3.Z3
Out
Modeling elements
Toolbox
System Structure
Pointer
Subsystem
ClassNode
Crosslink
Automatic generation of fault tree from three connected nodes
Initial and final assertion after recursive backtracking
Domain
Class Node
System
Operation and Operational Information

```
U3==false  1
and  0
W3==false  1
A1=false  3
and  2
B1=failed  3
or  1
Y1=failed  2
and  0
W3==false  1
```

```
and  0  -1
or  1  0
and  2  1
A1=false  3  2
B1=failed  4  2
Y1=failed  5  1
W3==false  6  0

[((A1=false) and (B1=failed)) or
(Y1=failed))and(W3==false)
```

```
(U3==false)and (W3==false)
(((A1=false) and (B1=failed)) or (Y1=failed))and(W3==false)
```

FIGURE 16. Result and explanations of running algorithm on multiple nodes.

(3) that means the strings in the left of equation is related to a node
(4) X is an inflow of node compA, and compA is in the level of current system
(5) By traversing all nodes in the level of current system, we can get that compA is an instance of what class
(6) Then if an assertion in this class related to inflow X, it will be partly replaced

TABLE 3. Result of rescursive analysis on the multiple nodes.

running times of recursive algorithm	linkinfo	original node	target node	assertion	linkinfo after running recursive algorithm
1	comp3.Z3	comp3	Out	(U3==false) and (W3==false)	(U3==false) and (W3==false)
2	comp3.U3= comp1.U1	comp1	comp3	((A1==false) and (B1==false)) or (Y1==false)	(((A1==false) and (B1==false)) or (Y1==false)) and (W3==false)

```
Lable1:
  //if subsystem to node, reset current view to the subsystem view
if ((strSubNodeNameB.IsEmpty()&&(!strSubNodeNameE.IsEmpty()))
{linkInfoPre=GetLinkPre(strPartBegin,strNodeKey,strViewKeyNow);
 CString strOverKey=linkInfoPre.strInKey+"_Out";
 for (i=0;i< ClassNodeToClassNodeLinkInfoArray.GetSize();i++)
{
ClassNodeToClassNodeLinkInfo linkInfoTemp= ClassNodeToClassNodeLinkInfoArray[i];
//get the link which links to the OutNode in the view of subsystem
    if((linkInfoTemp.strViewKey==linkInfoPre.strInKey)
    &&(linkInfoTemp.strOutKey==strOverKey))
    {strAssertionPre=linkInfoTemp.strInAssertion;
    CString strOut=(strClassNodeNamePre+"."+strOutFlow);
    if(strAssertionPre==strOut){
     //set the linkInfoPre to this link in subsystem
     linkInfoPre=linkInfoTemp;
     break;
     }
 }}
 //get the node connect to OutNode in subsystem
      ClassNodeInfo InfoNodeIn=
GetClassNodeInfoByClassNode(linkInfoPre.strViewKey,linkInfoPre.strInKey);
//get what class is this node
 CString strClassIn=infoNodeIn.strClass;
strAssertionPre=GetAssertionPre(strFlow,infoNodeIn.strName,linkInfo
Pre.strViewKey);
Lable2:
```

FIGURE 17. Algorithm of the Link from Subsystem to Node.

```
Lable1:
  //if to subsystem, reset the current view to the subsystem view
if (!strSubNodeNameB.IsEmpty()&&(strSubNodeNameE.IsEmpty())
{linkInfoPost=GetLinkPost(strPartEnd,strNodeKey,strViewKeyNow);
 CString strInKey=linkInfoPre.strOutKey+"_In";
 for (i=0;i< ClassNodeToClassNodeLinkInfoArray.GetSize();i++)
{
ClassNodeToClassNodeLinkInfo linkInfoTemp= ClassNodeToClassNodeLinkInfoArray[i];
//get the link which links to the InNode in the view of subsystem
    if((linkInfoTemp.strViewKey==linkInfoPost.strOutKey)
    &&(linkInfoTemp.strInKey==strInKey))
    {strAssertionPost=linkInfoTemp.strOutAssertion;
    CString strIn=(strClassNodeNamePost+"."+strInFlow);
    if(strAssertionPost==strIn){
     //set the linkInfoPost to this link in subsystem
     linkInfoPost=linkInfoTemp;
     break;
     }
 }}
 //get the node connect toInNode in subsystem
      ClassNodeInfo InfoNodeOut=
GetClassNodeInfoByClassNode(linkInfoPost.strViewKey,linkInfoPre.strInKey);
//get what class is this node
 CString strClassOut=infoNodeOut.strClass;
strAssertionPost=GetAssertionPost(strInFlow,infoNodeOut.strName,lin
kInfoPost.strViewKey);
Lable2:
```

FIGURE 18. Algorithm of the Link from Node to Subsystem.

(7) By analysis of the strings in the right of equation, it is sub1.comp2.W2 which has two ".”
(8) That means the strings in the right of equation is related to a node from subsystem
(9) By cutting and analysis the strings, we get that the Y is an outflow from node compA in subsystem sub1
(10) By traversing all nodes in the level of subsystem sub1, we can get that compA is an instance of what class. (For there may exist many nodes have same name in different levels of current system or subsystem, so those nodes may be instances of different class)
(11) Get the assertion in this class related to outflow Y and partly replace the assertion in that class related to inflow X.

Of course, we may also make a simple example: sub1.compA.Y = compA.X, which is an information in the link from node to subsystem. The process of dealing with it in this situation can be mostly referenced by what we have mentioned above.

Comparing to the algorithm of multiple nodes, there have some differences in nodes to subsystem:

```
class Class1
class Class2
class Class3
……
comp1:Class1
sub1: subsystem
sub1.comp2:Class2
comp3:Class3
……
Out=comp3.Z3
comp3.U3=comp1.U1//connection
comp3.W3=sub1.comp2.W2//connection
……
```

FIGURE 19. Description of the nodes and subsystem with connections.

(1) By cutting and analysis the information in the link, we should distinguish the information is related to nodes with subsystem or between nodes only
(2) We should get the subsystem, the nodes and the inflow or outflow by analysis of the information
(3) We should traverse all nodes in the subsystem to get what class is the nodes instance from
(4) We should come into the level of subsystem to get the related assertion of the inflow or outflow in the class

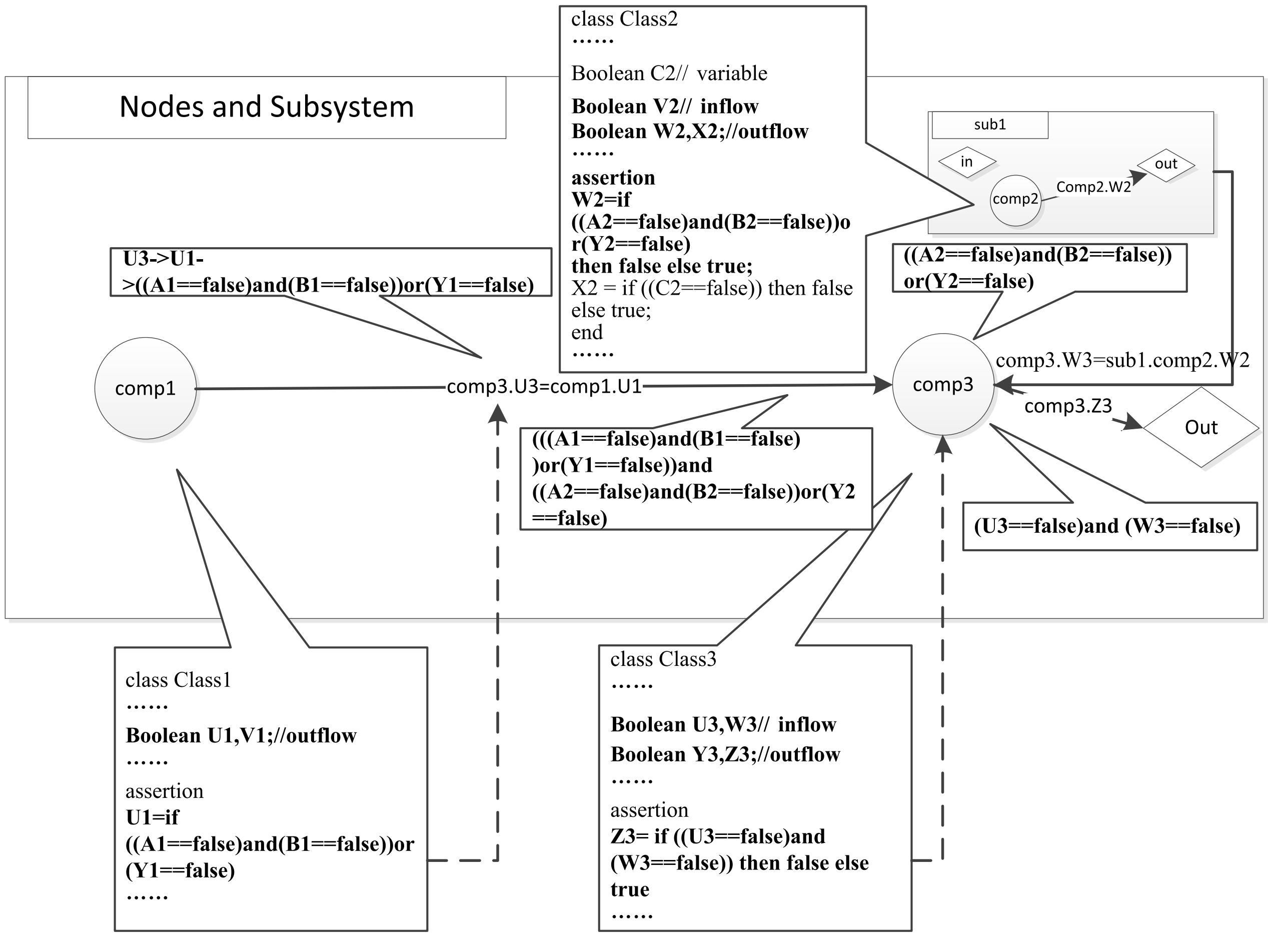

Nodes and Subsystem
class Class2
……
Boolean C2// variable
Boolean V2// inflow
Boolean W2,X2;//outflow
……
assertion
W2=if ((A2==false)and(B2==false))or(Y2==false)
then false else true;
X2 = if ((C2==false)) then false else true;
end
……
sub1
in
out
comp2
Comp2.W2
((A2==false)and(B2==false)) or(Y2==false)
U3->U1->((A1==false)and(B1==false))or(Y1==false)
comp1
comp3.U3=comp1.U1
comp3
comp3.W3=sub1.comp2.W2
comp3.Z3
Out
(((A1==false)and(B1==false))or(Y1==false))and ((A2==false)and(B2==false))or(Y2==false)
(U3==false)and (W3==false)
class Class1
……
Boolean U1,V1;//outflow
……
assertion
U1=if ((A1==false)and(B1==false))or (Y1==false)
……
class Class3
……
Boolean U3,W3// inflow
Boolean Y3,Z3;//outflow
……
assertion
Z3= if ((U3==false)and (W3==false)) then false else true
……

FIGURE 20. Dataflow and assertions run in nodes and subsystem.

(5) We should go on traversing if the node in the subsystem has pre-node
(6) We should back to the system level if all nodes in the level of subsystem has been transacted
(7) We should set the view point correctly when coming into or out from the subsystem in developing the tool SSMA.

The detailed algorithm of dealing with the situations including nodes to subsystem and subsystem to node are between the ***Lable1*** and ***Lable2*** as in Fig 17 and Fig 18.

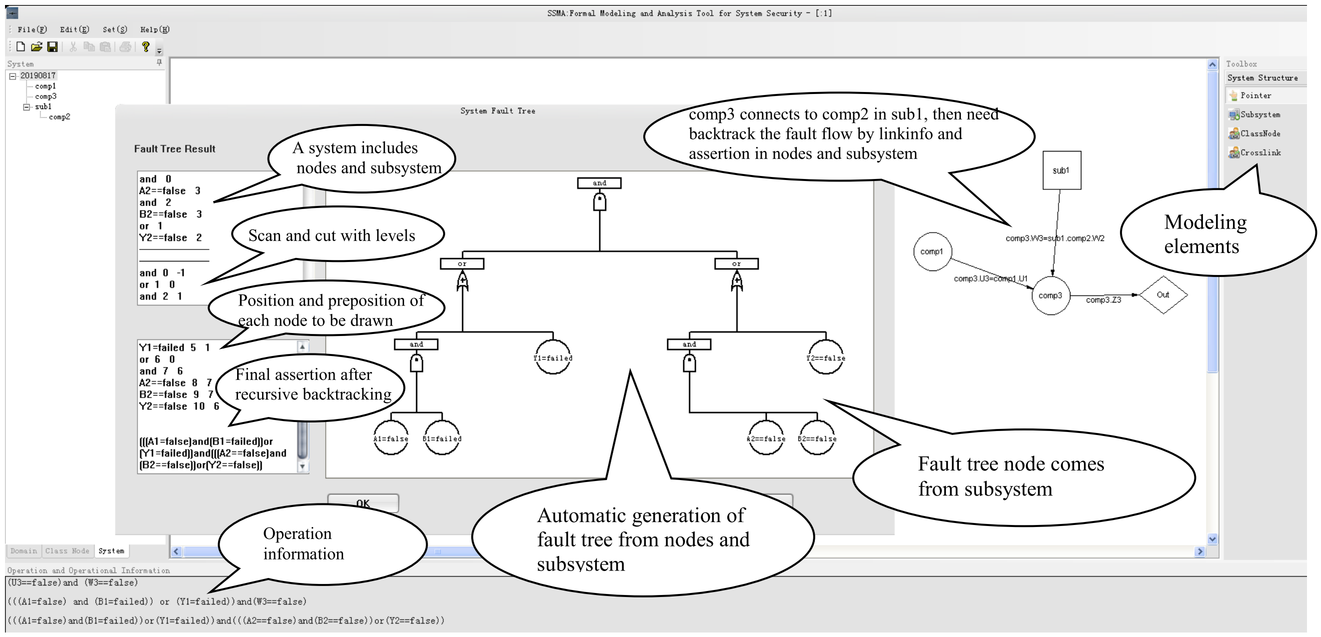
(1) The algorithm of analyzing link from subsystem to node is described in details as Fig 17.
(2) The algorithm of dealing with the link from node to subsystem is described in details as Fig 18. The difference is from *Lable1* to *Lable2*.

After the algorithm in Fig 17 and Fig 18, the link will go into the subsystem and there will recursively run to get the corrected fault information bind in links and automatically generate fault tree.

Then, this paper makes two instance comp1 and comp3 of Class1 and Class3, and comp2 of Class2 in subsystem as in Fig 19, and connect them to hand over fault by flow bind in the link among nodes and subsystem.

The process, which is detailed described as following and the related information is shown in Fig 20, includes:

(1) Get the linkinfo connects to "Out" node, which is "comp3.Z3"
(2) Find the comp3 is an instance of Class3 and Z3 is an outflow of Class3 by traversing all outflows in Class3
(3) The fault assertion of Z3 in Class3 is "(U3==false) and (W3==false)", so the linkinfo is expanded to "(U3==false) and (W3==false)"
(4) Find the U3 is an outflow, and comes from comp1 by analysis of the linkinfo "comp3.U3=comp1.U1"
(5) Find the comp1 is an instance of Class1 and U1 is an outflow by traversing all outflows in Class1
(6) The fault assertion of U1 in Class1 is "((A1==false) and (B1==false)) or (Y1==false)", so the linkinfo is expanded to "(((A1==false) and (B1==false)) or (Y1==false)) and (W3==false)"
(7) Recursive backtracking to analysis the "(W3==false)" in the linkinfo "(U3==false) and (W3==false)"
(8) Find the W3 is an outflow by traversing all outflows in Class3, and comes from comp2 in sub1, by analysis of "comp3.W3=sub1.comp2.W2"


SSMA:Formal Modeling and Analysis Tool for System Security - [:1]
File(F) Edit(E) Set(S) Help(H)
System
20190817
comp1
comp3
sub1
comp2
System Fault Tree
Fault Tree Result
A system includes nodes and subsystem
Scan and cut with levels
Position and preposition of each node to be drawn
Final assertion after recursive backtracking
comp3 connects to comp2 in sub1, then need backtrack the fault flow by linkinfo and assertion in nodes and subsystem
and
or
Y1=failed
Y2==false
A1=false
B1=failed
A2==false
B2==false
sub1
comp1
comp3
Out
comp3.W3=sub1.comp2.W2
comp3.U3=comp1.U1
comp3.Z3
Toolbox
System Structure
Pointer
Subsystem
ClassNode
Crosslink
Modeling elements
Fault tree node comes from subsystem
Automatic generation of fault tree from nodes and subsystem
Operation information
OK
Domain
Class Node
System
Operation and Operational Information

```
and  0
A2==false  3
and  2
B2==false  3
or  1
Y2==false  2

and  0  -1
or  1  0
and  2  1

Y1=failed  5  1
or  6  0
and  7  6
A2==false  8  7
B2==false  9  7
Y2==false  10  6

(((A1=false)and(B1=failed))or
(Y1=failed))and(((A2==false)and
(B2==false))or(Y2==false))
```

```
(U3==false)and (W3==false)
(((A1=false) and (B1=failed)) or (Y1=failed))and(W3==false)
(((A1=false)and(B1=failed))or(Y1=failed))and(((A2==false)and(B2==false))or(Y2==false))
```

FIGURE 21. **Result and all explanations of running algorithm on nodes and subsystem.**

TABLE 4. **Result of rescursive analysis on the nodes and subsystem.**

running times of recursive algorithm	linkinfo	original node	target node	assertion	linkinfo after running recursive algorithm
1	comp3.Z3	comp3	Out	(U3==false) and (W3==false)	(U3==false) and (W3==false)
2	comp3.U3= comp1.U1	comp1	comp3	((A1==false) and (B1==false)) or (Y1==false)	(((A1==false) and (B1==false)) or (Y1==false)) and (W3==false)
3	comp3.W3 =sub1.com p2.W2	sub1.comp2	comp3	(((A2==false) and (B2==false)) or (Y2==false))	(((A1==false) and (B1==false)) or (Y1==false))) and (((A2==false) and (B2==false)or (Y2==false))

(9) Find the outflow W2 in comp2 which is an instance of Class2, which related to the assertion "((A2==false) and (B2==false)) or (Y2==false)", so the linkinfo is expanded to "(((A1==false) and (B1==false)) or (Y1==false)) and (W3==false)) and (((A2==false) and (B2==false) or (Y2==false))"

(10) For there is no more expression to be analyzed, the process and recursive algorithm runs over

The result of recursive analysis on the linkinfo by flows and assertions in nodes and subsystem are described in Table 4. The operation result of tool developed by this paper and all explanations are shown in Fig 21.

V. OVERVIEW OF TOOL DEVELOPMENT

This paper here made a brief introduction of the tool named SSMA (System Safety Modeling and Analysis).

This development of System Safety Modeling and Analysis (SSMA) mainly uses Visual C++ programming to realize AltaRica's hierarchical system modeling and fault tree analysis, which includes:

- Interface Framework

The main interface framework realizes the functions of project file creation, domain modeling panel, class modeling panel and system modeling panel creation, graphical modeling view, modeling element list and operation information window.

- Class Modeling

Class modeling implements the visual modeling functions of class name, domain state, class state, input flow, output flow, event, transition, assertion, etc. After the modeling is completed, the class information is expanded hierarchically in the left class modeling area.

- System Modeling

System modeling realizes the new construction of subsystems and class nodes. Creating a node should be an instance from a class in the left list. When having created a subsystem, it will come into the internal by double click the icon of this subsystem on the left list.

- Automatic fault tree generation of class node

Automatic fault tree generation of class node include automatic analysis of assertion in class node, and automatic generation of fault tree in graphic.

- Automatic fault tree generation of multiple nodes

Automatic fault tree generation of multiple nodes include interface description between connected nodes, and

```
class Class1
Boolean A1,B1// variable
Boolean Y1,X1// inflow
Boolean U1,V1;//outflow, add V1 as an outflow
event eA1,eA2,eB1,eB2;
transition
eA1: A1==true ->A1:=false;
eA2: A1==false -> A1 := true;
eB1: B1==true ->B1:=false;
eB2: B1==false -> B1 := true;
assertion
U1=if ((A1==false)and(B1==false))or(Y1==false)
then false else true;
V1 = if (((A1=false) or (X1=false))) then false else true;
end
```

FIGURE 22. Details of Modified Class1.

```
class Class2
Boolean C2// variable
Boolean V2,U2// inflow, add U2 as a inflow
Boolean W2,X2;//outflow
……
Assertion
//(U2==false)
W2=if ((A2==false)and(B2==false))or(U2==false)
then false else true;
X2 = if ((C2==false)) then false else true;
end
```

FIGURE 23. Details of Modified Class2.

```
class Class4
Boolean C4// variable
Boolean V4,U4//inflow
Boolean W4;//outflow
……
Assertion
W4=if ((U4==false) and (V4==false)) then false else true;
end
```

FIGURE 24. Details of Class4.

automatic analysis of interface description, and automatic generation of fault tree of multiple nodes in graphic.

- Automatic fault tree generation of nested system

Automatic fault tree generation of nested system include interface description between subsystem and node, automatic analysis of interface description, and automatic generation of fault tree in graphic.

SSMA has a friendly interface, good usability and ability of system modeling to well support AltaRica. We can easily build system model in graphic and automatically implement the fault analysis in high efficiency. It is easy to be transplanted to integration of other tools run in Windows platform of Micro Software.

VI. DETAILED EXAMPLE

This paper here made a detailed case study of the nested system modeling and automatic fault tree generation from single class node to multiple nodes and nodes with subsystem.

In the case study, this paper firstly created three classes of class1, class2, class3, class4 and class5, and two subsystem s1 and s2.

```
class Class5
Boolean A5// variable
Boolean V5,U5// inflow
Boolean W5;//outflow
……
Assertion
W5=if ((U5==false) and (A5==false)) then false else true;
end
```

FIGURE 25. Details of Class5.

```
class Class1
class Class2
class Class3
class Class4
class Class5

comp1:Class1
comp2:Class2
comp3:Class3
s1: subsystem
s2: subsystem
s1.comp4:Class4
s2.comp5:class5

Out=comp3.Z3

comp3.U3=s2.comp5.W5
comp3.W3=comp2.W2

comp2.V2=comp1.V1
comp2.U2=s1.comp4.W4

comp1.X1=comp2.X2
comp1.Y1=comp4.Y3

s2.comp5.U5=comp1.U1
```

FIGURE 26. Description of the system in case study.

The class1 and class2 had been modified to show the fault propagation between nodes and subsystem which are introduced in section IV. And we here made a new description of Class4, Class5, s1 and s2.

As shown in Fig22, Class1 add V1 as an outflow, that means the assertion bind with it will propagate when V1 as an inflow to other node or subsystem.

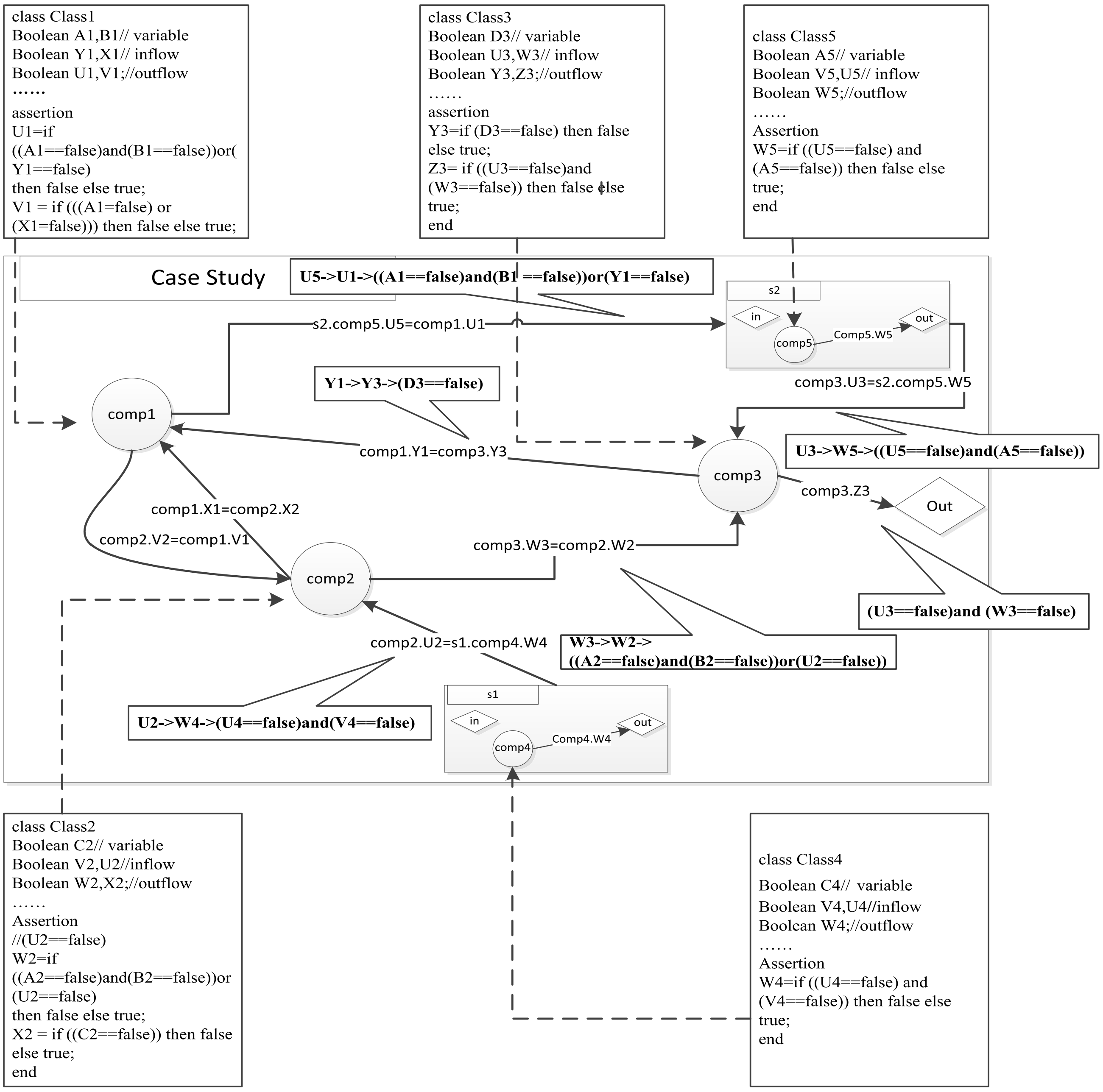
As shown in Fig23, Class2 add U2 as an inflow, that means the assertion bind with outflow in other node or subsystem will propagate into Class2 when the outflow connect with U2. Class2 also modified the assertion bind with W2 changing (Y2==false) into (U2==false) which will cause a fault propagation if U2 connects to outflows comes from other nodes or subsystems.

The details of Class4 is descrbed in Fig 24.

The details of Class5 is descrbed in Fig 25.

Then, this paper made comp1, comp2 and comp3 of Class1, Class2 and Class3, comp4 of Class4 in subsystem s1, comp5 of Class5 in subsystem s2, and connect them to hand over fault by flows bind in the link among nodes and subsystems which is described in Fig 26.

The process, which is detailed described as following and the related information is shown in Fig 27, includes:



```
class Class1
Boolean A1,B1// variable
Boolean Y1,X1// inflow
Boolean U1,V1;//outflow
……
assertion
U1=if
((A1==false)and(B1==false))or(
Y1==false)
then false else true;
V1 = if (((A1=false) or
(X1=false))) then false else true;
```

```
class Class3
Boolean D3// variable
Boolean U3,W3// inflow
Boolean Y3,Z3;//outflow
……
assertion
Y3=if (D3==false) then false
else true;
Z3= if ((U3==false)and
(W3==false)) then false else
true;
end
```

```
class Class5
Boolean A5// variable
Boolean V5,U5// inflow
Boolean W5;//outflow
……
Assertion
W5=if ((U5==false) and
(A5==false)) then false else
true;
end
```

```
class Class2
Boolean C2// variable
Boolean V2,U2//inflow
Boolean W2,X2;//outflow
……
Assertion
//(U2==false)
W2=if
((A2==false)and(B2==false))or
(U2==false)
then false else true;
X2 = if ((C2==false)) then false
else true;
end
```

```
class Class4
Boolean C4// variable
Boolean V4,U4//inflow
Boolean W4;//outflow
……
Assertion
W4=if ((U4==false) and
(V4==false)) then false else
true;
end
```

FIGURE 27. Dataflow and assertions run in case study.

(1) Get the linkinfo connects to "Out" node, which is "comp3.Z3"
(2) Find the comp3 is an instance of Class3 and Z3 is an outflow of Class3 by traversing all outflows in Class3, then modify the linkinfo into "(U3==false) and (W3==false)
(3) Find the U3 is an inflow come from comp5 in s2, then find that the U3 is connected to an outflow W5 in comp5 in s2
(4) Get the assertion of W5 and modify the linkinfo into ((U5==false) and (A5==false)) and (W3==false)
(5) Find the U5 is an inflow that connect to U1 in comp1
(6) Get the assertion of U1 and modify the linkinfo into ((((A1==false) and (B1==false)) or (Y1==false)) and (A5==false)) and (W3==false)
(7) Find the Y1 is an inflow and comes from Y3 in comp3
(8) Get the assertion of Y3 and modify the linkinfo into ((((A1==false) and (B1==false)) or ((D3==false))) and (A5==false)) and (W3==false)
(9) Find the A5 is not an inflow in comp5 which means that there is no outflow from other node or subsystem, then back and continue analysis
(10) Find the W3 is an inflow which comes from W2 in comp2

TABLE 5. Result of rescursive analysis on the case study.

running times of recursive algorithm	linkinfo	original node	target node	assertion	linkinfo after running recursive algorithm
1	comp3.Z3	comp3	Out	(U3==false) and (W3==false)	(U3==false) and (W3==false)
2	comp3.U3= s2.comp5. W5	s2.comp5	comp3	(U5==false)and(A5==false)	((U5==false)and(A5==false)) and(W3==false)
3	s2.comp5. U5=comp1. U1	comp1	s2.comp5	((A1==false)and(B1==false)) or(Y1==false)	((((A1==false)and(B1==false))or(Y1==false))and(A5==false))and(W3==false)
4	comp1.Y1= comp3.Y3	comp3	comp1	(D3==false)	((((A1==false)and(B1==false))or((D3==false)))and(A5==false))and(W3==false)
5	comp3.W3 =comp2.W 2	comp2	comp3	((A2==false)and(B2==false))or (U2==false)	((((A1==false)and(B1==false))or((D3==false)))and(A5==false))and(((A2==false) and(B2==false))or(U2==false))
6	comp2.U2= s1.comp4. W4	s1.comp4	comp2	(U4==false) and (V4==false)	((((A1==false)and(B1==false))or((D3==false)))and(A5==false))and(((A2==false)and(B2==false))or((U4==false) and (V4==false)))

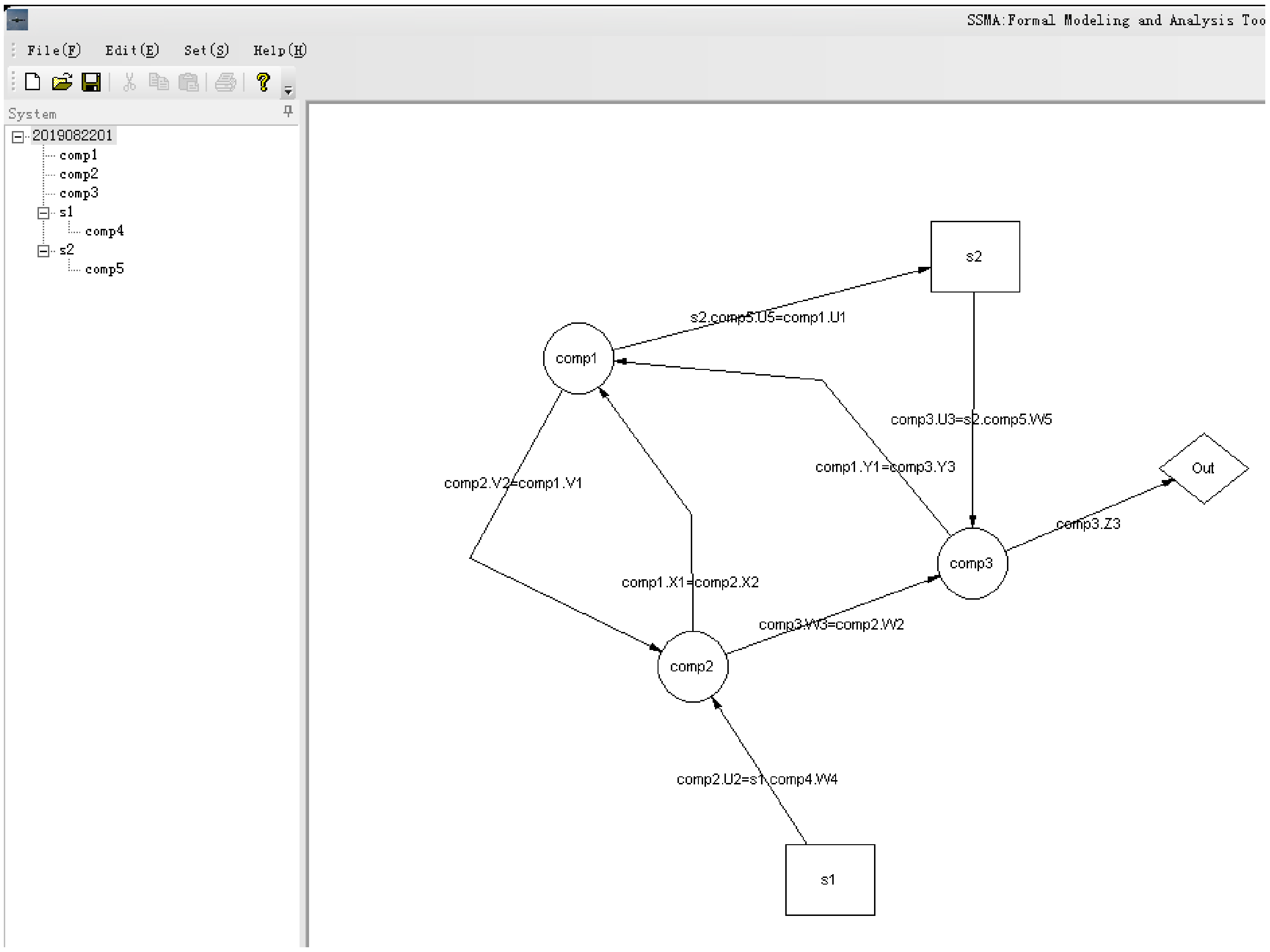


FIGURE 28. System modeling using SSMA in case study.

(11) Get the assertion of W2 and modify the linkinfo into ((((A1==false) and (B1==false)) or ((D3==false))) and (A5==false)) and (((A2==false) and (B2==false)) or (U2==false))

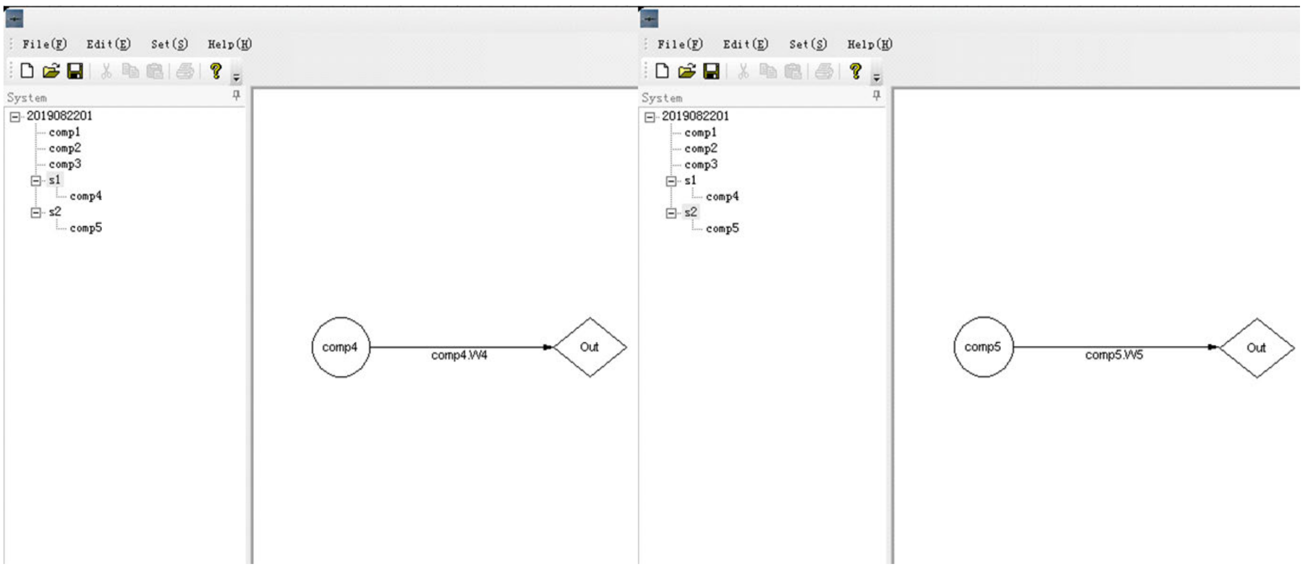
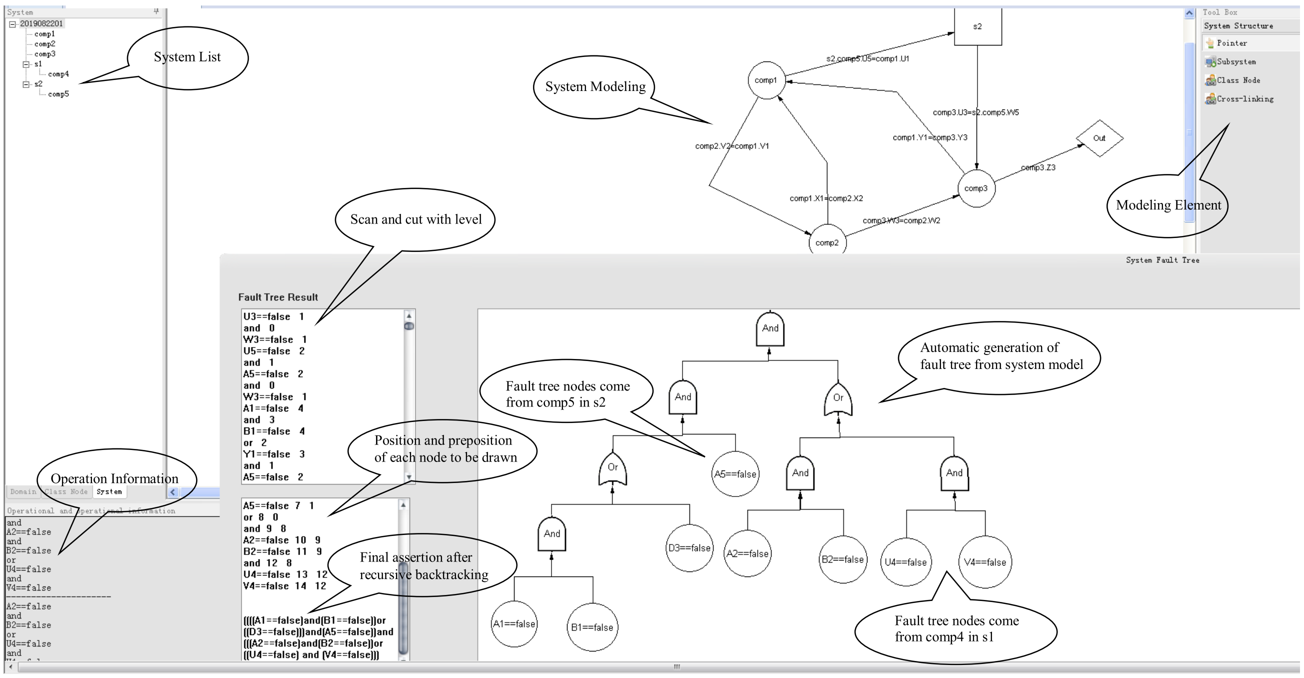


FIGURE 29. Details of s1 and s2.



```
U3==false  1
and  0
W3==false  1
U5==false  2
and  1
A5==false  2
and  0
W3==false  1
A1==false  4
and  3
B1==false  4
or  2
Y1==false  3
and  1
A5==false  2
```

```
A5==false  7  1
or  8  0
and  9  8
A2==false  10  9
B2==false  11  9
and  12  8
U4==false  13  12
V4==false  14  12


((((A1==false)and(B1==false))or
((D3==false)))and(A5==false))and
(((A2==false)and(B2==false))or
((U4==false) and (V4==false)))
```

```
and
A2==false
and
B2==false
or
U4==false
and
V4==false
----------------------
A2==false
and
B2==false
or
U4==false
and
```

FIGURE 30. Result and all explanations of running algorithm on case study.

(12) Find A5, A2 and B2 are all not flows, then back and continue analysis
(13) Find U2 is an inflow and connect with W4 of comp4 in s1
(14) Get the assertion of W4 and modify the linkinfo into ((((A1==false) and (B1==false)) or ((D3== false))) and (A5==false)) and (((A2==false) and (B2==false)) or ((U4==false) and (V4==false)))”
(15) Find U4 and V4 are inflows, but no connection to them. Then back and continue analysis
(16) All expressions have been traversed and analyzed, so the algorithm comes to the end

The result of recursive analysis on the linkinfo by flows and assertions in case study are described in Table 5.

The system modeling using SSMA in the case study are shown in Fig 28, included comp1, comp2, comp3, s1, s2 and linkinfo. The details of s1 and s2 are shown in Fig 29.

The operation result of case study and all explanations are shown in Fig 30. In this figure, we can see the final assertion after.

We may find the “Final assertion after recursing back-tracking” on the Figure 30, that is “((((A1==false) and (B1==false)) or ((D3==false))) and (A5==false)) and (((A2==false) and (B2==false)) or ((U4==false) and

(V4==false)))". The final assertion is the same as the line 6 of "linkinfo after running recursive algorithm" in Table 5 which proves the final assertion by using the algorithm proposed in this paper has been correctly transacted by computer. And more, the nodes and strcuture in the final whole system fault tree in Figure 30 is also correct in accordance with the "Final assertion after recursing backtracking" on the Figure 30.

VII. CONCLUSION

Based on the language specification of AltaRica, this paper studied the grammar and semantics of AltaRica language, combined visual modeling with fault logic description ability of AltaRica language, designed and implemented automatic fault tree generation algorithm from node to nested system, improved the ability of safety modeling and analysis, and realized synchronization of fault modeling and analysis.

This paper studied the structure and semantics of AltaRica, detailed illustrated the system modeling and automation fault tree generation including class, node and subsystem not only in theory but also is open in detailed programing and visual tool realization comparing to existed papers that some are absent in details of modeling and automation in visual fault tree generation and others are only focusing on the method in application using existed tools.

From the example it is found that's easy to modify and maintain the class, system and fault model, and greatly improves the accuracy and efficiency of fault modeling and analysis of safety critical system.

The method proposed in this paper and SSMA have been successfully applied in fault modeling and analysis of marine generator excitation system which will appear in our future paper.

Further work will focus on the situations of subsystem to subsystem, the issues on quantitative algorithm, and improvement on the artistic style of SSMA.

REFERENCES

[1] N. R. Storey, *Safety Critical Computer Systems*. Reading, MA, USA: Addison-Wesley, 1996.

[2] M. L. Chiozza and C. Ponzetti, "FMEA: A model for reducing medical errors," *Clinica Chim. Acta*, vol. 404, no. 1, pp. 75–78, Jun. 2009.

[3] L. M. Mcelroy *et al.*, "Fault tree analysis," *Amer. J. Med. Qual.*, vol. 32, no. 1, pp. 80–86, 2017.

[4] O. Lisagor, T. Kelly, and R. Niu, "Model-based safety assessment: Review of the discipline and its challenges," in *Proc. 9th Int. Conf. Rel., Maintainability Saf.*, Jun. 2011, 625-632.

[5] P. Fenelon, J. A. McDermid, M. Nicolson, and D. J. Pumfrey, "Towards integrated safety analysis and design," *ACM SIGAPP Appl. Comput. Rev.*, vol. 2, no. 1, pp. 21–32, Mar. 1994.

[6] Y. Papadopoulos and J. A. McDermid, "Hierarchically performed hazard origin and propagation studies," in *Proc. 18th Int. Conf. Comput. Saf., Rel., Secur. (SAFECOMP)*. Toulouse, France: Springer-Verlag, 1999, pp. 139–152.

[7] A. Arnold, G. Point, A. Griffault, and A. Rauzy, "The AltaRica formalism for describing concurrent systems," *Fundamenta Informaticae*, vol. 40, no. 2,3, pp. 109–124, 1999.

[8] J.-Y. Choley, F. Mhenni, N. Nguyen, and A. Baklouti, "Topology-based safety analysis for safety critical CPS," *Procedia Comput. Sci.*, vol. 95, pp. 32–39, Jan. 2016.

[9] L. Chen, J. Jiao, Q. Wei, and T. Zhao, "An improved formal failure analysis approach for safety-critical system based on MBSA," *Eng. Failure Anal.*, vol. 82, pp. 713–725, Dec. 2017.

[10] S. Kabir and Y. Papadopoulos, "Applications of Bayesian networks and Petri nets in safety, reliability, and risk assessments: A review," *Saf. Sci.*, vol. 115, pp. 154–175, Jun. 2019.

[11] G. Point and A. Rauzy, "AltaRica: Constraint automata as a description language," *Eur. J. Automat.*, vol. 33, nos. 8–9, pp. 1033–1052, 1999.

[12] M. Batteux, T. Prosvirnova, and A. Rauzy, "AltaRica 3.0 assertions: The why and wherefores," *Proc. Inst. Mech. Eng., O, J. Risk Rel.*, vol. 231, pp. 691–700, Sep. 2017.

[13] M. Issad, L. Kloul, and A. Rauzy, "Scenario-oriented reverse engineering of complex railway system specifications," *Syst. Eng.*, vol. 21, no. 2, pp. 91–104, Mar. 2018.

[14] A. Albore, S. D. Zilio, G. Infantes, C. Seguin, and P. Virelizier, "A model-checking approach to analyse temporal failure propagation with AltaRica," in *Proc. Model-Based Saf. Assessment (IMBSA)*, Trento, Italy, Sep. 2017, pp. 147–162.

[15] M. Lipaczewski, F. Ortmeier, T. Prosvirnova, and A. Rauzy, "Comparison of modeling formalisms for safety analyses: SAML and AltaRica," *Rel. Eng. Syst. Saf.*, vol. 140, pp. 191–199, Aug. 2015.

[16] J. Brunel, P. Feiler, J. Hugues, B. Lewis, T. Prosvirnova, C. Seguin, and L. Wrage, "Performing safety analyses with AADL and AltaRica," in *Proc. Int. Symp. Model-Based Saf. Assessment*. Cham, Switzerland: Springer, 2017, pp. 67–81.

[17] A. Cherfi, A. Rauzy, and M. Leeman, "AltaRica 3 based models for ISO 26262 automotive safety mechanisms," in *Model-Based Safety and Assessment*. Cham, Switzerland: Springer, 2014, pp. 123–136.

[18] S. Duo and S. Li, "A practicable safety modeling methodology for aircraft systems using AltaRica," *Procedia Eng.*, vol. 80, pp. 127–139, 2014.

[19] G. Long and A. Liang, "Product failure modeling method based on AltaRica language," in *Proc. Prognostics Syst. Health Manage. Conf. (PHM-Harbin)*, Jul. 2017, pp. 1–6.

[20] S. Humbert, C. Seguin, C. Castel, and J.-M. Bosc, "Deriving safety software requirements from an AltaRica system model," in *Computer Safety, Reliability, and Security*. Berlin, Germany: Springer, 2008, pp. 320–331.

[21] M. Batteux, T. Prosvirnova, A. Rauzy, and L. Kloul, "The AltaRica 3.0 project for model-based safety assessment," in *Proc. 11th IEEE Int. Conf. Ind. Informat. (INDIN)*, Jul. 2013, pp. 127–132.

[22] T. Prosvirnova and A. Rauzy, "AltaRica 3.0 project: Compile guarded transition systems into fault trees," in *Proc. Eur. Saf. Rel. Conf. (ESREL)*, Amsterdam, The Netherlands, Sep./Oct. 2013, pp. 1–8.

[23] S. Li and X. Li, "Study on generation of fault trees from AltaRica models," *Procedia Eng.*, vol. 80, pp. 140–152, Jan. 2014.

[24] X. Li and S. Li, "Graphical modeling of system failure behavior and its translating into AltaRica," *Procedia Eng.*, vol. 80, pp. 581–591, Jan. 2014.

[25] P. Bieber, C. Bougnol, C. Castel, J.-P. Christophe Kehren, S. Metge, and C. Seguin, "Safety assessment with AltaRica," in *Building the Information Society*, vol. 156. Cham, Switzerland: Springer, 2004, pp. 505–510.

[26] M. Bozzano, A. Cimatti, O. Lisagor, C. Mattarei, S. Mover, M. Roveri, and S. Tonetta, "Safety assessment of AltaRica models via symbolic model checking," *Sci. Comput. Program.*, vol. 98, pp. 464–483, Feb. 2015.

[27] [Online]. Available: https://www.apsys-airbus.com/en/digital-software-en/#SIMFIA

[28] F. Zhang and H. Dong, "Research on formal modeling and safety analysis method of head-up display system for civil aircraft based on AltaRica," in *Proc. 3rd Int. Conf. Circuits, Syst. Simulation (ICCSS)*, Jun. 2019, pp. 116–120.

[29] G. Qingfan, W. Guoqing, Z. Lihua, and Z. Ming, "Research on model based safety analysis technoloy for avionics system," *Comput. Sci.*, vol. 42, no. 3, pp. 124–143, 2015.

ZHEN LI was born in Xinghua, Taizhou, Jiangsu, China, in 1977. He studied at Nanjing Normal University, major in management. He received the bachelor's degree in computer application from Nanjing University, the M.S. degree in signal and information processing from the Jiangsu University of Science and Technology, Zhenjiang, in 2006, and the Ph.D. degree in aerospace system engineering from Beihang University, Beijing, in 2011.

From 2011 to 2014, he was a Lecturer with the School of Electronics and Information, Jiangsu University of Science and Technology, where he has been an Assistant Professor, since 2014. He is the author of more than 15 articles and holds two inventions. His research interests include reliability and system engineering, safety engineering, swarm intelligence, resilience, and software testing.

ZHENGQI JIANG was born in Suzhou, Jiangsu, China, in 1995. He received the bachelor's degree from the Huaiyin Institute of Technology, in 2018. He is currently pursuing the master's degree in electronics and communication engineering with the Jiangsu University of Science and Technology, Zhenjiang.

His research interests include system safety and reliability.

DONGSHENG WANG (Member, IEEE) was born in Yancheng, Jiangsu, China, in 1982. He received the master's degree in computer science from the Jiangsu University of Science and Technology, in 2007, and the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 2012. He is currently an Associate Professor. His primary research interests include software engineering, question answering, and natural language understanding.

ZHAOBIN WANG was born in Qiqihaer, Heilongjiang, China, in 1982. He received the master's and Ph.D. degrees from the Harbin Institute of Technology, in 2007 and 2013, respectively. He held a postdoctoral position with the Faculty of Mechanical Engineering and Automation, Zhejiang Sci-Tech University, from 2014 to 2018. He is currently an Associate Professor. His primary research interests include reliability theory and test in electronics, storage reliability, reliability, and PHM.