Verification of Dependable Architecture based on Prototype Verification System

Ling Yuan
School of Computer Science
Huazhong University of Science and Technology
Wuhan, China
cherryyuanling@gmail.com

Ping Fan*
School of Computer Science
Hubei University of science and technology
Hubei, China
fanping1028@126.com

***Abstract*—The electronic power system can be viewed as a system composed of a set of concurrently interacting subsystems to generate, transmit, and distribute electric power. The complex interaction among sub-systems makes the design of electronic power system complicated. Furthermore, in order to guarantee the safe generation and distribution of electronic power, the fault tolerant mechanisms are incorporated in the system design to satisfy high reliability requirements. As a result, the incorporation makes the design of such system more complicated. We propose a dependable electronic power system architecture, which can provide a generic framework to guide the development of electronic power system to ease the development complexity. In order to provide common idioms and patterns to the system *designers, we formally model the electronic power system architecture by using the PVS formal language. Based on the PVS model of this system architecture, we formally verify the fault tolerant properties of the system architecture by using the PVS theorem prover, which can guarantee that the system architecture can satisfy high reliability requirements.**

Keywords- System Architecture; PVS System; Fault Tolerance; Formal Modeling; Formal Verification.

I. INTRODUCTION

Electronic power system is a highly dependable distributed system, where substations, power companies, control areas, and interconnection cooperate with each other to generate, transmit, and distribute electric power safely. The functional and nonfunctional requirements make the development of such distributed systems complicated. Software architecture is identified as a critical design methodology, which can ease the complexity of the development of distributed systems [1], [2]. We propose an Electronic Power System Architecture (EPSA) to guide the development of such systems, which involves not only different kinds of concurrency but also fault tolerant mechanisms [3], [4].

Formal methods [5], [6], [7], [8] provide precise specification and rigorous verification for architecture design in virtue of well-defined semantics. Prototype Verification System (PVS) [9] is a powerful theorem prover with its highly integrated environment for writing formal specifications and developing rigorous verification. PVS, built on over years of experience at SRI in developing and using tools to support formal methods, has been successfully applied to large and complex application in both academic and industrial areas. PVS modeling language expands higher-order logic with a sophisticated type system. The interactive theorem prover of PVS offers powerful automatic reasoning techniques at low levels such as arithmetic of real numbers and sets. Users can directly control proof development at a high level by defining their own proof strategies which combine primitive PVS proof commands. These strengths of PVS are useful to verify the fault tolerant properties of electronic power system architecture.

In order to provide precise idioms and patterns to the system designers, we formally model EPSA using PVS specification language. Based on the formal model of EPSA, we utilize the theorem proving method of PVS to verify the fault tolerant properties of EPSA with high degree of automation.

The remainder of this paper is organized as follows. Section 2 describes the construction of electronic power system. Section 3 explains the architecture styles of EPSA. Section 4 illustrates the formal model of EPSA using PVS specification language. Section 5 presents the verification of fault tolerant properties of EPSA. Section 6 concludes the paper.

II. ELECTRONIC POWER SYSTEM AND PVS

A. Electronic Power System

Electronic power system is composed of substations, generating station, power companies, control areas, and interconnection. As shown in Fig. 1, the substation and generating station report the demand and supply to their parent power company respectively. The power company accepts the data, calculates and reports the surplus or deficit to its parent control area. The control area accepts power balances, calculates and reports the balance to its parent control region. The control region accepts power balance, calculates and reports the balance to its parent inter-connection. The interconnection accepts the power balances, calculates, swaps power with other interconnections, and redistributes power interchanges amongst its control regions.

The hierarchical relationship among substation, generating station, power company, and control area can be applied to the hierarchical relationship among control area, control region, and interconnection. Therefore, we focus our development of EPS on the concurrency among

* corresponding to: Ping Fan, School of Computer Science, Hubei University of science and technology, fanping1028@126.com

substation, generation station, power company and control area.

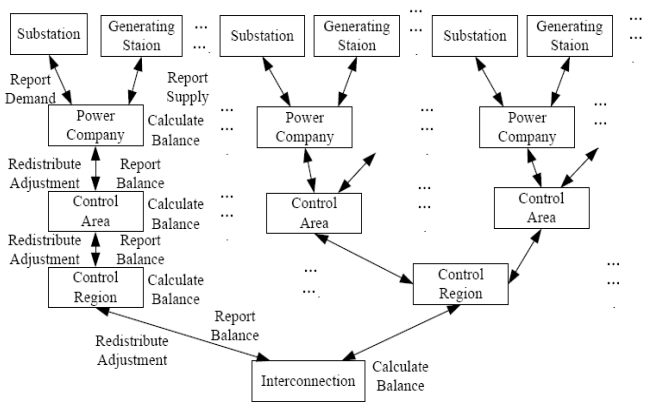

Figure 1. Construction of Electronic Power System

B. *Prototype Verification System (PVS)*

PVS is an integrated environment for formal specification and verification. In order to support modularity and reuse, specifications in PVS are logically organized into parameterized *theories*, which can be linked by *import* and *export* lists. We use a queue example to explain how to build a theory. The *queue* is defined as a *THEORY* associated with the generic parameter *Item*. Meanwhile, the *TYPE+* indicates that the type of the *Item* is uninterpreted and nonempty. Function *Join* defines receiving a new element *item*. Function *Leave* defines leaving an old element *item*.

```
Queue [Item: TYPE+]: THEORY
BEGIN
items:
TYPE=[#size: nat, elements: ARRAY[{i|i<size} -> Item]#]
itms: VAR items
nonemptyqueue?(itms): bool=(size(itms)>0)
nitms: VAR (nonemptyqueue?)
join(item,itms):items=(#size:=size(itms)+1,
   elements:=elements(itms) WITH [(size(itms)):=item]#)
leave(item, nitms): items=(#size:=size(nitms)-1, elements:=
(LAMBDA(j:{i|i<size(nitms)-1}): elements(nitms)(j+1))#)
END Queue
```

The theorem prover of PVS maintains a proof tree, and the objective is to construct a complete proof tree in which all leaves are trivially true. Each node of a proof tree is a proof goal, which is a *sequent* consisting of a sequence of formulas as *antecedents* and a sequence of formulas as *consequents*. As an example shown in Fig. 2, the *antecedent* is composed of two formula *A1* and *A2*, and the *consequent* is composed of two formulas *B1* and *B2*.

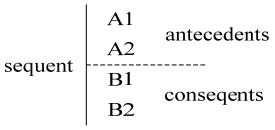

Figure 2. PVS Proof Tree Node

The intuitive interpretation of a proof goal is that the conjunction of the antecedents implies the disjunction of the consequents. Users can guide the PVS theorem prover by entering PVS proof commands which can be used to introduce lemmas, expand definitions, apply decision procedures, eliminate quantifiers, and so on.

III. DEPENDABLE EPSA

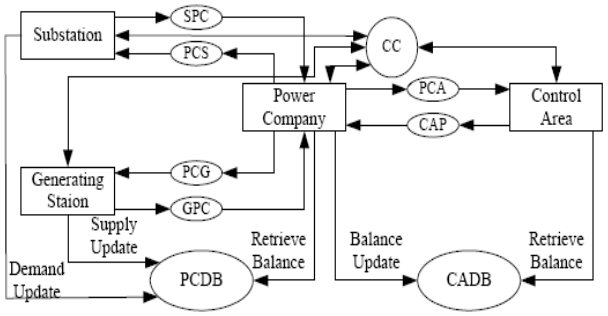

Figure 3. Dependable Electronic Power System Architecture

EPSA is proposed to guide the development of electronic power system with high reliability requirements. As shown in Fig.3, EPSA is composed of distributed components:Substation, GeneratingStation, PowerCompany, and Control-Area; share resources: PCDB, and CADB; fault tolerant component: CC; and connectors: SPC, PCS, PCG, GPC, PCA, and CAP.

The distributed component encapsulates its independent data representation and their associated primitive operations. Then several distributed components can execute concurrently and corporate with each other to cater for the same goal. The share resource is shared by all the distributed components. The connector connects the in_port of one distributed component and the out_port of another distributed component.

EPSA integrates fault tolerant techniques with functional aspects at the architecture level to satisfy high reliability requirements. Once a local exception is raised in one distributed component, the component can call the corresponding exception handler in its own exception context to cope with the exception. If this exception cannot be handled successfully, the component should signal a global exception and transfer it to the fault tolerant component. When multiple global exceptions are raised concurrently in specific distributed components, these global exceptions are passed to the fault tolerant component. The fault tolerant component resolves the received global exceptions into a universal exception and broadcasts the universal exception to related components and shared resources within a network. The informed components can deal with the exceptions based on the integrated exception handling mechanisms.

IV. FORMAL MODEL OF EPSA USING PVS

The formal model of EPSA can provide precise idioms and patterns to the system developers. The distributed components, shared resources, connectors, and fault tolerant components are all formally specified using PVS

specification language. We take the fault tolerant component CC, and distributed component Substation as a snapshot to illustrate the formal model of EPSA.

A. Formal Model of Fault Tolerant Component

The CC theory describes how the fault tolerant component in EPSA implements the coordinated error recovery mechanism when a global exception is raised in an Object or multiple global exceptions are raised concurrently in different Objects. This CC theory imports the generic-type theory and the parameterized theory queue. Note that theory queue is instantiated with type OBSTATE. The OBSTATE denotes the state of distributed component, either be normal, a local exception, a global exception, or fail.

```
CC: THEORY
BEGIN
IMPORTING GenericType, Queue[ EXCEPTION ]
except_graph: [items[EXCEPTION] -> EXCEPTION]
exception: EXCEPTION
CC: TYPE = [#exceptions: items [EXCEPTION],
                    uni_exception: EXCEPTION # ]
cc: VAR CC
emptycc: CC=(#exceptions:=empty, uni_exception:=e#)
ExceptRec(cc): CC=(#exceptions:=join (exception,
            exceptions(cc)), uni_exception:=exception#)
ExceptGraph(cc): CC=
 IF exceptions(ExceptRec(cc)) /=empty
   THEN  (#exceptions:=empty, uni_exception:=
          except_graph(exceptions(ExceptRec(cc)))# )
   ELSE emptycc
 ENDIF
```

In the CC theory, the function except_graph is declared to model how to derive a global exception from a set of global exception by using specific exception graph methodology. Function ExceptRec represents that the fault tolerant Component receives exception from the distributed components. Function ExceptGraph specifies that the fault tolerant component uses the function except_graph to set the value of the uni_exception when receiving one or more than one global exception.

B. Formal Model of Distributed Cmponent

The distributed component Substation sends the used and required electronic power data to its parent power company. When exceptions raised, the Substation can handle these exceptions.

```
Substation[SRSTATE: TYPE+]: THEORY
BEGIN
   IMPORTING CC
n_states, excepts: setof [ SRSTATE ]
tsin_ports, tsout_ports: setof [ PORT ]
dc_msg: [ PORT -> MSG ]
senddata: [ [ SRSTATE, [ PORT -> AMSG ]] ->
            [SRSTATE ] ]
except_context: [EXCEPTION -> EH]
except_handle: [EH -> SRSTATE]
Station: TYPE = [# inter_state: SRSTATE, checkpoint:
          SRSTATE, ue_rec: SIG # ]
st : VAR Station
SendData (st):Station= IF member(inter_state (st),n_states)
THEN (#inter_state:=PROJ_1(senddata (inter_state (st),
(LAMBDA p: dc_msg (p)))), checkpoint:=inter_state(st),
ue_rec:=0 #)
ExceptPropagate(st):SRSTATE=IFmember(inter_state(st),
excepts) AND ue_rec(st)=0  THEN   inter_state(st)
ccp: VAR CC
UniExceptReceive (st, ccp): Sensor=IF uni_exception=
except_graph(exceptions(ExceptRec(ccp))) THEN
(#inter_state: =uni_exception, checkpoint:=checkpoint(st),
ue_rec:=1 #)
UniExceptHandle (st): Station=
IF member(inter_state(st), excepts) AND ue_rec(st)=1
THEN (IF member(except_handle (except_context
(inter_state(st))), n_states) THEN
(#inter_state:= except_handle (except_context
(inter_state(st))), checkpoint: = inter_state(st), ue_rec:=0 #)
ELSIF
except_handle(except_context(inter_state(st)))=Fail
THEN (#inter_state:=Fail, checkpoint:=inter_state(st),
  ue_rec:=0 #)
END Subsation
```

In the *Substation* theory, the function *SendData* sends the electronic power data to the power company via connectors. When the Substation raises a global exception, the function *ExceptPropagate* is used to propagate the raised exception to the fault tolerant component. The function *UniExceptReceive* specifies when the distributed component receives a universal exception (uni_exception) from the fault tolerant component, the state of Substation (inter state is updated to uni_exception and signal ue_rec is updated to 1). Function *UniExceptHandle* changes the states of four elements in the variable *st* according to the result of exception handling with the universal exception. If the Substation can successfully deal with the universal exception by using specific exception handler, denoted as checking whether the execution result is one of stable states (n_states), the inter state of *st* would be set as a stable state, otherwise the inter state would be a *Fail* state.

V. VERIFICATION OF EPSA USING PVS

Since EPSA is used to guide the development of dependable electronic power system, it is important and necessary to rigorously analyze the fault tolerant properties of proposed EPSA. In this section, with the help of powerful theorem proving of PVS, we can mechanically verify that EPSA can satisfy the fault tolerant properties. The verification about one significant fault tolerant property is presented here to demonstrate EPSA can satisfy the high reliability requirements.

A. A Fault Tolerant Property

When the Shared Resource PCDB is attacked, a distributed component (e.g. PowerCompany) raises a global exception *PCDBAttacked*, then the other distributed components (e.g. Substation) should be informed about this exception and deal with it. This property is firstly input to the theorem prover of PVS, which is represented as the consequent of a sequent, as shown below.

```
pcdb_ pred1 :
|-------
{1} (EXISTS (pc: PowerCompany): member (inter_state (sr),
excepts) AND ue_rec(sr) = 0)  IMPLIES
   (FORALL (ss: Substation), (ccp: CC):
   inter_state (UniExceptReceive(dc, ccp)) = except_graph
      (exceptions(ExceptRec(ccp))))
Rule?: (flatten)
tsft_ pred1 :
{-1} (EXISTS (pc: PowerCompany): member
(inter_state(sr),excepts)  AND ue_rec(sr) = 0)
|-------
{1} (FORALL (ss: Substation), (ccp: CC):
   inter_state(UniExceptReceive(obj, ccp)) = except_graph
      (exceptions(ExceptRec(ccp))))
Rule?:
```

During a proof, we enter PVS proof commands after Rule?, which is prompted by the PVS prover so as to interactively verify the property. For example, proof command *flatten* converts the consequent, namely, pred1 ft into a sequent, by eliminating the disjunctive connectives (denoted by *IMPLIES* here).

B. Verification of Fault Tolerant Property

The PVS proof commands after each Rule? constitute the proof script for the property verification. As shown below, the proof script of property pcdb_pred1 starts with proof command *flatten*, followed by *skolem!* which replaces the existentially quantified variable obj in the antecedent (as prefixed by {-1}) with an constant pc!1.

```
(flatten)(skolem!)
(lemma "Propagate")(assert)(skolem!)(instantiate –1 ("pc!1"))
(assert)(prop)(hide -2)(hide -2)
(lemma "Propagate")(assert)(skolem!)(instantiate -1("coc!1"))
(assert)(prop)(hide -3)(hide -3)
(lemma "ExceptPropagate")(instantiate -1 ("pc!1"))
(replace -1 (-1 -3) rl)(hide -1)
(lemma "NonEmpty")(instantiate -1 ("ccp!1" "pc!1"))
(lemma "CCReceive")(instantiate -1 ("ccp!1"))(assert)
(lemma "ExceptGraph1")(instantiate -1 ("ccp!1" "ss!1"))
(replace -1 (-1 -2) rl)(hide -1)
(lemma "UniExcept")(instantiate -1 ("ccp!1" "dc!1"))(prop)
(lemma "ExceptGraph1")(instantiate -1 ("ccp!1" "crr!1"))
(replace -1 (-1 -2) rl)(hide -1)
(lemma "UniExcept")(instantiate -1 ("ccp!1" "crr!1"))(prop)
```

Followed *skolem!*, there are several user defined lemmas which have been proved to be true. These lemmas can induce the property verification until the verification result is true. The strategy of the proof is that since the condition of property *pcdb_pred1* is that the state of PowerCompany 1) one of the global exceptions, and 2) has not received a universal exception from the fault tolerant component, the PowerCompany should send such exception to the fault tolerant component. Furthermore, the fault tolerant component needs to receive the raised exception. When receiving an exception, the fault tolerant component should put such exception into an exception list(exceptions) by using function *ExceptRec* defined in the coordinating theory, Lemma *NonEmpty* implies that the exception list of the fault tolerant component is not empty, denoted as exceptions(ExceptRec(cc))/=empty. When the exception list is not empty, Lemma *CCReceive* is applied to induce the universal exception(uni_exception) which covers all received exceptions by using function *except_graph*, where uni_exception =except _graph(exceptions(ExceptRec(cc))). After obtaining the universal exception, the fault tolerant component should send it to the distributed component Substation. Lemma *ExceptGraph1* indicates that the Substation receives the universal exception by using the function *UniExceptReceive*. By using lemma *CCReceive*, we can get that, inter_state(UniExceptReceive(obj, ccp)) = except_graph(exceptions(ExceptRec(cc))),which is the deduction result of pcdb_pred1. By running the above proof script, the PVS theorem prover can verify the pcdb_pred1 property automatically.

Other fault tolerant properties, such as CADB is attacked, or both PCDA and CADB are attacked, can be verified using the theorem prover of PVS.

VI. CONCLUSION

In this paper, we propose an Electronic Power System Architecture (EPSA) to provide a framework to guide the development of dependable electronic power systems. The formal model of EPSA is presented to provide precise idioms and patterns to the system designers. With the help of powerful theorem prover of PVS, the fault tolerant properties of EPSA are verified with high degree of automation, which can demonstrate the proposed EPSA can satisfy high reliability requirements.

ACKNOWLEDGMENT

The authors would like to thank for the sponsor supported by National Natural Science Fund (No. 61100059).

REFERENCES

[1] M. Shaw, “The coming-of-age of software architecture research”, Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001), 2001.

[2] D.Garlan and M.Shaw, “Introduction to software arch-itecture”, IEEE Transactions on Software Engineering. 1995, 21(4): 269-274.

[3] J.C.Laprie, “Dependability basic concepts and terminology”, Dependable Computing and Fault Tolerant Systems. Springer-Verlag, 1992.

[4] M.Xie, K.L.Poh, and Y.S.Dai,“Computing system reliability: models and analysis”, Springer, 2004.

[5] G.T.Leavens and M.Sitaraman, “Foundations of component-based systems”, Cambridge University Press, 2000.

[6] G.D.Abowd, R.Allen, and D.Garlan, “Formalizing style to understand descriptions of software architecture”, ACM Transactions on Software Engineering and Methodology, 1995, vol4(4): 319-364.

[7] J.Sun and J.S.Dong, “Specifying and reasoning about generic architecture in TCOZ”,. In Proceedings of the 9th Asia-Pacific Software Engineering Conference (APSEC’02), IEEE Computer Society Press, 2002: 405-414.

[8] M.Shaw and D.Garlan, “Software architecture: perspectives on an emerging discipline”, Prentice Hall, 1996.

[9] S.Owre and N.Shankar, “The formal semantics of PVS”, Computer Science Laboratory, SRI International, Menlo Park, CA, 1997.