Collective Self-adaptive Software Architecture Specification: Understanding Uncertainty in Cyber-Physical Convergence

Hua Wang and Zhijun Zheng
School of Information and Electronic Engineering, Zhejiang University of Science and Technology, Hangzhou, China
Email: wanghua96@gmail.com, zjzheng9999@sina.com

***Abstract*—There had been past attempts at making adaptation strategies with analytic model that used the current environment information in Cyber-physical System (CPS) to keep the software architecture from deteriorating. However, the research is still in its infancy on the closely-related issue of how to take corrective self-adaptive actions to reconcile the CPS system behavior with the variability of the survival environment. In particular, architects have almost no assistance in reasoning about questions such as: How should we stage the architectural evolution to improve self-adaptation to accommodate uncertainties in CPS? In this work, the specification of software architecture is extended using CHAM (Chemical Abstract Machine) in the presence of uncertainty. The key benefits of our approach are that it leverages standard software architecture models, and quantifies behaviors within the system in terms of relevant architectural elements. Compared to the traditional model, the proposed method could arrange optimized response sequence and adjust the behaviors within the software system to be adaptive to the new situations over time in CPS.**

***Index Terms*—self-adaptive, software architecture, cyber-physical system, uncertainty, chemical abstract machine**

I. Introduction

Cyber–physical System (CPS) is the next generation of engineered systems in which computing, communication, and control technologies are tightly integrated. CPSs are integrations of computation and physical processes [1]. To satisfy emerging and ever-changing user requirements and expectations, software systems in CPS should be self-adaptive to preserve the system architecture from deterioration. Endowing such a system with an adaptive property can strengthen the survival ability of the system, which we term collective self-adaptation in the presence of cyber-physical convergence. Recent retrofitting autonomic adaptation capabilities into software-intensive systems are prevalently done in an ad-hoc fashion. The proposed approaches should offer a compositional and systematic model using software architecture, which guides the topology of the constituent computational elements (such as components and connectors) of the system. The behavior of the whole system is the synergistic effect of these interacting elements. Problems occur when a component is used in an unanticipated way and/or components become a constituent of a larger complex system. The dynamic context where the system is running can cause unexpected behavior resulting from a combination of the unanticipated use of components and interactions between components, which stems from the fact that CPSs are inherently dynamical and uncertain. Dealing with context uncertainty and predicting complex behavior in social networking systems are challenged [2].

To this end, a recent common approach to monitor and adapt system behavior at runtime is to decouple self-adaptive mechanisms from the target system by considering the intrinsic nature of CPS. The non-invasive manners have the main advantage of that style-specific adaptation knowledge has not been "hardwired" into the target system, i.e., realizing separation of concerns. However, a variety of sources of uncertainty are introduced while utilizing these separate control units [3]. These uncertainties may include bringing the system into unknown system states, the random user requests for services, the unanticipated effects of performing adaptation policies, to name a few. There is a substantial body of discussion literatures on handling uncertainty of software behaviors, such as [4-6], etc.

There have also been past attempts at making adaptation strategies with analytic model that used current environment information, such as [7-9]. However, on the closely-related issue of how to take corrective self-adaptive actions to reconcile the CPS system behavior with the variability of the survival environment, the research is still in its infancy. An adaptation agent was employed to collect massive distributed data from widespread sensors, and then constructed and executed adaptation plans [10]. Evolutionary computation was applied to mitigate uncertainty in the case of high-assurance adaptive software, especially in CPS [11]. Jeffrey and colleagues put forward an architecture evolution means among potential candidate evolution paths [12].

This paper is based on "Self-adaptive Method Based on Software Architecture by Inspecting Uncertainty," by Hua Wang and Zhijun Zheng, which appeared in the Proceedings of the 2010 International Conference on Artificial Intelligence and Computational Intelligence (AICI 2010), Sanya, China, October 23-24, 2010. © 2010 IEEE.
Manuscript received March 23, 2013; revised October 13, 2013; accepted October 30, 2013.
Corresponding author: Hua Wang, wanghua96@gmail.com.

doi:10.4304/jcp.9.4.802-811

The increasing shift toward unanticipated adaptation calls for revisiting current reactive self-adaptive approaches to anticipate situations on the recent future and adjust the behaviors within the system under consideration in CPS. The traditional method puts itself to some extent that the adaptation process can only be preplanned and defined in a limited space. The need to anticipate the aforementioned adaptation tasks has driven the development of predictive self-adaptation. The unanticipated inherence and complexity of upcoming services and applications make proactive self-adaptation essential. In particular, architects have almost no assistance in reasoning about questions such as: How should we stage the architectural evolution to improve self-adaptation to accommodate uncertainties in CPS? This presents the main technical challenge, i.e., the proposed model can learn from the environment of cyber-physical convergence. In this paper, we propose a collective self-adaptive method based on the software architecture framework in the presence of uncertainty attributes in CPS. Compared to the traditional model, the proposed method could arrange optimized response sequence and adjust the behaviors within the software system to be adaptive to the new situations over time.

In our previous work [13], Hidden Markov Model was employed to model uncertainty and learn from history behavior of target system, and then anticipatory actions were issued on the target system. In this paper, Chemical Abstract Machine (CHAM) [14] is employed as a tool to specify software architecture. CHAM is good as the description of the dynamics and parallels of systems. The semantic of the dynamic operation of software architecture is described and analyzed by formal methods of CHAM. Beneficial extension of dynamic features of software architecture is presented by inspecting uncertainty. Formal reasoning of specification of self-adaptive software architecture is realized. The focus of the software architecture specification by CHAM is how to describe the dynamic interaction behavior of components comprising of software architecture. In this work, the proactive ability is added by underlying anticipatory self-adaptation. Our approach is novel as it leverages standard software architecture models, and quantifies behaviors within the system in terms of relevant architectural elements. The new contribution of this paper is to apply self-adaptive software architecture in CPS field base on our previous paper [15].We extended the model check of CHAM and specified *Safety* and *Liveness* properties of CHAM by performing two different algorithms. Also, the new experiment was designed to show the efficiency of the proposed method and more results were obtained.

The remainder of this paper is organized as follows: Section II introduces the specification of collective self-adaptive software architecture based on CHAM. We discuss the uncertainty in our context in Section III and model check of the specification of collective self-adaptive software architecture in Section IV. In Section V, the improved Viterbi-I algorithm is presented. In Section VI, we report on a case study on the self-adaptive method based on software architecture in CPS followed by discussion of the remaining challenges and possible directions in future research.

II. Specification of Collective Self-adaptive Software Architecture

From the formal semantics of CHAM, a conclusion can be reached that the reaction rule is an ideal tool to reflect the dynamic behaviors within the target system. Components of software architecture could be considered as molecules to represent the inner status of CHAM. Relevant molecules can interact with each other by different statuses. The specification of architectural CHAM includes architecture (molecules) grammar of static components, the grammar of the initial state of software architecture and the syntax of the system in response to the dynamic development phase according to the grammar rules. The initial status set is the subset of relevant molecules configured by grammar. The initial static configuration of software architecture relates to the initial status set of CHAM. The transform rules applied to initialize the solution set define how the system develops dynamically. The initial solution is relevant to the static configuration of the targeted system. Each molecule of the initial solution corresponds to the initial status of each architectural element. The transform rules define how software architecture evolves dynamically from the initial configuration.

A. Specification of Collective Software Architecture based on CHAM

Definition 1. The specification of collective self-adaptive software architecture based on CHAM is defined as

$$Cham = \left(ComM, ConnM, R, S, s_0, S_f, Path \right)$$

where *ComM* is the grammar set of component molecules of software architecture; *ConnM* is the grammar set of connector molecules of software architecture; $R \in WP$ is the reaction rules of a molecule set; S is the solution set; $s_0 \in S$ is the initial solution set of software architecture; $S_f \in S$ is the terminal solution set of software architecture; and *Path* is the path of status change of software architecture.

The component molecule set reflects component ontology while connector molecule set represents the role set of connectors. Components, connectors and constraints are basic elements of software architecture. Port and Role are other basic elements of software architecture. The interface is the only bridge for the interaction of components with other elements of software architecture. The port of components is the communication point for components and software survival environment. The interface is composed of a group of ports. Connectors have interfaces that are composed of a group of roles. The role represents the actor participating the activity. In this sense, ports, interfaces and roles imply interaction and constraint of architectural elements, which belongs to grammar layer. On the other hand, *Path* demonstrates how connectors use

Weaving Policy (*WP*) to instruct the dynamic behaviors from the initial solution to terminal solution, and this is belonged to semantic layer. *WP* is built by extensible *ECA* rules based on our previous works [16].

Since *Cham* considers the initial status of the target system as the initial solution, the system is a single flow launched by an initial solution [17]. However, molecules, membranes and sub solutions concurrently react each other probably, as a result, a special connection symbol " || " is used to handle the concurrent reaction. The elements connected by " || " can react concurrently, but there is no intersection. Consequently, the molecule algebra Σ_{Cham} is redefined as follows:

Definition 2. Σ_{Cham} is defined as:

$$Molecule ::= Com \mid Connector \mid Connector \Diamond M \mid M \Diamond Connector \mid M \Diamond M \mid M \parallel M \mid Cmd \mid Connection$$

$$Com ::= ID : Type$$

$$Type ::= AC \mid PC$$

$$Adwc ::= Role \mid DUP(number) : Molecule / ID : Mediator \Theta WP$$

$$Role ::= RoleName / Modecule \mid RoleName = \mid RoleName = ID$$

$$Cmd ::= bc(Com) \mid ec(ID) \mid rc(Com) \mid ac(Connector) \mid ec(ID) \mid rc(Connector)$$

$$Connector ::= RoleName - RoleName$$

where *Com* is components identified by *ID*; *AC* is Aspect Component; *PC* is Primary Component; *Connector* is composed of *Mediator* and *WP* (*Mediator* reasons about *WP* by the symbol ' Θ '); the command of *bc* (build a component), *ec* (erase a component), *uc* (unify components) and *sc* (split a component) can be used to realize the static evolution of software architecture. Component encapsulating crosscutting concern (such as security, requirement and distribution) is defined as *AC* while component performing a functional operation is defined as *PC*. *Connector* coordinates the *AC* and *PC*.

Definition 3. Cluster component molecule is represented by component molecule and is encapsulated by a membrane, defined as follows:

$$ClusterCom ::= \left\{ \left| Com_1, Com_2, \cdots, Com_k \right| \right\}$$

Definition 4. The initial solution of *Cham* is the initial static configuration of software architecture and the starting point of the evolution of software architecture. Suppose that the initial sub solution is $s_{01}, \cdots, s_{0n}$, then the initial solution of software architecture is:

$$s_0 = s_{01} \parallel \cdots \parallel s_{0n}, s_i = \left\{ \left| m_i', m_i', \cdots, m_l' \right| \right\}$$

B. Reaction Rules of Cham

Definition 5. Reaction rules are defined as follows:

$$T_1 \equiv m_1 \parallel \cdots \parallel m_l \to m_1, \cdots, m_l$$

$$T_2 \equiv bc(ac : AC), pc : PC, pc' : PC \xrightarrow{WP} ac : AC, pc : PC, pc' : PC, pc \Diamond ac \Diamond pc'$$

$$T_3 \equiv bc(pc : PC), ac : AC, pc' : PC \xrightarrow{WP} pc : PC, ac : AC, pc' : PC, pcpc \Diamond ac \Diamond pc'$$

$$T_4 \equiv ec(pc : PC), pc \Diamond ac \Diamond pc', ac : AC, pc' : PC \xrightarrow{WP} ac \Diamond pc'$$

$$T_5 \equiv ec(ac : AC), pc \Diamond ac \Diamond pc', pc : PC, ac' : AC, pc' : PC \xrightarrow{WP} pc \Diamond ac' \Diamond pc'$$

$$T_6 \equiv rc(pc' : PC), pc_1 \Diamond ac \Diamond pc_2, pc_1 : PC, ac : AC, pc_2 : PC \xrightarrow{WP} pc' \Diamond ac \Diamond pc_2$$

$$T_7 \equiv rc(ac' : AC), pc_1 \Diamond ac \Diamond pc_2, pc_1 : PC, ac : AC, pc_2 : PC \xrightarrow{WP} pc' \Diamond ac' \Diamond pc_2$$

$$T_8 \equiv uc, pc_1 \Diamond ac \Diamond pc_2, pc_1 : PC, ac : AC, pc_2 : PC, pc' : PC \xrightarrow{WP} pc'' \Diamond ac \Diamond pc_2$$

$$T_9 \equiv sc, pc_1 \Diamond ac \Diamond pc_2, pc_1 : PC, ac : AC, pc_2 : PC \xrightarrow{WP} (pc_{11} \parallel pc_{12}) \Diamond ac \Diamond pc_2$$

where operator ' ◊ ' represents the connection based on roles for components and connectors. Architectural elements can be modeled by a group of *PC*s, *AC*s, roles and *WP*s. If a certain solution does not permit the reaction of specified molecules, membrane and airlock reaction rules are used.

Definition 6. Membrane reaction rules:

$$T_{10} \equiv \left\{ \left| ac \Diamond pc_1 \lhd \left\{ \left| pc_2, \cdots, pc_k \right| \right\} \right| \right\} \xleftrightarrow{WP} ac \Diamond \left\{ \left| ac \Diamond pc_1 \lhd \left\{ \left| pc_2, \cdots, pc_k \right| \right\} \right| \right\}$$

where $ac \in AC; pc_1, \cdots, pc_k \in PC$, and $\left| pc_2, \cdots, pc_k \right|$ represents membrane and its reaction rules are similar to solution. Molecules can be picked up from solution by ' ◁ '. If a reaction does not reflect the inner status of membrane, the following reaction rule is required.

Definition 7. Airlock reaction rule:

$$T_{11} \equiv \left\{ \left| ac \Diamond pc_1 \lhd \left\{ \left| pc_2, \cdots, pc_k \right| \right\} \right| \right\} \Diamond ac \xrightarrow{WP} \left\{ \left| pc_1 \Diamond ac \lhd \left\{ \left| pc_2, \cdots, pc_k \right| \right\} \right| \right\}$$

where $ac \in AC; pc_1, \cdots, pc_k \in PC$.

III. Uncertainty in Cham

A. Source of Uncertainty

As mentioned before, randomness inherent in the dynamic context and the noisy nature of the predictive model leads to uncertainty in CPS. The model must explicitly dispose the uncertainty. Some research results constitute a contribution to deal with the uncertainty, such as Poladian [18] proposed an approach to self-adaptation that leverages predictions of future resource availability. However, unlike dealing with uncertainty in resource predictions, we present uncertainty as probability distributions, i.e., consider the uncertainty of user requests for services as a stochastic process. The arrival of requests into a service and the transition from $Request_i$ to $Request_j$ is uncertain, and may be represented as a *Poisson* distribution and transition probability matrix with some parameters, respectively.

B. Uncertainty Analysis

Uncertainty is presented by probability distribution to analyze and analog software architecture. For example, the arrival-rate of requests is considered as *a Poisson distribution* with certain mean and variance. The accurate analysis of distribution helps to analyze the dynamic behavior within the system under consideration meaningfully. A particular operator *AssDis* (Assign Distribution) is used to capture a probability distribution. The operator can be used as a command in-built. Software architect uses the operator to specify the uncertainty of software survival environment.

Definition 8. Uncertainty *Cham*:

$$Cmd' ::= Cmd \mid AssDis$$

$$AssDis ::= pc : PC \otimes \left(dt : DistributionType \right)$$

$$DistributionType ::= \left\{ Normal, Exponential, Poisson, ... \right\} \bullet \left[\begin{array}{l} prop_1 = val_1, \\ prop_2 = val_2, ... \end{array} \right]$$

where $\otimes$ represents the type of probability distribution of components in software survival environment, such as *normal* distribution, *exponential* distribution and *Poisson* distribution. The distribution attributes can be expressed by '•'. For example, an online health evaluation system makes an assessment of the diabetes risk of a patient. The assessment component is *DiabetesPC*. Suppose the requests of assessment of the diabetes risk satisfy *Poisson* distribution, the probability feature can be added into *Cham* specification, expressed by *AssDis* as follows:

$$AssDis ::= DiabetesPC \otimes \left(Poisson \bullet \lambda = 0.23 \right)$$

By using the extensive probability distribution, *Cham* can employee some standard probability technique to evaluate the effect of uncertainty on software architecture and then self-adaptive actions can be performed. Uncertainty lies in the stochastic nature within the context. *Cham* focuses on request of users to think about self-adaptive actions. Self-adaptive model considers uncertainty as a probability distribution, that is to say, requests of users are regarded as a stochastic process. The arrival of requests is uncertain and also the transmission of requests (from $Request_i$ to $Request_i$) is also uncertain. Accordingly, history request information and relevant probability distribution are speculated. The transition probability matrix can be used to represent the transition of requests.

Our proposed method provides with a user-defined probability property for designers to specify the new probability type that captures different distribution characteristics in different domains. A *Poisson* distribution, for example, can be specified to depict the arrival rate of user requests as illustrated in Figure 1. According to the definition of the *Poisson* distribution type in *Cham*, model designer can use the self-definition type to specify certain desired probability characteristics in different domains.

```
Distribution Type PoissonDistribtion{@float : lambda;
}
(success)->do [sa_action];
```

Figure 1. Specifying poisson distribution type in *Cham*.

Then, uncertainty is inserted into the specification of software architecture. Software architect can describe statistical features of system behavior employing probability distribution to specify architectural behavior in a wide range. All consequent research can be done based on the mathematical analysis model. The uncertain evolution process of software architecture can be depicted by this way. After that, the future features of the target system can be conferred. The self-adaptation is a specific nature of the behavior of software architecture. The driving of self-adaptation of software architecture is the survival environment in CPS. On the other hand, the self-adaptation of the software-intensive system is closely relative to software architecture. Consequently, the uncertainty of software survival environment and self-adaptability of software architecture are the two base point of our work. By analyzing uncertainty of software survival environment, we can design self-adaptive software architecture to induct the evolution within the system and realize the self-adaptation to survival environment.

IV. Model Check of Cham

A. Export Specification of Software Architecuture

The advantage of *Cham* model is to effectively describe dynamic evolution of software architecture. The dynamic features of *Cham* model can be described by LTS (Labeled Transition System) [19]. *Cham* has no the dedicated mechanism to control reaction rules at every time because it is possible for *Cham* to apply several rules at the same time. A certain rule is selected nondeterministically among these rules. The formalism of *Cham* allows to verify and validate multiple properties by employing different analysis tools and approaches. Especially, LTS is deduced from *Cham* using its operationabilities and further reasoning is feasible in LTS. Altruistic lock [20] is used to export the status tree of LTS from *Cham* model.

Considering the long transaction for the reaction of molecules resulting in postponing the reaction of other molecules, altruistic lock allows molecules of long transaction to release relevant locks once these molecules need no more data to permit other molecules to react. The export algorithm uses generalized list to describe the data structure for reaction rules and solution. The altruistic lock protocol ensures the parallel reaction to generate the right results. The protocol maintains two transition rule sets $L(c)$ and $D(c)$, where c is a certain molecule, defined as follows.

Definition 9. $L(c)$ means transition rule sets performing *a Lock operation* for c. $D(c)$ means transition rule sets performing *Donate* operation.

And the protocol maintains another two transition rule sets, $W(l)$ and $P(l)$ defined as follows.

Definition 10. $W(l)$ denotes transition rule sets of which donate molecule sets include the total access sets of l. $P(l)$ denotes molecules required by l are locked.

When solutions are initial, there is $P(l)= L(c)= D(c)= \varnothing$. The *Lock*, *Unlock* and *Denote* algorithm is illustrated in Fig. 2.

Algorithm1: *Lock*
```
Lock(c,l){
  pl=P(l) ∪ L(c);
  wd=W(l) ∩ D(c);
  if(pl ⊆ wd) {
    L(c)=L(c) ∪{l};
    P(l)=pl;
    W(l)=wd;
    return true; //lock success
  } else {
    return false; //lock fail
  }
}
```

Algorithm2: *unlock*
```
Unlock(c,l){
  L(c)=L(c)-{l};
  D(c)=D(c)-{l};
  for each r ∈ L(c) {
    W(r)=W(r)-{l};
    P(r)=P(r)-{l};
  }
  return true; //unlock success
}
```

Algorithm3: *Donate*
```
Donate(c,l){
  if(l ∈ L(c)) {
    D(c)=D(c) ∪{l};
    return true;//donate success
  } else {
    return false; //donate fail
  }
}
```

Figure 2. *Lock*, *Unlock* and *Denote* algorithms.

In *Lock* algorithm, *pl* denotes the transition rule sets where *l* only follows up and *wd* denotes the transition rule sets which follows up. Thus, the transition rules with altruistic lock can entirely follow up other transition rules. The export algorithm for *Cham* model is designed illustrated in Figure 3.

Algorithm4: *ChamPSA* generates LTS
```
1  TRACE=NULL; //initialization
2  Si=initial solution; //initial solution
3  Sf=final solution; //terminal solution
4  while(true){
5    if(Si=Sf){
6      terminate; //If the initial solution is equal with terminal one, end
7      break;
8    }else{
9      Match(ci,cj,Tk);
10     if (Tk!=NULL){
11       if(Lock(ci,Tk))
12         if(!Lock(cj,Tk)) Donate(cj,Tk); //donate cj
13         else{
14           TRACE.add(ci,cj,Tk);
15           generate new solution Si; //generate new solution
16           construct reaction tree; }//construct reaction tree
17     }
18     if(TRACE=null) continue;
19   }
```

Figure 3. Export algorithm for *Cham* model.

The first line of the algorithm ***TRACE*** is used to follow the track of every pair of c_i, c_i and the reaction rule T_k. The ninth line denotes that molecule c_i, c_j and reaction rule T_k are matched by $Match(c_i,c_j,T_k)$. The 19th line judges if molecule c_i and reaction rule T_k are locked. And furthermore, if molecule c_i is not locked, molecule c_i, c_j and reaction rule T_k are added into ***TRACE*** and repeats it.

B. Specify Safety and Liveness Properties of Cham using LTS

Modal mu-calculus has excellent logic description and can be used to formulate the safety and liveness properties of concurrent systems. As a very expressive propositional temporal logic, modal mu-calculus is a modal logic with external fixed points employed to specify safety and liveness properties of *Cham* represented by as the resulting LTS. The greatest fixed point operator of mu-calculus denotes the safety property of *Cham* suggesting that no unanticipated results are brought about whether the system has the safety property, that is, the safety property excludes a group of bad characteristics. While the least fixed point operator indicates a group of liveness attributes and specifies a special status implementing good characteristics.

The model check of LTS is performed by modal mu-calculus. The key idea is to describe the behaviors of *Cham* using LTS and the system properties using logic formula *mmf* in modal mu-calculus, respectively. As a result, whether the system holds the promising characteristics or not is converted into a mathematical problem, and that is whether *lts* is a model of the formula *mmf* or not by defining $lts \mapsto mmf$, which is interpreted by LTS as follows.

$$[\![Z]\!]e = e(Z)$$
$$[\![\ell_1 \cup \ell_2]\!]e = [\![\ell_1]\!]e \cup [\![\ell_2]\!]e$$
$$[\![\ell_1 \cap \ell_2]\!]e = [\![\ell_1]\!]e \cap [\![\ell_2]\!]e$$
$$[\![\langle k \rangle \ell]\!]e = \left\{ s \mid \exists s', s \xrightarrow{k} s' \cap s' \in [\![\ell]\!]e \right\}$$
$$[\![[k]\ell]\!]e = \left\{ s \mid \forall s', s \xrightarrow{k} s' \Rightarrow s' \in [\![\ell]\!]e \right\}$$

where e is an environment and defined as $Var \to 2^S$ (*Var* is a definite variable set and *S* the state set of LTS), and $k \in K$ is an action specified in the self-adaptive language in our previous work [15]. We call S_j is the derivative of S_i on the condition *l* iff

$$s_i \xrightarrow{l} s_j$$

The mutually recursive equational block [21] is employed for the purpose of compensating for the defect that *mmf* can only describe the behavior of a limited part of the system attributes. An equational block has one of two forms-$min\{E\}$ or $max\{E\}$, where E is a list of equations as follows.

$$Z_1 = \ell_1$$
$$\vdots$$
$$Z_n = \ell_n$$

where ℓ_i is a basic formula, and Z_i is different from each other. Let $Z[1...n]$ and $C[1...n]$ be bit array and counter array, respectively, to store the information during the process of checking models. $s.Z[i]$ is true if s belongs to the set of propositional variables Z_i. If b is a *max* block, where $b \in B$ and B are the set of mutually recursive equational blocks, then the following possibilities exist for $C[i]$.

1) If $Z_i = Z_j \cup Z_k$ is an equation and belongs to B, $s.C[i]$ counts all disjunctive terms (*true* in terms of s) appearing on the right-hand side to the equation.
2) If $Z_i = \langle k \rangle Z_j$ is in B, $s.C[i]$ counts l derivative of s associated with Z_j.
3) Others, $C[i]$ is not used.

In the same way, if b is a *min* block, then the following possibilities exist for $C[i]$.

1) If $Z_i = Z_j \cap Z_k$ is an equation and belongs to B, $s.C[i]$ counts all conjunctive items (*false* in terms of s) appearing on the right-hand side to the equation.
2) If $Z_i = \langle k \rangle Z_j$ is in B, $s.C[i]$ counts l derivative of s being not associated with Z_j.
3) Others, $C[i]$ is not used.

The block processing about *max* is organized in algorithms as follows.

Algorithm5：*Max Block Processing*

```
1  Bmax=initial max block; //initial max block
2  SVP[m]=initial state variable pairs;
3  if (Zj=Zi∪Zk) or (Zj=Zk∪Zi) {//beginning of the first case
4   for (each Zi){
5    if(Zj is the left-hand side of Bmax){
6       s.C[j]--;
7    }
8    if (s.C[j]=0){//no disjunctive term meets s
9       delete s from Sz;
10      s.Z[j]=false;
11      SVP[m++]=<s,Zj>;
12   }
13  }
14 }//end of the first case
15 if (Zj=Zi∩Zk) or (Zj=Zk∩Zi) {//beginning of the second case
16  for (each Zi){
17   if(Zj is the left-hand side of Bmax) and (s.Z[j]=true) {
18      s.Z[j]=false;
19      SVP[m++]=<s,Zj>;
20   }
21  }
22 }//end of the second case
23 if (Zj=<k>Zi) {//beginning of the third case
24  for (each Zi){
25   if (Zj is the left-hand side of Bmax){
26     if (s' is s derivation on l)
27       for (each counter) C[j]--;
28   }
29   if (C[j]=0){//s' has no l derivation meeting Zi
30      s'.Z[j]=false;
31      SVP[m++]=<s',Zj>;
32    }
33  }
34 }//end of the third case
35 if (Zj=[k]Zi) {//beginning of the fourth case
36  for (each state s'){
37   if (Zj=true) and (Zj is the left-hand side of Bmax){
38    if (s' is s derivation on l){
39      Z[j]=false;
40      SVP[m++]=<s',Zj>;
41    }
42  }
43 }//end of the fourth case
```

Figure 4. Algorithm for *Max* block processing.

Actually, the algorithm for *max* block processing continuously deletes <s, Z_i> from the list $SVP[n]$ untill $SVP[n]$ is empty as illustrated in Figure 4. At this point, the bit vector for each state relevant to the *max* block has the final value of fixed-point. The *max* block excludes the bad characteristics and expresses the *safety* property.

In the same way, the algorithm for *min* block processing is illustrated in Figure 5, which continuously deletes <s, Z_i> from the list $SVP[n]$ untill $SVP[n]$ is empty. At this point, the bit vector for each state relevant to the *min* block has the final value of fixed-point. Whether a certain state meets the given formula or not can be judged by inspecting the final bit vector, that is, if <s, Z_i> is equal 1, then Z_i meets s; otherwise, if <s, Z_i> is equal 0, the state s has no path meeting K.

Algorithm6：*Min Block Processing*

```
1  Bmin=initial max block; //initial min block
2  SVP[m]=initial state variable pairs;
3  if (Zj=Zi∪Zk) or (Zj=Zk∪Zi) {//beginning of the first case
4   for (each Zi){
5    if(Zj is the left-hand side of Bmin){
6       s.C[j]--;
7    }
8    if (s.C[j]=0){//all righ-hand disjunctive terms meet s
9       add s into Sz;
10      s.Z[j]=true;
11      SVP[m++]=<s,Zj>;
12   }
13  }
14 }//end of the first case
15 if (Zj=Zi∩Zk) or (Zj=Zk∩Zi) {//beginning of the second case
16  for (each Zi){
17   if(Zj is the left-hand side of Bmin) and (s.Z[j]=false) {
18      s.Z[j]=true;
19      SVP[m++]=<s,Zj>;
20   }
21  }
22 }//end of the second case
23 if (Zj=<k>Zi) {//beginning of the third case
24  for (each Zi){
25   if (Zj is the left-hand side of Bmin){
26     if (s' is s derivation on l)
27       for (each counter) C[j]--;
28   }
29   if (C[j]=0){//s' has no l derivation meeting Zi
30      s'.Z[j]=true;
31      SVP[m++]=<s',Zj>;
32    }
33  }
34 }//end of the third case
35 if (Zj=[k]Zi) {//beginning of the fourth case
36  for (each state s'){
37   if (Zj=false) and (Zj is the left-hand side of Bmin){
38    if (s' is s derivation on l){
39      Z[j]=true;
40      SVP[m++]=<s',Zj>;
41    }
42  }
43 }//end of the fourth case
```

Figure 5. Algorithm for *Min* block processing.

Actually, the *max* block processing results expresses the *liveness* property because good characteristics are included.

V. HMM Model and Viterbi-I Algorithm

In previous work [13], we address the situations of the evolution of software and focus on dynamic self-adaptation at runtime using the idea of HMM (Hidden Markov Model). We classify the modes of requested services, analyze the statistical properties of requests, and then use an XML file describing the context at runtime. The model treats every *Request Type* as a state and the invocation number of *Components* as the observation sequence. Based on this analysis, we model this environment and schedule clients' requests to respond to clients in a more efficient and rapid fashion. However, the classic Viterbi algorithm requires much multiplication for p ($p \in [0,1]$), resulting in the loss of precision and the overflow. This will lead to a reduced performance of the algorithm. The improved Viterbi-I algorithm uses *LOG* function for p. The multiplication of the probability values of p can be converted to the addition of $\log_p$. The observation value probability is sorted given certain request sequence. It is magnificent for improvement of Viterbi algorithm [22] because there is no need to search full status sequence. Suppose $q_t(l)$ denotes the optimized request path excluding l observation values.

$$q_t(l) = \arg\max_{q_t}\left(\prod_{t=2}^{t} \log p(q_t \mid q_{t-1}) + \chi_l(O_t \mid q_t)\right)$$

where $\chi_l(O_t \mid q_t) = \prod_{i=l+1}^{T} \log p(o_t \mid q_t)$.

At time t, $\delta_t(i)$ denotes the max probability generating observatthe pathsequence $O = O_1, O_2, \cdots O_T$ along with path $R_1, R_2, \cdots, R_t$, i.e.,

$$\delta(i) = \max_{q_1, q_2, \ldots, q_{i-1}} P(q_1, q_2, \ldots, q_t, q_t = \theta_i, O_1, O_2, \ldots, O_t / \lambda)$$

The algorithm of the optimized request path is as follows:

(1) **Initialization**:

$$\delta(i,0) = \log \pi_i + \log b_i(o_1), 1 \le i \le N$$

$$\varphi(i) = 0, 1 \le i \le N$$

(2) **Recursion**:

$$for\ 2 \le t \le T\ and\ 1 \le j \le N,$$

$$\delta_t(j) = \max_{1 \le i \le N}[\delta_{t-1}(i, l-1) + \log a_{ij}] \log b_j(O_t)$$

$$\varphi_t(j) = \arg\max_{1 \le i \le N}[\delta_{t-1}(i, l-1) + \log a_{ij}], 2 \le t \le T, 1 \le j \le N$$

(3) **Termination**:

$$P^* = \max_{1 \le i \le N}[\delta_T(i,l)]$$

$$q_T^* = \arg\max_{1 \le i \le N}[\delta_T(i,l)]$$

(4) **Solution of request sequence**:

$$for\ t = T-1, T-2, \cdots, 1$$

$$q_t^* = \varphi_{t+1}(q_{t+1}^*)$$

VI. Case Study

To demonstrate the usefulness of our approach, we show how it can be applied to a motivating industry project to illustrate the proposed modeling method, providing real-time *Pervasive Health Management Director* (*PHmD*) system based on the medical health knowledge database by analyzing health data collected from remote wireless health devices, such as Blood Pressure Monitor, Electrocardiogram Measurement Instrument, Blood-glucose Meter and so on. A possible scenario is the realtime physiological signal is relayed to *PHmD* while the jogger is running with the wearable textiles or devices. *PHmD* accommodates for the health evaluation service based on QoS (Quality of Service) requirements subscribed by different customers who receive relevant services to monitor and improve their health level before potential diseases occur, as illustrated in Figure 6.

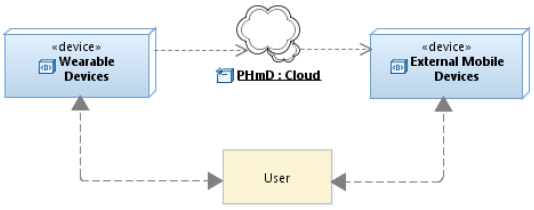

Figure 6. *PHmD* computing environment for fitness application.

PHmD will ultimately lead in the application of low-cost and innovative technological systems to daily support and assist patients, to prevent and control the ongoing of the pathology, to adjust drug therapies, and to avoid hospitalization. In particular, measurement of body movement and physical activity by inertial sensors has disclosed a wide variety of applications in favor of monitoring of bio-signals (in particular, heart rate, respiration, and gait).

An authorized healthcare professional will be able to access the health information for a customer in real-times, send health promoting messages and actuate implanted drug delivery *in situ*. The fitness application of *PHmD* computing environment uses facilities to identify disease processes or to detect adverse health-related events. The services provided by the application measure the continuous behavior of these systems, rather than their average value over some time period, to detect the dynamics.

A. Software Architecture Specification Model of PHmD

As an illustrative motivating example, let us consider the following requirement scenario for our *PHmD*. As a part of *PHmD* system requirements, *Jogging Monitor* specifies the *Device* (*Wearable Textiles*) communicating with the *Node* (*Health App.*) hosted in wireless cellphones, shown in Table I according to *IEEE* Recommended Practice for Software Requirements Specifications (SRS).

TABLE I.
Description of Jogging Monitor Using SRS

Priority	Scenario Description	Stimulus/Response Sequences
High	Monitor the vital sign of the jogger, the enormity, if	1. While jogging, the wearable device is on.
		2. The data of *Blood Pressure*, *Body Temperature*, *Respiratory Frequency*

Priority	Scenario Description	Stimulus/Response Sequences
	any, will be informed to the jogger.	and *Oxygen Consumption* are measured in the wearable device every 20 seconds.
		3. And these Vital Signs are to be transferred to the hosting Mobile App.
		4. After analysis, an abnormality, if any, will be sent to the jogger.
		5. The jogger takes actions according to the analysis report.

And all interfaces of the interactive dynamics of *PHmD* are summarized in Table II as follows.

Table II.
INTERFACES OF INTERACTION DYNAMICS OF *PHMD*

Source Component (Instance)	Sink Component (Instance)	Interface
UserInterface (ui)	Sensor (sensor)	interface_ui_sensor
Sensor (sensor)	Actuator (actuator)	interface_sensor_actuator
Actuator (actuator)	UserInterface (ui)	interface_actuator_ui
WirelessBodyEreaNetwork (wben)	MobileDevice (md)	interface_wben_md
CyberInfrastructure (ci)	WirelessBodyEreaNetwork (wben)	interface_ci_wben

Let us think about, for definiteness and without loss of generality, that the software architecture model is given by only considering two components as illustrated in Figure 7.

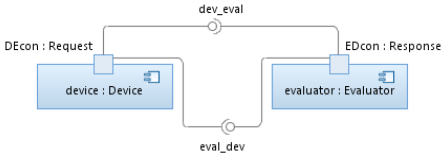

Figure 7. Software architecture for *PHmD* (partial).

The components are ***Device*** and ***Evaluator***. The connectors are ***dev_eval*** and ***eval_dev*** send physiological signals to ***Evaluator*** and response evaluation result to ***Device***, respectively. When ***Device*** sends a request message to ***dev_eval*** and waits for an evaluation result. ***dev_eval*** and ***eval_dev*** are assigned as actors, connecting ***Device*** and ***Evaluator***. After ***dev_eval*** receives a request, it reports an event to ***Evaluator*** and waits for a sign of acknowledgment from ***Evaluator***. ***Evaluator*** performs the corresponding operation according to forward message. The *Cham* specification is composed of molecule algebra Σ_{Cham}, reaction rules $T_1,...,T_7$ and initial solution s_0. Σ_{Cham} is specified as follows.

$$Molecule ::= Com \mid Connection \mid Data$$
$$Com ::= \textbf{\textit{Device}} : PC \mid \textbf{\textit{dev\_eval}} : PC \mid \textbf{\textit{eval\_dev}} : PC \mid \textbf{\textit{Evaluator}} : PC$$
$$Connection ::= i(Role) \mid o(Role)$$
$$Role ::= DEcon \mid EDcon \mid dev\_eval \mid eval\_dev$$

where, ***dev_eval*** and ***eval_dev*** are considered as special connectors. *Role* is used to denote communication among components specifying message types as different roles. *i*(*Role*) denotes an import role and *o*(*Role*) is an export role. Components of *PHmD* in the scenario are ***Device*** and ***Evaluator***, and connectors are ***dev_eval*** and ***eval_dev***. The initial solution and reaction rules are described as follows.

$$s_0 = \textbf{\textit{Device}} \Diamond o(DEcon) \Diamond i(dev\_eval), \; i(DEcon) \Diamond o(dev\_eval) \Diamond i(EDcon) \Diamond o(eval\_dev) \Diamond dev\_eval \Diamond eval\_dev, \; i(DEcon) \Diamond \textbf{\textit{dev\_eval}}, \; i(EDcon) \Diamond \textbf{\textit{eval\_dev}}, \; i(EDcon) \Diamond o(eval\_dev) \Diamond \textbf{\textit{Evaluator}}$$

$$T_1 \equiv \textbf{\textit{Device}} \Diamond o(DEcon) \rightarrow o(Decon) \Diamond \textbf{\textit{Device}}, \; \textbf{\textit{Device}} \Diamond o(DEcon)$$

$$T_2 \equiv \textbf{\textit{Device}} \Diamond o(DEcon) \Diamond i(dev\_eval) \rightarrow o(DEcon) \Diamond i(dev\_eval) \Diamond \textbf{\textit{Device}}, \textbf{\textit{Device}} \Diamond o(DEcon) \Diamond i(dev\_eval)$$

$$T_3 \equiv i(dev\_eval) \Diamond \textbf{\textit{dev\_eval}}, o(dev\_eval) \Diamond \textbf{\textit{Device}} \rightarrow i(dev\_eval) \Diamond \textbf{\textit{dev\_eval}},$$

$$T_4 \equiv i(DEcon) \Diamond o(dev\_eval) \Diamond i(EDcon) \Diamond o(eval\_dev) \Diamond \textbf{\textit{dev\_eval}} \rightarrow o(dev\_eval) \Diamond i(EDcon) \Diamond o(DEcon) \Diamond \textbf{\textit{eval\_dev}}, \; i(DEcon) \Diamond o(dev\_eval) \Diamond i(EDcon) \Diamond o(DEcon) \Diamond \textbf{\textit{dev\_eval}}, \; i(DEcon) \Diamond \textbf{\textit{Device}} \Diamond o(DEcon)$$

$$T_5 \equiv i(DEcon) \Diamond o(dev\_eval) \Diamond \textbf{\textit{Evaluator}}, \; o(DEcon) \Diamond i(dev\_eval) \Diamond o(eval\_dev) \Diamond \textbf{\textit{dev\_eval}} \rightarrow o(dev\_eval) \Diamond \textbf{\textit{Evaluator}}, \; i(DEcon) \Diamond o(dev\_eval) \Diamond \textbf{\textit{Evaluator}}, \; i(dev\_eval) \Diamond o(eval\_dev) \Diamond \textbf{\textit{eval\_dev}}$$

$$T_6 \equiv o(dev\_eval) \Diamond \textbf{\textit{Evaluator}}, i(dev\_eval) \Diamond o(eval\_dev) \Diamond \textbf{\textit{dev\_eval}} \rightarrow o(eval\_dev) \Diamond \textbf{\textit{eval\_dev}}$$

$$T_7 \equiv i(dev\_eval) \Diamond \textbf{\textit{Device}} \Diamond o(DEcon), o(EDcon) \Diamond \textbf{\textit{dev\_eval}} \rightarrow \textbf{\textit{Device}} \Diamond o(DEcon) \Diamond i(EDcon)$$

T_1 allows devices to send a request to evaluation center.

T_2 denotes devices are ready to communicate with ***dev_eval***. After sending requests, devices wait for acknowledging from ***dev_eval***. To generate multiple instances of ***Device***, *Cham* regenerates ***Device*** molecules to allow other ***Device***s to send request messages. The rule restricts some ***Device*** sends new requests before receiving acknowledgement from ***dev_eval***.

T_3 specifies the synchronization between devices and ***dev_eval***. After that, ***dev_eval*** is ready to receive another device requests.

T_4 specifies ***eval_dev*** receives requests from devices and forwards them to evaluate components and after that, ***eval_dev*** is ready to receive another device requests.

T_5 denotes device requests are forwarded to evaluator components. Then ***Evaluator*** sends an acknowledgement to ***eval_dev***.

T_6 denotes ***eval_dev*** receives the acknowledge from ***Evaluator*** and return it to devices.

T_7 denotes devices receive the acknowledge from ***eval_dev*** and are ready to send new device requests.

B. Quantitative Analysis Results

Moving beyond the specification of the software architecture and model check, we have implemented a prototype of *PHmD* based on *Cham*. The architecture-layer entities are implemented in Java using *Cham* specification. Java-RMI interface is employed to store and retrieve necessary architectural information from design phrase and perform a further model check. The self-adaptive effectiveness and performance of *Cham* framework's are two important issues requiring evaluation. We use the jogging scenario of *PHmD* to illustrate the two aspects of the proposed framework. To demonstrate this, an experiment was conducted on a dedicated testbed under different running conditions with regard to different concurrent threads, user requests, wireless bandwidth and servers, respectively. This experiment improved Viterbi-I algorithm on *PHmD* system that brought about a certain latency for customers. Our experiment results indicate that the proposed method improved system performance while specific loads used in the experiment, as illustrated in Figure 9 compared with Figure 8. Results are shown for system performance without self-adaptation in Figure 8, where the latency experienced by customers never falls on the desired threshold. Figure 9 shows, on the other hand, that the customers experience performance improvement by falling on the optimal value after the period of time is used to execute the optimization algorithm. It is not unexpectedly revealed that the optimization process has an associated latency. However, this is deserved for the *PHmD* to be self-adaptive according to the optimization algorithm because the latency falls back into the range (depicted in black dash line in Figure 9) afterwards.

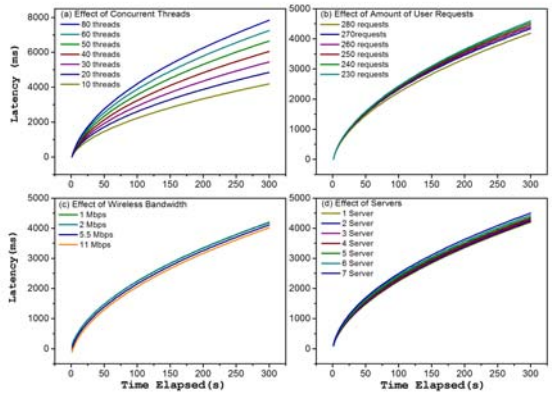

Figure 8. Latency experienced without self-adaptation.

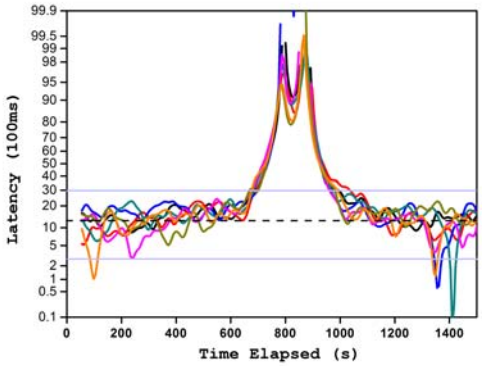

Figure 9. Overall latency experienced with self-adaptation using the optimization algorithm.

VII. Conclusions and Future Works

In recent years, a self-adaptation method based on software architecture has become a new research focus. Software architecture provides a whole guideline to create a target system, including specification of architectural elements, constraint of components and the communication contract between architectural elements and software survival environment. The proposed method overcomes the shortcoming of previous research method based on qualitative analysis. CHAM is employed to get specification of software architecture of CPS. By the improved Viterbi-I algorithm, the system promotes performance.

Software architecture-based method showed the effectiveness of performance improvement over the long term. However, the repair intervals of self-adaptive actions are under consideration. The jetter would be possible if the interval is too short. On the other hand, if it is too long, the effectiveness of the self-adaptive actions could not be demonstrated. This remains a wide-open research issue.

Acknowledgment

This work was supported in part by a grant from Zhejiang Provincial Education Department Research Projects (No. Y201119886).

References

[1] Derler P, Lee E A, Vincentelli A S. "Modeling Cyber-Physical Systems," Proceedings of the IEEE, vol.100, no.1, pp. 13-28, 2012.

[2] Conti M, Das S K, Bisdikian C, et al. "Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber–physical convergence," Pervasive and Mobile Computing, vol.8, no.1, pp. 2-21, 2012.

[3] Cheng S-W, Garlan D. "Handling Uncertainty in Autonomic Systems," In International Workshop on Living with Uncertainties (IWLU'07), colocated with the 22nd International Conference on Automated Software Engineering (ASE'07), Atlanta, Georgia, USA, pp. 2007.

[4] Celiku O, Garlan D, Schmerl B. "Augmenting Architectural Modeling to Cope with Uncertainty," In Proceedings of the International Workshop on Living with Uncertainty(IWLU'07), Atlanta, Georgia, USA, pp. 2007.

[5] Cheung L, Golubchik L, Medvidovic N, et al. "Identifying and Addressing Uncertainty in Architecture-Level Software Reliability Modeling," In Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, Long Beach, California, USA, pp. 1-6, 2007.
[6] Dai Y-S, Xie M, Long Q, et al. "Uncertainty Analysis in Software Reliability Modeling by Bayesian Analysis with Maximum-Entropy Principle," Software Engineering, IEEE Transactions on, vol.33, no.11, pp. 781-795, 2007.
[7] Seceleanu T, Garlan D. "Developing adaptive systems with synchronized architectures," Journal of Systems and Software, vol.79, no.11, pp. 1514-1526, 2006.
[8] Cheng S-W, Garlan D, Schmerl B. "Architecture-based self-adaptation in the presence of multiple objectives," In Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems, Shanghai, China, pp. 2-8, 2006.
[9] Safi G, Mirian-Hosseinabadi S-H. "A Service Based Approach to Self-Adaptive Software Systems Based on Constructing a Group of Autonomic Elements," In Engineering of Autonomic and Autonomous Systems (EASe), 2010 Seventh IEEE International Conference and Workshops on, 101-105, 2010.
[10] Chun I, Park J, Lee H, et al. "An agent-based self-adaptation architecture for implementing smart devices in Smart Space," Telecommunication Systems, vol.pp. 1-12, 2011.
[11] Mckinley P, Cheng B C, Ramirez A, et al. "Applying evolutionary computation to mitigate uncertainty in dynamically-adaptive, high-assurance middleware," Journal of Internet Services and Applications, vol.3, no.1, pp. 51-58, 2012.
[12] Barnes J, Garlan D, Schmerl B. "Evolution styles: foundations and models for software architecture evolution," Software & Systems Modeling, vol.pp. 1-30, 2012.
[13] Wang H, Ying J. "Toward Runtime Self-adaptation Method in Software-Intensive Systems Based on Hidden Markov Model," In *Computer Software and Applications Conference, 2007. COMPSAC 2007 - Vol. 2. 31st Annual International*, Beijing, China, pp. 601-606, 2007.
[14] Gerard B, Gerard B. "The chemical abstract machine," In Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, San Francisco, California, United States, pp. 1990.
[15] Hua Wang, Zhijun Zheng. "Self-adaptive Method Based on Software Architecture by Inspecting Uncertainty," In 2010 International Conference on Artificial Intelligence and Computational Intelligencen (AICI 2010), Sanya, pp. 208-214, 2010.
[16] Wang H, Ying J. "An Approach for Harmonizing Conflicting Policies in Multiple Self-Adaptive Modules," In Machine Learning and Cybernetics, 2007 International Conference on, Hong Kong, pp. 2379-2384, 2007.
[17] Inverardi P, Wolf A L. "Formal specification and analysis of software architectures using the chemical abstract machine model," Software Engineering, IEEE Transactions on, vol.21, no.4, pp. 373-386, 1995.
[18] Poladian V, Garlan D, Shaw M, et al. "Leveraging Resource Prediction for Anticipatory Dynamic Configuration," In Self-Adaptive and Self-Organizing Systems, 2007. SASO '07. First International Conference on, 214-223, 2007.
[19] Jaeyoun J, Youngeung K, Yeondae C, et al. "A study on implementation of model checking tool for verifying LTS specifications," In Information Networking, 1998. (ICOIN-12) Proceedings., Twelfth International Conference on, 539-543, 1998.
[20] Kenneth S, H, Ctor G, et al. "Altruistic locking," ACM Trans. Database Syst., vol.19, no.1, pp. 117-165, 1994.
[21] Cleaveland R, Steffen B. "A linear-time model-checking algorithm for the alternation-free modal μ-calculus," Formal Methods in System Design, vol.2, no.1, pp. 121-147, 1993.
[22] Rabiner L R. "A Tutorial on hidden Markov models and selected applications in speech recognition," Proceedings of the IEEE, vol.77, no.2, pp. 257-286, 1989.

Hua Wang was born in Zhejiang, China. He received his Ph.D. degree in computer science from Zhejiang University, China, in 2009.

He is currently an associate professor at School of Information and Electronic Engineering, Zhejiang University of Science and Technology, China. His research interests are in Cyber-physical System, Self-adaptive System and Software Architecture.

Dr. Wang is a senior member of International Association of Computer Science and Information Technology (IACSIT).