A Software Dependability Growth Model based on Self-Reconfiguration

Qian Zhao1 HuiQiang Wang1 HongWu Lv1 Guangsheng Feng1

1 College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China

Abstract

With wide application of computers, software quality attracts people’s attention. Traditional software dependability theory can’t satisfy people’s requirement, which need induct new idea to resolve the serious software quality crisis. This paper uses self-reconfiguration mechanism of Autonomic Computing to handle problems of software dependability. Senior and Junior Self-Reconfiguration Method are defined in this paper. A software dependability growth model based on self-reconfiguration (SDGMSR) is established, which is analyzed by means of Markov Regenerative Stochastic Petri Net (MRSPN). Experimental results show that our approach can improve software dependability and reduce software maintenance cost more effectively while chooses a proper self-reconfiguration period.

Keywords: Autonomic Computing, Self-reconfiguration, Software Dependability, MRSPN

1. Introduction

With the application of computers in all fields, softwares have penetrated many crucial departments, such as bank, national defence and military affairs, which results in human relying on them unprecedentedly. However, the quality of software have not made people satisfied, moreover, the management and maintenance of running dependable becomes more and more difficult. The theories and technics of software security are unwieldy and difficult to meet actual application requements. An atomismic, flexible and fine-grained management method is expected to resolve the problem of software undependability.

Autonomic Computing, considered as a new method for settling the self-management of heterogeneous computing systems, has become an international research focus. Development software dependability growth model based on autonomic computing can cause qualitative changes of current research and design in software dependability.

The main idea this paper is using the self-reconfiguration of Autonomic computing to realize the software dependability oriented self-management mechanism, reduce human intervention and improve software dependability dynamically, which provide a noval method for software dependability study.

2. Related works

Dependability is researched more than 60 years which results in many achievements, which accelerate the development of software dependability. Measurement-based dependability analysis of operational software has a history of over 20 years[1]. With the development of system dependability and universality of software application, software dependability
is researched widely, and research results are exerted in many fields. In theory aspect, Bev Littlewood \[2\] outlines specific difficulties in applying a sound engineering approach to software reliability engineering, and establishes a certain foundation for development of system dependability. System dependability is analyzed in detail from the view of software failure in \[3\]. It defines basic error characteristics and relation between software error and hardware in \[4\]. Multivariate State Estimation Technique is used to enhance the software dependability by Sun Microsystems Inc. and University of Maryland in \[5\], their results suggest that they can cheaply and reliably predict impending runtime failures and respond to them in time to improve the system’s dependability. Arup Mukherjee \[6\] measures software dependability from the view of software robustness, and classifies as well as compares the software faults. Software dependability attributes are classified in detail, which provides reference for research of software dependability attributes \[7\]. Chen points the state of art of its engineering technologies for high confidence software and the challenges it faced and the importance of formalization \[8\]. Hong MEI proposes a realization method through dependable structure and reflective middleware \[9\].

In project aspect, DARPA, NSF, NASA, NSA, NIST, FAA, FDA and DoD have participated in the research of software dependability and high confidence software. NSTC proposes a serious reports. Computer science and engineering department of Technische Universität Darmstadt has research content about Trusted Project in Databases and Distributed Systems Group project.

This paper uses self-reconfiguration mechanism for reference and researches how to dynamically change considering exterior environment and application demand, and make out proper reflection automatically according to dependable demand. MRSPN is used to analyze the software dependability growth model based on self-reconfiguration (SDGMSR). Related parameters of software self-reconfiguration are optimized in this paper, which aims at using software transparently.

### 3. Software Dependability Growth Model based on Self-Reconfiguration

**Symbol Definition:**
- \( t \) is time;
- \( \sigma \) is threshold value of time;
- \( f \) is failure proportion of components, whose threshold value is \( F \);
- \( v \) is predictive value of software autonomic dependability, whose threshold value is \( V \);
- \( R(t) \) is inside and outside rules at time \( t \), which is get by self-reflection component:
  - \( R_{pr_s} \) is rule in senior self-reconfiguration database;
  - \( R_{unpr_s} \) is rule in junior self-reconfiguration database;
  - \( A_{pr_s} \) is action according to rule in senior self-reconfiguration database;
  - \( A_{unpr_s} \) is action according to rule in junior self-reconfiguration database;
- \( f \) is failure proportion of components, whose threshold value is \( F \);
- \( v \) is predictive value of software autonomic dependability, whose threshold value is \( V \);
- \( R(t) \) is inside and outside rules at time \( t \), which is get by self-reflection component:
- \( R_{pr_s} \) is rule in senior self-reconfiguration database;
- \( R_{unpr_s} \) is rule in junior self-reconfiguration database;
- \( A_{pr_s} \) is action according to rule in senior self-reconfiguration database;
- \( A_{unpr_s} \) is action according to rule in junior self-reconfiguration database;
- \( R \) denotes rule matching;
- \( R \) denotes rule mismatching;
- \( R \) denotes getting action \( A \) according to rule \( R \);
- \( \alpha \) denotes addition new strategy.

To realizing self-reconfiguration without human interference, rule-action strategy is inducted in this paper

#### 3.1. Self-reconfiguration Method
To improving the effective of self-reconfiguration and satisfying the requirement of software dependability growth, rule in senior self-reconfiguration database will be optimized if rule does not match, which is called rule in junior self-reconfiguration database. Rule in junior self-reconfiguration database can become senior during information feedback and test of software.

The definitions of senior and junior self-reconfiguration shows as follow:

**Definition 1 (Senior Self-Reconfiguration, SSR):** Rules apperceived by Self-reflection component in current time is matched with SSR. If rule matching, the strategy is sprung and the action of strategy will be executed.

\[ \forall t \geq 0, \text{if } R(t) = R_{pr_s}, \text{ then } R_{pr_s} \Rightarrow A_{pr_s}, \]

put strategy in practice.

Accomplishment of SSR strategy is show in Fig.1.

**Definition 2 (Junior Self-Reconfiguration, JSR):** Rules apperceived by Self-reflection component in current time is not matched with SSR and the action according to rule can not be executed. If satisfying require of software dependability growth, dynamic extension of self-reconfiguration database must be added new rule-action strategy.

\[ \forall t \geq 0, k \in Z^+ \cup \{0\}, a \in Z^+ , \text{if } R(t) \not= R_{pr_s}, \]

let \( k = 0, \)

\( (R_{pr_s} \Rightarrow A_{pr_s}) \& R(t) \Rightarrow A_{wp}, \text{ then } A_{wp} \Rightarrow A_{pr_s}, \cup \{k = k+1\} \)

put strategy in practice

Accomplishment of JSR strategy is show in Fig.2.

**3.2. Software Dependability Growth Model**

SDGMSR is show in Fig.3.

![Fig.3. Software Dependability Growth Model based on Self-Reconfiguration](image)

Self-Reflection Component extracts parameters values from inside and outside environment and provides them to Self-Reconfiguration Component. Self-Reconfiguration Component judges whether they need reconfigure or not, for example preestablished time or prior threshold value of predictive value of software autonomic dependability. If Self-Reconfiguration is needed, Self-Reconfiguration Component compares the collected rule with the rule in SSR Database, if they match with each other, run the according action through Control Component; if they don’t match, find the rule in JSR Database. If there exists matching rule, runs the according action through Control Component and records the new rule-action strategy into JSR Database. If rule-action strategy in JSR Database satisfies given condition, such as using times of strategy id more than a threshold value, it will be deleted from JSR Database and stored in SSR Database.

**4. Performance Analysis of SDGMSR based on MRSPN**

Because time interval of SDGMSR may be certain, software will not be a Markov
chain, which made Continue Markov chain theory not fit for the analysis. A non-
Markov theory MRSPN is used in this paper for analysis.

For a MRSPN model, corresponding stochastic processes is \( X_t = (X_t; t \geq 0) \),
there \( X_t \) is software state in \( t \), and the state space is \( \Omega \). Sample in stochastic pro-
cesses \( X_t \), regenerative state set \( \Omega = \{X_n; n \geq 0\} \) (\( \Omega := \Omega \)), sampled re-
generative time point is \( \tau_n (n = 1, 2, \ldots) \), and \( \tau_1 < \tau_2 < \tau_3 < \ldots \). \( \{X_n; \tau_n \geq 0, n = 1, 2, \ldots\} \) is
embedded Markov chain[10,11].

4.1. MRSPN Model

Model of SDGMSR based on MRSPN shows in Fig.5. In the figure, the circles
represent places with dots inside representing the tokens held inside that place. Filled rectangle denotes constant transi-
tion and empty rectangles denote EXP transition.

The normal service state is modeled by the place \( P_{up} \). Transition \( T_{fail} \) models the
degradation of software dependability. When \( T_{fail} \) is fired, software may come
into place \( P_{fail} \), which models the failure state of software. Transition \( T_{crash} \) models
software crash aroused by software fault, which made software come into \( P_{crash} \) place. The crashed software restarts and comes into normal service state, which is
modeled by \( T_{run} \). \( T_{run} \) models software self-reconfiguration operation to
improving system dependability. \( T_{st1} \) modules JSR operation and \( T_{st2} \) modules
SSR one. After a confirmed time \( \tau \), \( T_{period} \) is executed. Token in \( P_{period} \) comes into
place \( P_{ST} \). Whenever the token is in \( P_{run} \) or \( P_{fail} \), \( T_{st1} \) or \( T_{st2} \) must be executed and clock must be reset. \( P_{ST} \) models self-
reflection. During the self-reconfiguration phase, every other activity in the system is
suspended. This is modeled by inhibitor arcs from place \( P_{ST} \) to transitions \( T_{fail} \) and
\( T_{down} \). \( T_{fail}, T_{st1}, T_{st2}, T_{crash} \) and \( T_{run} \) in Fig.5 are EXP transitions and the transi-
tion rates are \( \lambda_1, \lambda_2, \lambda_3, \lambda_4 \) and \( \lambda_5 \), let \( \sigma \) be the firing time associated with \( T_{period} \).

Fig.5. Model of SDGMSR based on MRSPN

4.2. Analysis of MRSPN Model

Because \( P_{ST} \) produces data real-timely, model of SDGMSR based on MRSPN
can be predigested and denoted by 5-
tuple \( (P_{run}, P_{fail}, P_{crash}, P_{ST}, P_{period}) \).

Fig. 6 shows the reachability graph with ovals representing the markings and arcs
representing possible transitions between the markings. From Fig. 6, there are 5
markings are possible viz \( (01010, 01010, 00110, 01001) \),
and be represented by \( M_1, M_2, M_3, M_4 \) and \( M_5 \). The state space of
MRGP is \( \Omega = \{M_1, M_2, M_3, M_4, M_5\} \).

Fig.6. Reachability Graph for SDGMSR based
on MRSPN

From the actual connection and transi-
tion of model, the state of software is de-
termined uniquely by the current state if
coming into \( M_1 \), \( M_2 \), \( M_3 \), \( M_4 \) or \( M_5 \). Once
state comes into \( M_2 \), next state is still decided by many factors and may be state
\( M_4 \), \( M_5 \) or \( M_3 \). So, state space of the un-
derlying MRGP is \( \Omega = \{M_1, M_2, M_3, M_4, M_5\} \).

The state transition matrix is \( K(t) \).

\[
K(t) = \begin{bmatrix}
0 & K_{up}(t) & K_{up}(t) & K_{up}(t) \\
K_{fail}(t) & 0 & 0 & 0 \\
K_{crash}(t) & 0 & 0 & 0 \\
K_{st1}(t) & 0 & 0 & 0
\end{bmatrix}
\]
$K_{\text{ST}}(t) = \{X_i, X_j, \tau < t | X_i = X_j \}$

where, $u(t)$ is unit-step function.

$K_{\text{WT}}(t) = \{X_i = M_1, \tau < t | X_i = M_1 \}$

$= \left\{ \begin{array}{ll}
1 - \frac{\lambda}{\lambda - \delta} e^{-\lambda \tau} + \frac{\lambda}{\lambda - \delta} e^{-\delta \tau} & 0 \leq \tau < \sigma \\
1 - \frac{\lambda}{\lambda - \delta} e^{-\lambda \sigma} + \frac{\lambda}{\lambda - \delta} e^{-\delta \sigma} & 1 \geq \sigma
\end{array} \right.$

$K_{\text{WIM}}(t) = \{X_i = M_i, \tau < t | X_i = M_i \}$

$= \frac{\lambda}{\lambda - \delta} [e^{-\lambda \sigma} - e^{-\delta \sigma}] u(t - \sigma)$

$K_{\text{WIM1}}(t) = \{X_i = M_i, \tau < t | X_i = M_i \}$

$= 1 - e^{-\lambda \tau}$

$E(t) = \sum_{\tau=0}^{K(t)} E(t | \tau)$

$E(t) = \left[ \begin{array}{c}
E_{\text{E}}(t) \\
E_{\text{E-M}}(t) \\
E_{\text{E-WT}}(t) \\
E_{\text{E-WIM}}(t) \\
E_{\text{E-WIM1}}(t)
\end{array} \right]$

$E_{\text{E-M}}(t) = P \{X_i = M_1, \tau < t | X_i = M_1 \}$

$= 1 - (1 - e^{-\lambda \tau}) \cdot [u(t) - u(t - \sigma)]$

$E_{\text{E-WT}}(t) = 1 - [E_{\text{E-M}}(t) + E_{\text{E-WT}}(t) + E_{\text{E-WIM}}(t) - E_{\text{E-WIM1}}(t)]$

$E_{\text{E-WIM}}(t) = \frac{\lambda}{\lambda - \delta} [e^{-\lambda \sigma} - e^{-\delta \sigma}] [u(t) - u(t - \sigma)]$

$E_{\text{E-WIM1}}(t) = 1 - (1 - e^{-\lambda \tau}) = e^{-\lambda \tau}, i = 3, 4, 5$

It can get steady state probabilities of software according the transient state probabilities.

$\alpha_j = \int_0^\infty E_j(t) dt$

$N = \lim_{t \to \infty} K(t)$

$\pi_j = \sum_{i=1}^{\omega} \alpha_i$

From the actual SDGMSR, software is available in state $M_1$ and $M_2$. The availability of software is denoted as $S_A$.

$S_A = \frac{\pi_1 + \pi_2}{\pi_1 + \pi_2 + \pi_3 + \pi_4 + \pi_5}$

Let $C_C$ be the fixed cost per unit time when the software is crashed and $C_{ST}$ be the fixed cost per unit time when it has self-reconfiguration. $C$ is a random variable denoting the cost incurred, then, the expected total cost incurred in the interval $[0, \sigma]$ is $E[C]$.

$C = (\pi_1 + \pi_2) C_{ST} + \pi_3 C_C$

5. Simulation Experiment Result

<table>
<thead>
<tr>
<th>Parameter Values</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_1$</td>
<td>240 hours</td>
</tr>
<tr>
<td>$\lambda_2$</td>
<td>2160 hours</td>
</tr>
<tr>
<td>$\lambda_3$</td>
<td>3 minutes</td>
</tr>
<tr>
<td>$\lambda_4$</td>
<td>2 seconds</td>
</tr>
<tr>
<td>$C_C$</td>
<td>5000 $$/hour</td>
</tr>
</tbody>
</table>

One hand much frequent software self-reconfiguration will exhaust system resource and affect the usage of users, the other hand, if software always runs on an environment with failure, which made software crashed and also affect the usage of users. The selection of $\sigma$ value is very important for SDGMSR and ensures software from out of service and crash. In the view of the overall trend of software availability, with $\sigma$ growth, it will increase firstly, and then decrease, which show in Fig.7. If $\sigma = \infty$, no self-reconfiguration, the expected cost is a function of $C_{ST}$ only and hence all graphs approach the same value; if $\sigma = 0$, the software is always in self-reconfiguration operation and the cost incurred is infinite.

6. Conclusions and Future Works

This paper uses self-reconfiguration me-
lanism of Autonomic Computing and handles problems of software dependability through dynamic capturing resources inside and outside environment. According to prior predictive value of software dependability or relative threshold value, SDGMSR uses rule-action strategy of SSR or JSR to instruct the actions of Control Component and adjusts the entities of software, which made the growth of software dependability. At last, MRSPN is used to analyze SDGMSR. In accordance with the result of simulation testing, SDGMSR can be used to conduct the growth of software dependability, and is an effective method to handle problem of software quality.

In the future, the prior predictive algorithm of software dependability and relative threshold value should be work out. Optimization algorithm of self-reconfiguration strategy is also the hot point of the work team.

7. References


