Are Your Sites Down?
Requirements-Driven Self-Tuning for the Survivability of Web Systems*

Bihuan Chen[1], Xin Peng[1], Yijun Yu[2], Wenyun Zhao[1]
[1]School of Computer Science, Fudan University, Shanghai, China
[2]Centre for Research in Computing, The Open University, Milton Keynes, UK
{09210240005, pengxin, wyzhao}@fudan.edu.cn, y.yu@open.ac.uk

Abstract—**Running in a highly uncertain and greatly complex environment, Web systems cannot always provide full set of services with optimal quality, especially when work loads are high or subsystem failures are frequent. Hence, it is significant to continuously maintain a high satisfaction level of survivability, hereafter *survivability assurance*, while relaxing or sacrificing certain quality or functional requirements that are not crucial to the survival of the entire system. After giving a value-based interpretation to survivability assurance to facilitate a quantitative analysis, we propose a requirements-driven self-tuning method for the survivability assurance of Web systems. Maintaining an enriched and live goal model, our method adapts to runtime tradeoff decisions made by our PID controller and goal-oriented reasoner for both quality and functional requirements. The goal-based configuration plans produced by the reasoner is carried out on the live goal model, and then mapped into system architectural configurations. Experiments on an online shopping system are conducted to validate the effectiveness of the proposed method.**

Keywords-survivability assurance; earned business value; goal-oriented reasoning; self-tuning

I. INTRODUCTION

The commercial success of modern e-businesses depends largely on those Web systems that are reliable and resilient to changes in the running environment. Failing to have such a working Web system, the businesses can easily lose their customers because it costs little for them to transfer to other competing websites [1]. On the other hand, the running environment for Web systems is often too complex to predict at design time. Uncertainty factors, such as access load, resources availability, and platforms and components heterogeneity, makes it hard to provide the full set of services with optimal quality, especially when the workloads are high or failures in subsystems occur frequently.

On 29 June, 2010, *Amazon.com*, known for its reliability, suffered a widespread outage for more than three hours. According to the news report from CNET [2], only partial or even blank pages were displayed then, while the searching services didn't work, and the shopping carts were all forced to be emptied; "Amazon faces a potential loss of an average of $51,400 a minute when its site is offline. Amazon shares closed down 7.8 percent." It is worth to mention that it is not the first time Amazon suffers an outage: a 30-minute outage in 1999, an hour outage in 2006, and a 90-minute outage in 2008. Interestingly, intervals of such incidents are shorter while their duration is getting longer.

Therefore, survivability rather than absolute reliability is a more practical and reasonable consideration for Web systems. A system with survivability has the capability of ensuring critical system services under severe or adverse conditions, with acceptable quality degradation or even sacrifice of some desired services [3]. Due to its intrinsically dynamic characteristic, on one hand, survivability should be assured by runtime reconfiguration on the structures and behaviors of the software system that is autonomic to avoid irresponsive and error-prone human intervention. On the other hand, survivability should be interpreted in a quantitative way to guide the reconfiguration precisely.

To achieve quantitative survivability assurance without human intervention, certain self-adaptation mechanism is required to autonomously tune the quality of the offering and the set of provided services under different conditions.

Such survivability-oriented self-tuning can be considered as runtime tradeoff decisions about both quality and functional requirements. Quality tradeoff concerns the decisions among multiple conflicting quality attributes, e.g. security, usability, performance, etc. Considering alternative architectures and designs, it is common that quality requirements conflict with each other and the one better in certain quality dimensions can be worse in others [4]. Therefore, when the system cannot ensure optimal satisfaction levels for all the quality requirements, it is reasonable to relax the satisfaction levels of the *desired* but conflicting quality requirements to ensure more *crucial* ones. When any quality tradeoff decisions cannot achieve the survivability objective, functional tradeoff decisions come to the rescue. In functional tradeoff decisions, some desired functional services may be sacrificed to ensure the crucial ones and its quality. Only when the survivability is achievable, can the system reconfigure to satisfy the sacrificed functional requirements and the relaxed quality requirements.

In order to achieve autonomic survivability assurance for Web systems, conceptually we adopt the MAPE (Monitor-Analyze-Plan-Execute) control loop [5], which is being widely used in self-adaptive systems [6-7]. What we propose is a requirements-driven self-tuning method, extending our previous work on runtime tradeoff decisions about quality requirements [8], the method further takes into account runtime tradeoff decisions about functional requirements. Here the challenges are *when* and *which* desired functional service ought to be unbound to ensure the crucial services and their

* This paper is an extended version of our poster paper [34] that briefly presents the motivation and overview of our method.

quality, and *when* and *which* such unbound services should be bound to preserve the functional integrity. To address the challenges, we modify the PID controller previously reported [8], present a goal-oriented reasoning algorithm, and maintain an enriched and live goal model to facilitate the reasoning. Using the traceability links from goal models to implementations [9], reconfigurations on the goal model are mapped to architectural reconfigurations on the Web system.

On the other hand, since survivability assurance implies the protection of the value delivered by a system [3], we adopt a value-based interpretation of survivability assurance at two levels of quantifications. The first level concerns *how to quantify survivability assurance* while the second level concerns *how to model survivability assurance.* Besides, the two levels respectively facilitate the "when" and "which" part of challenges.

The rest of the paper is organized as follows: Section II introduces the running example used in the paper. Section III presents our value-based interpretation of survivability assurance. Section IV proposes our requirements-driven self-tuning method including the method overview and some key algorithms. Section V evaluates the method with discussion. Section VI introduces and compares with some related work before Section VII draws our conclusions.

II. A RUNNING EXAMPLE

We use an online shopping system as the running system to illustrate our method throughout the rest of the paper. The system supports the following use cases. After logged into the account, a customer can (i) search for the products, (ii) view the details of interested products such as price and produced area, other recommended products and customer reviews, and (iii) add the wanted products to the shopping cart. In addition, the customer can (iv) consult any question about the products, and (v) share reviews with other customers. The customer must (vi) give details about address and shipment, and (vii) pay the bill for purchasing anything before finally (viii) the orders are placed.

Usability and performance are generally two conflicting quality requirements for such a Web system. There are two typical runtime tradeoff scenarios:

(1) Whether usability or performance should be given a higher priority? In normal situations, usability measured as "themed look and feel" should be given a higher priority while it does not have apparently negative influence on the performance. Once the system is overloaded by a large number of concurrent requests that could reduce the successful purchase orders or even crash down the system, however, performance should be given a higher priority.

(2) Even when performance has been given a higher priority in the high workload scenario, the system could still suffer from a great degradation of performance that drastically damages the overall quality of the system, reduces the successful purchase orders and threatens the survival of the system. Naturally, few customers would continue shopping on a system with both poor user-friendliness and slow response. Therefore, it is reasonable to stop certain services (e.g., making reviews) temporarily by providing hints to lower customers' expectation (e.g., sorry. making reviews is temporarily stopped due to the high concurrent loads) in order to release some resources and thus relieve the severe scenario and increase the chance of success in purchase orders.

Both scenarios illustrate the requirements-driven tradeoff decisions. The difference between them is whether the desired quality requirements such as usability can be temporarily relaxed, and whether the desired functional requirements such as making reviews can be temporarily sacrificed when survivability is threatened. To tell such differences, requirements goal modeling is used to model the quality and functional requirements. Fig. 1 shows the goal model of the online shopping system, where quality requirements are modeled as soft goals (cloudy shapes) while functional requirements are modeled as hard goals (rounded rectangles).

In this example, there are four soft goals: [Minimize] **Response Time**, [Maximize] **Usability**, [Minimize] **Cost** and [Maximize] **Availability**. Response time measures the average duration the server process the requests, usability measures the "themed look and feel", cost measures the amount of money that should be paid to the third-party payment service providers, and availability measures the probability that the payment service is accessible.

Note that **Search Products**, **Basic Information**, **Pay Bill**, referred as *variation points*, are OR-decomposed into alternative sub-goals. They contribute to the four soft goals differently. For instances, either a plain form (e.g., only descriptive texts) or a rich form (e.g., texts, pictures and flashes) can be chosen to view products' basic information. The plain form helps to minimize the response time but hurts the usability, while the rich form contributes to them in the other way around.

III. VALUE-BASE INTERPRETATION OF SURVIVABILITY ASSURANCE

Nowadays, Web systems support business activities that often represent the business itself [10]. Hence, the survivability assurance of Web systems implies the protection of both the business and the business value. The value-based software engineering (VBSE) [11] and value-based requirements engineering (VBRE) [12] are two views with emphasis on integrating value into the SE and RE principles and practices. In this section, we present a two-level value-based interpretation of survivability assurance. The first level concerns how to quantify survivability assurance from the perspective of VBSE, while the second level concerns how to model survivability assurance from the perspective of VBRE.

A. Quantifying Survivability Assurance

In the first level, we interpret survivability assurance as the capability of maximizing the satisfaction level of system value proposition: a formula measuring the "earned business value" from the perspective of service-side business value is defined and used as the quantitative indication for the tuning process.

However, many factors can influence the earned business value in a direct/indirect or quantifiable/unquantifiable way.

In online shopping, for example, sales is a direct factor and can be easily quantified into profits, while customers' review is an indirect factor that is hard to be quantified into values. In terms of probability, good reviews of a product attract more customers to purchase the same product while negative reviews prevent potential customers from purchasing it. Therefore, the earned business value should be measured by marketing or customer-relationship management experts through a comprehensive economic analysis on such complicated factors.

To illustrate, we simplify the earned business value in our example as a formula involving the following 5 factors. Each of the 5 factors corresponds to a kind of runtime system transaction, and each transaction can produce certain amount of profits.

(1) **Sales**: once a bill is paid, 5 percent of the money is the profit;
(2) **Product Details**: once product details is viewed, it will produce $0.08 on average because the customer will likely purchase the product after viewing;
(3) **Reviews**: once a review of a product is made, it will produce on average $0.032 because other customers are likely to purchase the product after viewing the review;
(4) **Consultations**: once a consultation of a product is made, it will produce on average $0.04 because the customer may buy the product after receiving satisfactory replies;
(5) **Recommended Products**: once recommended products are displayed, it generates on average $0.048 because the customer may also buy the recommended products.

Table I shows the transaction information of the system at 2 time slots. At the time slot 1, the amount of money of paid bills is $100; there are 8 transactions of viewing product details, 7 transactions of reviewing, 9 transactions of consultation and 3 transactions of displaying recommended products. And the total earned business value is thus measured as $100 * 5% + 8 * $0.08 + 7 * $0.032 + 9 * $0.04 + 3 * $0.048 = $6.368.

TABLE I. MEASUREMENT OF THE EARNED BUSINESS VALUE

Time Slot	Transactions					Total ($)
	(1) ($)	(2) (#)	(3) (#)	(4) (#)	(5) (#)	
1	100	8	7	9	3	6.368
2	150	5	3	2	7	8.412

B. Modeling Survivability Assurance

Since we use goal models as the formal representation of the runtime requirements, survivability assurance should also be formally modeled. We observed that some hard goals become unbound without influencing the achievement of their parent goals, which violates the semantics of static goal models. For example, **Make Review** can be unbound, and customers can still achieve the goal of **Online Shopping**. Thus, these hard goals can be temporarily sacrificed to ensure other crucial goals.

Similar to the qualitative concept of *optional goal* [13-14], we enrich the AND/OR decompositions of the goal model with value contribution annotations, *relative parent values*, to characterize to what extent the achievement of the sub-goals will contribute to the achievement of the value of their parent goals, i.e., to model the survivability relationships between sub-goals and their parent goals from a business perspective.

For the sake of simplicity, relative parent value is defined between 0.0 and 1.0 (as shown in Fig. 1). A hard goal's relative parent value is said to be 1.0 if the hard goal must be retained to achieve the value of its parent goal; otherwise, the value of its parent goal can still be partially achieved even when it is unbound.

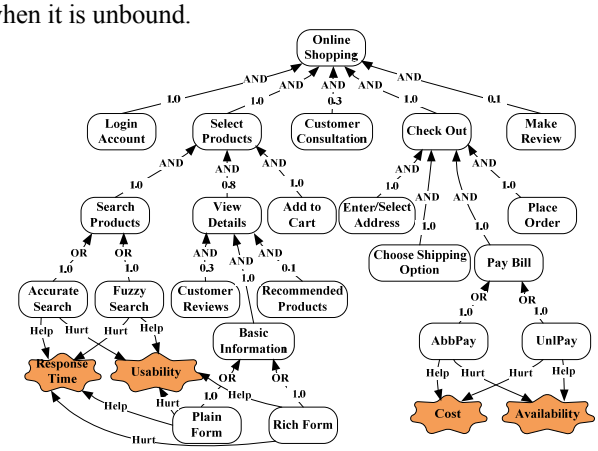

Figure 1. A goal model enriched with relative parent values.

It is worth to note that relative parent values can also deal with the customers' consuming behaviors. For example, if the customers cannot make reviews of products, how many customers will continue or stop purchasing the products? Hence, how to elicit relative parent values can be induced as an economic problem of consumer behavior analysis [15], which can and shall be resolved by economics experts.

In our study, the elicitation of relative parent values was simplified by asking the following question: "If the hard goal is unbound, what is the probability of not achieving its parent goal when all other sibling sub-goals have been achieved?" For example, if **Make Review** is unbound, the probability of **Online Shopping** will not be achieved is 0.1; if **Check Out** is unbound, however, the probability is clearly 1.0 regardless of the achievability of the other sibling sub-goals.

IV. OUR METHOD

In this section, we first present the overview of our requirements-driven self-tuning method that employs the PID controller and the goal-oriented reasoner. After introducing the PID controller, we describe the quality goal reasoner in brief and the functional goal reasoner in detail. Finally, we highlight how to reconfigure the goal model based on the reconfiguration produced by the reasoner.

A. Method Overview

To implement requirements-driven self-tuning of Web systems for survivability assurance, our method maintains a live goal model, postponing the traditionally design-time quality and functional tradeoff decisions to runtime (i.e., late binding) so that they can be adapted autonomously and dynamically in response to the changing environment in order to maximize the satisfaction level of system value proposi-

tion. Fig. 2 gives an overview of our method, illustrating the main components and their mappings to the MAPE control loop. It is worth to mention that our version of MAPE control loop can be triggered when the system is saturated and can be stopped when the system is unsaturated through system performance gauges, since in unsaturated status the system can simply provide the full set of services with acceptable satisfaction quality.

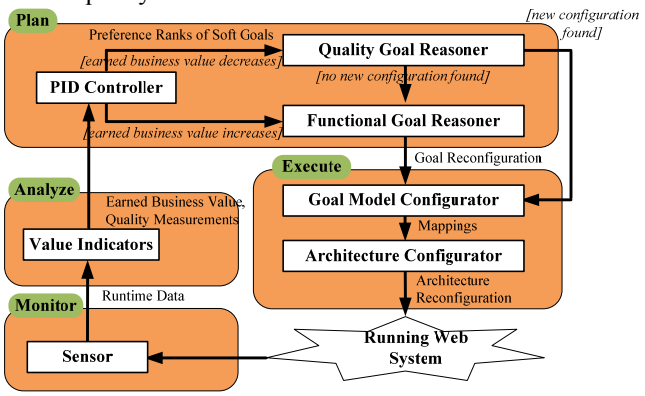

Figure 2. Overview of our method.

Monitor. We use *sensors* to collect runtime data, e.g., failure/success of a payment service, response time of a request, etc., from the running Web system through logging records.

Analyze. By analyze of the runtime data, we use *value indicators* to measure the earned business value as shown in Section III.A, and obtain the quality measurements, e.g., availability of the payment service, average response time of the server, etc., of the related quality requirements.

Plan. Based on the earned business value and various quality measurements, we use the *PID controller* to decide whether or not to make quality or functional tradeoffs decisions in order to maximize the satisfaction level of the system value proposition. In the former case, the PID controller is also used to tune the preference ranks of related soft goals to guide the *quality goal reasoner*. Then, quality or/and *functional goal reasoners* are executed to obtain a reconfiguration on the goal model.

Execute. Finally, we use the *goal model configurator* to reconfigure the goal model according to the planned goal reconfiguration, and we use the *architecture configurator* to execute the self-tuning by reconfiguring the runtime architecture according to the mappings between the goals and the architectural components. Such architecture reconfigurations, including actions of replacing, unbinding and binding a component, are currently supported by a reflective component model (e.g., Fractal [16]).

The monitor runs all the time while analyze, plan and execute run iteratively at regular intervals (e.g., per minute). In the **plan** step, there are three possible planning paths.

(P1) When the earned business value decreases by a certain degree α (i.e., survivability assurance is unachieved), the quality goal reasoner is triggered to produce a set of configurations that can reach the satisfaction levels of high-ranked soft goals by relaxing the satisfaction levels of low-ranked soft goals. Each configuration is a selection of the OR-decomposed goals.

(P2) If no new configuration is found (i.e., survivability assurance cannot be achieved by quality tradeoff decisions), the functional goal reasoner is triggered to produce a reconfiguration that ensures the crucial goals by sacrificing one leaf-level hard goal.

(P3) When the earned business value increases by a certain degree β (i.e., survivability assurance is achieved), the functional goal reasoner is triggered to produce a reconfiguration that improves the functional integrity of the Web system by rebinding the recently sacrificed leaf-level hard goal.

Note that quality tradeoff decisions are made to replace one component with another, and the functional tradeoff decisions are made to unbind or bind a component. Comparing with replacing a component, unbinding or binding a component is a more radical tuning action due to its more significant change to the architecture. And there is often an uncertainty in whether or not such a radical tuning action has taken effect in the target system [17]. As a result, the system could suffer from an oscillation of the tuning action (e.g., unbind and bind the same component alternately). To cope with such oscillation and uncertain effect, we adopt a timed delay for the effect of the radical tuning action to be achieved in the functional tuning execution as suggested by S. W. Cheng et al. [17]. Specifically, once an unbinding or binding action has taken place, a specified window of timed delay is given for the effect of the action to be achieved. In addition, no other unbinding or bind action is allowed to happen in this window. Measurement of the timed delay will be further discussed in our experimental study.

B. PID Control Algorithm

Our MAPE control loop is a typical instance of feedback control loop. Fig. 3 shows its basic structure: a *set point*, the desired characteristics of the controlled *process*, can be specified by technical experts; an *output*, the controlled behavior of a process, provides feedback to the *controller*; the controller tries to minimize the *error*, the delta between the set point and output, to control the process such that the output is close to the set point. And we adopt the PID (proportional-integral-derivative) controller since it is most widely used for its precision, stability, and responsiveness among various feedback controllers [18].

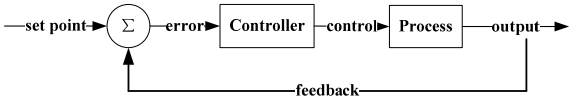

Figure 3. Basic structure of feedback control loop.

Upon receiving the feedbacks, the earned business value and the quality measurements, the PID controller's purpose is to decide: under current environment, whether or not to startup a new cycle of quality and/or functional goal reasoning to tune the requirements tradeoff decisions.

The PID control algorithm (Algorithm 1) used in our method is modified from an earlier version [8]. In the control loop, the set point of the earned business value is the expec-

tation of the earned business value per unit time, which is evaluated as the average of all the past business value earned per unit time (Line 3). Hence, the set point is gradually updated from old value to new one rather than set in advance to reflect the runtime environment. Then, the degree of change of earned business value can be defined as the delta ratio from its set point (Line 2). When the change of earned business value is less than α (α < 0), quality goal reasoner is executed first, followed by functional goal reasoner if needed. When the change of earned business value is larger than β (β > 0), functional goal reasoner is executed. The details of preference ranks tuning (Line 11-15) can be found in [8].

Algorithm 1. PID Control Algorithm.
input: the business value earned in the last time unit *earnedValue* and the monitored quality measurements in the last time unit *QoS*
output: the type of tuning to be executed *tuningType*
procedure
begin
1. *// Calculate the delta ratio of the earnedValue from its set point*
2. *delta* = |(*earnedValue – set point*) / *set point*|*;*
3. update the *set point* as average of the past earned business values;
4.
5. *// Decide to execute which type of tuning*
6. **if** (*delta* < α) **then** *tuningType* = quality;
7. **else if** (*delta* > β) **then** *tuningType* = functional;
8. **else** *tuningType* = null;
9. **end if**
10.
11. calculate the *error signals* and *control variables* of *QoS* using PID formulation, and update their *set points* [8];
12. **if** (*tuningType* = quality) **then**
13. tune the *preference ranks* [8];
14. **if** no configuration found **then** tuningType = functional; **end if**
15. **end if**
16. **return** *tuningType*
end

Choosing the threshold values for α and β must involve considerations on the system's tolerance of earned business value fluctuation. For example, if system has a low tolerance of value fluctuation, α should be larger (e.g., -0.1) and β should be smaller (e.g., 0.1); otherwise, α should be smaller (e.g., -0.4) and β should be larger (e.g., 0.4). Thus, α and β should be determined on the basis of an analysis of the business goal and policy. In our example in the experiments, α is set to -0.05 and β is set to 0.05.

C. Goal-Oriented Requirements Reasoning

The goal-oriented requirements reasoning involved in our method includes the quality goal reasoning algorithm and the functional reasoning algorithm.

The quality goal reasoning algorithm is triggered to relax the satisfaction levels of some desired soft goals to improve other more crucial but conflicting soft goals when the earned business value decreases by a certain degree (P1). The algorithm takes as input the given goal model and the tuned preference ranks of the soft goals, and has an iterative step to invoke a SAT solver for finite times. It tries initially to find a configuration that can accommodate all soft goals to reach their expected satisfaction levels. If not possible, the lowest ranked soft goals will be unbound so as to accommodate the remaining soft goals, and so on, until either a viable configuration is found or all soft goals have been unbound. It terminates with a valid configuration because we assume the root hard goal is satisfied by design. Detailed descriptions about the quality goal reasoning algorithm can be found in [8].

The functional goal reasoning is triggered in two cases: (1) the earned business value decreases by a certain degree, but the quality goal reasoning finds no new configuration (P2); (2) the earned business value increases by a certain degree (P3). In the former situation, it finds and unbinds a hard goal to release some resources for ensuring the crucial goals. In the latter situation, it finds and binds a hard goal to improve the functional integrity of the Web system. Thus, the "when" part of the challenges has been resolved.

As to the "which" part, it should consider the final influence to the whole system, i.e., the root goal by propagating the relative parent values bottom-up. Remembering that relative parent value is defined relative to the parent goal rather than to the root goal, we define *relative root value* of a hard goal as the product of the relative parent values in the path from the hard goal to root goal. For example, the relative root value of **Make Review** is 0.1; the relative root value of **Rich Form** is 1.0 * 1.0 * 0.8 * 1.0 = 0.8. Thus, relative root value characterizes the relative value contribution to the root goal from the business perspective.

Considering the desired functional service, crucial functional service and their mappings to the goal model, the desired goal and crucial goal are formally defined as follows:
Desired Goal and Crucial Goal: Desired goal and crucial goal are leaf hard goals. If a leaf hard goal's relative root value is 1.0, it's a crucial goal (1); otherwise, if its parent goal's relative root value is 1.0 and its parent goal has only one sub-goal at the time of consideration, it's a crucial goal (2); otherwise, it's a desired goal (3).

By these definitions, the determination of desired or crucial goal is dynamic. In Fig. 4 (a), **Goal2** is a crucial goal corresponding to case (1); **Sub1** and **Sub2** are desired goals corresponding to case (3). After **Sub2** is unbound (Fig. 4 (b)), **Goal2** is still a crucial goal corresponding to case (1). However, **Sub1** becomes a crucial goal corresponding to case (2) because **Goal1**, which has to be achieved for achieving **Root**, won't be achieved if **Sub1** is unbound. In our example, the desired goals are **Customer Consultation**, **Make Review**, **Customer Reviews**, **Plain Form**, **Rich Form**, and **Recommended Products**.

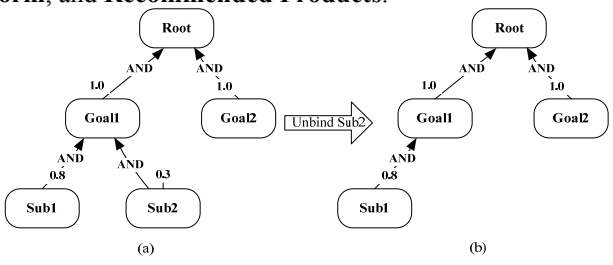

Figure 4. Example of desired and crucial goal.

Intuitively, the challenge of which goal to unbind can now be resolved by finding the bound desired goal with minimal relative root value, by Algorithm 2. Remember that only one of the OR-decomposed sub-goals is bound. And the challenge of which goal to bind can now be simply resolved by binding the recently unbound desired goal. E.g., if

initially **Rich Form** is selected for **Basic Information** and never changed, a possible order of unbinding and binding is: unbind **Recommended Products** (0.08), unbind **Make Review** (0.1), bind **Make Review (**0.1**)**, unbind **Make Review** (0.1), unbind **Customer Reviews** (0.24), unbind **Customer Consultation** (0.3), and unbind **Rich Form** (0.8).

Algorithm 2. Find the Minimal Bound Desired Goal to Sacrifice for the Survival of Root Goal.
input: the goal model and its hard goal set *goals*
output: a desired goal with minimal relative root value
procedure
begin
1. *minGoal* = null; *minValue* = 1.0;
2. *iterator* = get the iterator of *goals;*
3. **while** *iterator* has next **do**
4. *g* = get *iterator*'s next; *p* = get *g*'s parent goal;
5. **if** *g* is leaf goal **and** *g* is bound **and** *p* is **not** null **and not** (*p*'s relative root value **equal** 1.0 **and** *p* has one sub-goal) **then**
6. **if** *g*'s relative root value < minValue **then**
7. *minGoal* = *g*;
8. *minValue* = get *g*'s relative root value;
9. **end if**
10. **end if**
11. **end do**
12. **return** *minGoal*
end

It is worth noting that we define relative root value based on relative parent value rather than directly give the relative root value. Annotating a goal model with relative parent values along with the refinement of the model, one can describe the underlying value contributions between hard goals and their parent goals more precisely. In addition, it is easier to update the relative root values through updated relative parent values to reflect the actual status of the changing business environment (e.g., sales promotion at holidays). Currently, we only consider the fixed relative parent value, so the relative root values can be obtained on the fly.

D. Goal Model Reconfiguration

Using the quality goal reasoner on a given goal model, a reconfiguration is obtained, where the leaf goals will be mapped to components of the runtime architecture and the selection of the OR-decomposition goals are mapped to the switch connectors to allow for runtime reconfigurations by simply unbinding or binding the components in relation to that of the parent goal [19]. For example, if the current configuration is **Fuzzy Search**, **Rich Form** and **UnlPay**, the new configuration is **Accurate Search**, **Plain Form** and **UnlPay**, then **Fuzzy Search** and **Rich Form** are unbound goals while **Accurate Search** and **Plain Form** are bound [8].

Unlike the quality goal reasoner, the functional goal reasoner produces a reconfiguration that is a selection of a desired goal to be unbound or bound when the quality goal reasoner cannot produce any feasible reconfiguration. Considering that the bind process is the inverse of the unbind process, for brevity here we only discuss the unbind process (Algorithm 3). The algorithm first unbinds the desired goal which is the output of Algorithm 2. Then Algorithm 3 either terminates, or recursively unbinds its parent goal in the one of the following two cases: (1) the desired goal's relative parent value equals 1.0 which means that the parent goal cannot be achieved as long as the desired goal is unbound (Line 4-6). In this case, all its parent goal's sub-goals are also unbound because they are no longer necessary; (2) the desired goal is the only sub-goal of its parent goal (Line 8-10).

Algorithm 3. Unbind (g, G).
input: goal model *G* and a desired goal *g*
output: *g* is unbound from the goal model
procedure
begin
1. unbind *g* from *G*;
2. *p* = get the parent goal of *g*;
3. **if** *p* is **not** *null* **then**
4. **if the** relative parent value of *g* = 1.0 **then**
5. unbind all the other sub-goals of *p*;
6. Unbind (*p*, *G*);
7. **else**
8. **if** *g* is the only sub-goal of *p* **then**
9. Unbind (*p*, *G*);
10. **end if**
11. **end if**
12. **end if**
end

In Fig. 5 (a), if **Sub1** or **Sub2** is the output of Algorithm 2, then **Sub1**, **Sub2**, and **Goal1** should all be unbound according to case (1). In Figure 5 (b), assuming that **Sub1** has been unbound before and currently **Sub2** is in the output of Algorithm 2, then **Goal1** should also be unbound according to case (2).

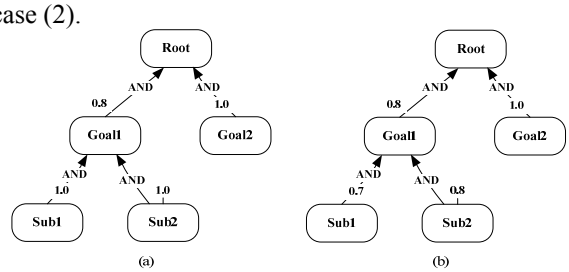

Figure 5. Examples of the unbind process.

V. EXPERIMENTAL STUDY

This section presents an experimental demonstration of the effectiveness of our method using the online shopping system introduced in Section II.

A. Experimental Settings

In the experiment, we simulated a continuous running of the system with different concurrent loads. The stress testing tool JMeter 2.3.4 was used to simulate concurrent accesses to the system. The JMeter test plan was recorded by Badboy 2.1. Several data files such as Customer ID and Password Data File, Search Keywords Data File and Products ID Data File were involved in the JMeter test plan to stimulate the real-life process of online shopping. In addition, the availability of two payment services was stochastically distributed with different density to stimulate their failure rate. The experiments were conducted on a ThinkPad R400 laptop with 2 Intel Core2 Duo 2.53 GHz processors and 2GB RAM, running Windows XP.

Analyzing the database transactions and applying the formula in Section III.A obtained the earned business value in terms of profits. Moreover, system and Web server log files were analyzed at runtime to capture runtime qualities like response time, usability, cost, and availability.

We conducted the experiments with three kinds of methods using the same experimental setting:

(1) *Static*: this is the naive method corresponding to design-time static quality tradeoff decisions, which has a fixed configuration for all variation points. In our experiment, **Search Products** is configured to **Fuzzy Search**, **Pay Bill** is configured to **AbbPay**, and **Basic Information** is configured to **Rich Form**.

(2) *Quality Reasoning*: this is the method we proposed in our earlier work [8], which makes quality tradeoff decisions adaptively through quality goal reasoning, without dealing with functional tradeoff decisions.

(3) *Quality and Functional Reasoning-**: this is the method we proposed in this work. The * specifies the window of timed delay with 0 representing no timed delay, 1 representing one minute of timed delay and 2 representing two minutes of timed delay.

B. Main Results and Comparisons

In each experiment with one of the three kinds of methods, we collected and analyzed the average earned business value and the four qualities per minute in a continuous running of about 77 minutes with different numbers of concurrent users varying from 10 to 90. The results of this experiment are shown in Fig.6 and Fig. 7.

We can observe from Fig. 6 that the optimal concurrent user number with maximum earned business value for the current Web system is about 30. In a range until this load, the three methods have almost the same level of earned business value. When the load exceeds this optimal load, the earned business value per unit time declines gradually, and the quality and functional reasoning-1 method outperforms both the static and quality reasoning methods significantly. The quality and functional reasoning-2 mostly outperforms the static and quality reasoning method, and the quality and functional reasoning-0 mostly outperforms the *static* method and partially outperforms the quality reasoning method.

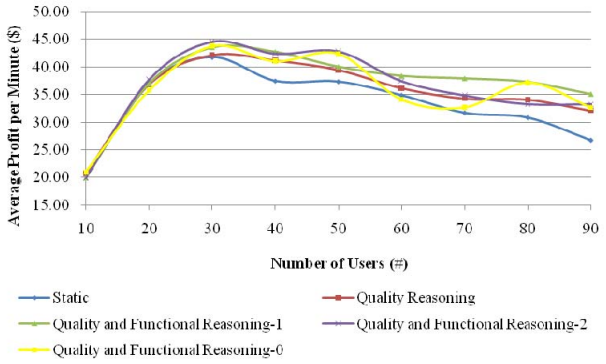

Figure 6. Experimental results with the earned business value.

Numerically, Table II shows the means and the standard deviations of profit (after reaching the optimal level). The quality and functional reasoning-* methods have a larger average profit than static and quality reasoning methods. Specially, the quality and functional reasoning-1 method has an average profit gain of 11.2% and 4.7% over static and quality reasoning method respectively. The standard deviations confirm that the curve of quality and functional reasoning-1 is smoother that the others.

TABLE II. MEANS AND STANDARD DEVIATIONS OF PROFIT.

Methods	Mean ($)	Standard Deviation
Static	33.16	5.02
Quality Reasoning	35.24	3.90
Quality and Functional Reasoning-0	35.61	4.73
Quality and Functional Reasoning-1	36.88	3.02
Quality and Functional Reasoning-2	36.25	4.81

Fig. 7 shows the results of two groups of conflicting qualities under different concurrent accesses: the response time and the usability of the system, the cost paid to third-party payment service providers and their availability. Here, we only compare the quality and functional reasoning-1 method with static and quality reasoning method. Discussion about the quality and functional reasoning method with different window of timed delay will be presented later. Adopting the quality and functional reasoning-1 method, the response time is significantly improved compared to static method and slightly improved compared to quality reasoning method; however, the usability is a little reduced but is still acceptable since the usability for static, quality reasoning and quality and functional reasoning-1 method are 10.0, 8.99 and 9.01 respectively. Similarly, availability is improved while cost is increased. And these are in accord with one of our motivations: the satisfaction levels of some quality requirements can be relaxed to ensure the acceptable satisfaction levels of other conflicting quality requirements. Above all, the earned business value is increased.

C. Evaluation of the Detailed Results

From Fig. 6, one can see that our method has a notable improvement over static and quality reasoning-only methods. In order to further evaluate the effectiveness of our method in details, we also investigate the self-tuning process of our method. Fig. 8 records the self-tuning process of our method when there are 70 concurrent users and the window of timed delay is one minute. On the curve, the reconfigurations, including selecting OR-decomposition goals, unbinding a desired goal and binding a desired goal, are identified as red points, yellow points and black points respectively. The *X* axis denotes discrete time intervals of one minute while the *Y* axis denotes profit in each time interval.

Let's first look at the self-tuning process for the three variation points through quality goal reasoning. There are four tuning scenarios in our example:

(1) **Search Products** and **Basic Information** are reconfigured whilst **Pay Bill** is not. E.g., at time 14 (profit: $25.91, response time: 8985ms, usability: 10.0, cost: $19.5, and availability: 0.87), the profit decreases because the response time is too long (at time 13, profit: $40.41, response time: 4605ms, usability: 10.0, cost: $34.50, and availability: 0.70), which drives the reconfiguration from **Fuzzy Search** to **Accurate Search** and from **Rich Form** to **Plain Form**. And this is consistent with the goal model, in which **Fuzzy Search** and **Rich**

Form have a Hurt contribution to **Response Time** and **Accurate Search** and **Plain Form** have a Help contribution to **Response Time**.

(2) **Pay Bill** is reconfigured whilst **Search Products** and **Basic Information** are not, which is similar to (1). The two scenarios are marked as red diamonds in Fig. 8.

(3) **Search Product**, **Basic Information** and **Pay Bill** are all reconfigured, marked as red triangles in Fig. 8, which shows none of the current configuration of the variation points is suitable for the current environment.

(4) None of the **Search Product**, **Basic Information** and **Pay Bill** is reconfigured, marked as yellow circles in Fig 8, which means quality tradeoff decisions cannot achieve the survivability objective. Thus, unbinding a desired goal is the tuning action followed.

Furthermore, we can see that all the above four tuning scenarios happen when profit decreases. And in most cases, the profit will increase after the tuning action (e.g., at time 11). However, if the profit continues to decrease after the tuning action, another tuning action is followed (e.g., at time 24 and 25) to drive the profit to increase.

Now let's look at the self-tuning process for unbinding or binding desired goals through functional goal reasoning. There are two tuning scenarios:

(1) A desired goal is unbound, which is following the previous self-tuning scenario (4). In most cases, the tuning scenario happens when profit decreases. Besides, the tuning scenario happens infrequently when profit increases (e.g., at time 55 and 73) due to the timed delay (details will be discussed in Section V.D).

(2) A desired goal is bound, marked as black circles in Figure 8. In most cases, the tuning scenario happens when profit increases. However, the tuning scenario infrequently happens when profit decreases (e.g., at time 75) due to the timed delay (details will be discussed in Section V.D).

Interestingly, profits will mostly increase right after unbinding action, and will increase in the timed delay after binding action. This reveals that the binding needs more time to take effect to the system.

D. Discussion

1) The Window of Timed Delay

Fig. 6 and Table II show that quality and functional goal reasoning with 1-minute of timed delay produces the highest profit and the smoothest QoS under online shopping loads.

To evaluate the effect of different windows of timed delay, four qualities are contrasted. It is obvious in Fig. 7 that one minute of timed delay gives the best average response time, comparing with no timed delay and two minutes of timed delay methods. In terms of average usability, one minute of timed delay also scores the best (9.01) compared with no timed delay (8.97) and two minutes delays (8.72).

Regardless of the delay parameter, on the other hand, all the three adaptive quality and function reasoning methods experimented have shown better average availability (0.77, 0.79, 0.78 respectively) than static (0.69) and quality reasoning (0.75) methods. However, it is also obvious in Fig. 7 that they cost more than static and quality reasoning method.

Consider the frequency of unbinding or binding actions, there are totally 29, 19 and 20 times of unbinding or binding actions happened in the 77 minutes respectively with no timed delay, one minute of timed delay and two minutes of timed delay. We also found an oscillation of the tuning action with no timed delay. Above the previous two observations, one minute of timed delay is sufficient for eliminating the oscillation and making the tuning timely effective.

Consider the effect of timed delay shown in Fig. 8. At the 62nd minute, a binding action happened, however profit decreased at the 63rd minute. If there was no timed delay, an unbinding action would have happened. But with one minute of timed delay, nothing would have happened. At the 64th minute, profit increased and another binding action happened, which proves that no action at time 63 would be correct. Similar situations are found between time 49 and 51, 55 and 59, 66 and 68.

On the other hand, one minute of timed delay maybe also ignore the really-necessary tuning actions. For example, a binding action happened at time 53 and profit decreased at time 54. If there were no timed delay, an unbinding action would have happened. But there was one minute of timed delay, no unbinding action happened. At time 55, profit increased a little and an unbinding action still happened, which proves that no unbinding action at time 54 is incorrect. Similar situations happened between time 71 and 75. However, timed delay is effective in most cases.

2) Resource versus Relative Root Value

When deciding which desired goal to be unbound or rebound, we only consider the relative root value, without taking into account the resources it holds and consumes. However, it seems to be a better way to select the desired goal with both low relative root value that consume large resources to be unbound first and the desired goal with high relative root value that consume small resources to be bound. A challenge here is to precisely characterize the resources usages for tradeoff decisions between relative root values and resources, which will be considered in our future work.

3) Performance Evaluation

To evaluate the performance of our goal-oriented reasoning algorithm with respect to the growth of the sizes of the goal model, we conducted a series of experiments with randomly generated goal models whose sizes vary from 25 to 150. The results have shown that the quality goal reasoning algorithm is performed in less than three seconds and Algorithms 2 and 3 are performed in less than one second for the size of 150. This suggests that our methods can be effectively applied in real-life Web applications.

VI. RELATED WORK

A. Requirements-Driven Self-Tuning. This work falls in the area of requirements-driven self-managing systems [20]. A desired self-managing system has the capabilities of self-reconfiguring [21], self-healing [22], self-optimizing [23] or self-protecting [24]. Although a few requirements-driven self-reconfiguration methods have been proposed [8, 21], it is our belief that turning some functionalities into tuning parameters is a mandatory for self-tuning.

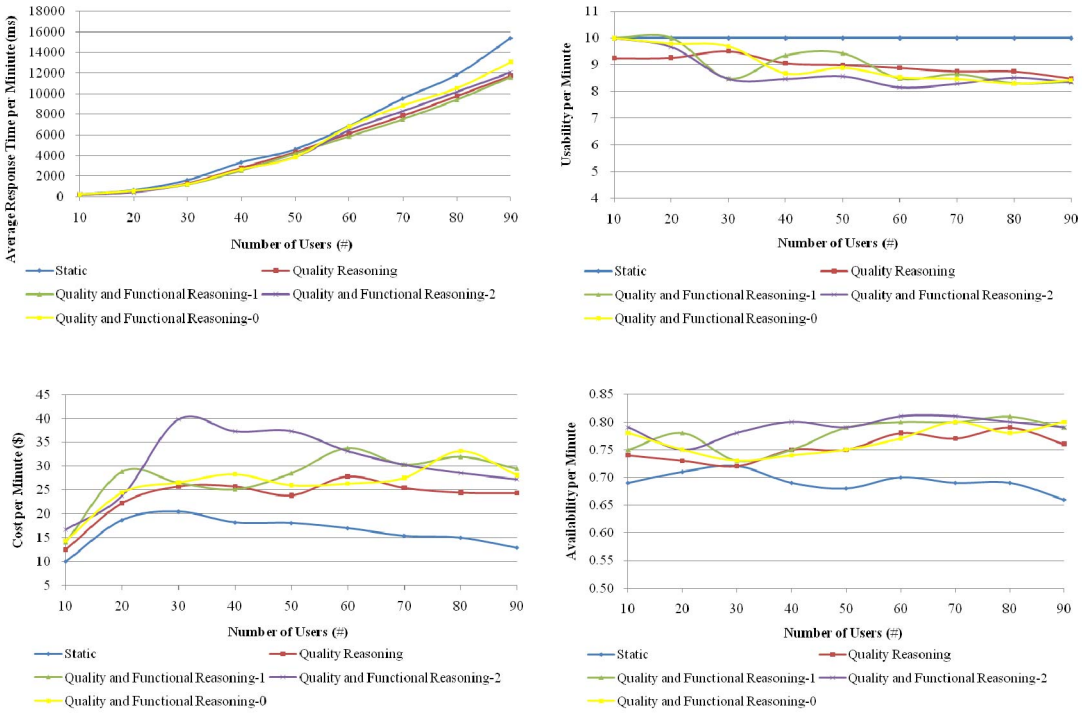

Figure 7. Experimental results with the four qualities.

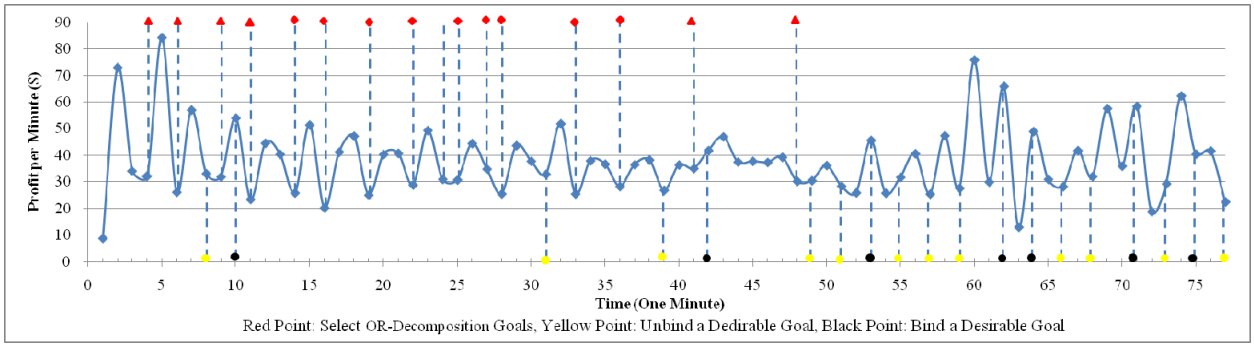

Figure 8. The self-tuning process (profit vs. time).

B. Reasoning on NFRs. For the selection of alternatives, NFRs have been used to perform both qualitative and quantitative goal reasoning [25]. Such goal reasoning methods suffer from lacking accurate goal formulations and propagation rules for domain-specific metrics; therefore such goal reasoning methods were suggested to be done partially [26]. In addition, NFRs can be expressed in probabilistic models [27] facilitating the reasoning as the system evolves. Extending the SAT-based reasoning method that allows more flexible preference-based encoding for soft goals [8], we propose a new method to select hard goals by differentiating desired from crucial.

C. Optional Goal. Recently, it is first proposed in a qualitative way to handle optional goals whose achievement is desired [13-14]. In this work, we interpret optional goals quantitatively to facilitate functional goal reasoning.

D. RE for Self-Adaptive Systems. How to tame uncertainty of the environment is still a key challenge for developing self-adaptive systems [28]. Recently, RELAX [29] is proposed as a formal language to express uncertain requirements; FLAGS [30] is proposed as a goal model extension to distinguish crisp and fuzzy goals. They both adopt fuzzy logic, which naturally needs to use precise membership functions for evaluating the satisfaction levels. In complementary to these approaches, we further consider sacrificing desired requirements to ensure crucial ones.

E. Value-Based SE and RE. Our method shows that it is possible to integrate business value considerations into the goal models [11]. Value-based RE framework [12] has been applied to model the business requirements for large finance organizations [31] and for Web applications [10]. In this work, value-based requirement analysis is used not only to differentiate desired and crucial goal, but also to guide the tuning process at runtime.

F. Survivability. Our work is closely related to those for system survivability [3], which addresses the survivability under contexts such as system damages or software failures. However, we take a value perspective that concerns more about optimising the satisfaction levels of quality requirements for a system value proposition.

G. Quality of Service Management. Many works have addressed the problem of Web Systems performance management under unpredictable workloads [32-33]. Hill climbing techniques [32] and feedback control theory [33] are used to configure low-level parameters, such as the MaxClients of a server, in order to manage generic qualities such as throughput. Our work here includes more domain-specific quality measures that concern the high-level requirements.

VII. CONCLUSIONS AND FUTURE WORK

With value-based interpretation of survivability assurance in two levels, a requirements-driven self-tuning method for survivability assurance of Web systems has been proposed to tune both quality and functional tradeoff decisions at runtime through the PID controller and the goal-oriented reasoning algorithms. Through an online shopping system case study with realistic transactional data, we have validated the effectiveness of our method by comparing quantitatively with the ad-hoc and pure quality tuning alternatives.

For future work, we intend to investigate the timed delay and assess its precise impact more closely, and explore some control mechanisms that offer potentially better stability through the complex system theory. We also intend to apply the method to the PaaS (platform as a service) infrastructure in order to provide value-based survivability assurance for business applications in the cloud.

REFERENCES

[1] J. Offutt, "Quality attributes of Web software applications," IEEE Software, vol. 19 no. 2, 2002.

[2] "Amazon.com experiences hours-Long outage," Website 2010. http://news.cnet.com/8301-1023_3-20009241-93.html.

[3] J. C. Knight, and E. A. Strunk, "Achieving critical system survivability through software architectures," Architecting Dependable Systems II, Springer, 2004.

[4] R. Kazman, M. Klein, and P. Clements, "ATAM: method for architecture evaluation," 2000.

[5] IBM, "An architectural blueprint for autonomic computing," Technical Report, 2003.

[6] M. C. Huebscher, and J. A. McCann, "A survey of autonomic computing - degrees, models, and applications," ACM Computing Surveys, vol. 40, no. 3, 2008.

[7] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola, "QoS-driven runtime adaptation of service oriented architectures," The ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2009.

[8] X. Peng, B. Chen, Y. Yu, and W. Zhao, "Self-tuning of software systems through goal-based feedback loop control," The 18th IEEE International Requirements Engineering Conference, 2010.

[9] Y. Yu, Y. Wang, J. Mylopoulos, S. Liaskos, A. Lapouchnian. and J. C. S. do Prado Leite, "Reverse engineering goal models from legacy code," The 13th IEEE International Conference on Requirements Engineering, 2005.

[10] F. Azam, Z. Li, and R. Ahmad, "Integrating value-based requirement engineering models to WebML using VIP business modeling framework," The 16th International Conference on World Wide Web, 2007.

[11] B. Boehm, "Value-based software engineering," ACM SIGSOFT Software Engineering Notes, vol. 28, no. 2, 2003.

[12] J. Gordijn, "Value-based requirements engineering: exploring innovative e-commerce ideas," Vrije University, 2002.

[13] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling," The 18th IEEE International Requirements Engineering Conference, 2010.

[14] S. Liaskos, S. A. McIlraith, S. Sohrabi, and J. Mylopoulos, "Integrating preferences into goal models for requirements engineering," The 18th IEEE International Requirements Engineering Conference, 2010.

[15] A. Deaton, and J. Muellbauer. Economics and Consumer Behavior, Cambridge, UK: Cambridge University Press, 1980.

[16] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani, "The Fractal component model and its support in Java," Software: Practice and Experience, vol. 36, no. 11-12, 2006.

[17] S. W. Cheng, and D. Garlan, "Rainbow: cost-effective software architecture-based self-adaptation," Carnegie Mellon University, 2008.

[18] G. F. Franklin, J. D. Powell, and A. E. Naeini, Feedback Control of Dynamic Systems, Upper Saddle River, NJ, USA: Prentice-Hall, 2006.

[19] Y. Yu, A. Lapouchnian, S. Liaskos, J. Mylopoulos, and J. C. S. P. Leite, "From goals to high-variability software design," The 17th International Conference on Foundations of Intelligent Systems, 2008.

[20] B. H. C. Cheng, R. de Lemos, D. Garlan, H. Giese, M. Litoiu, J. Magee, H. A. Muller, and R. Taylor, "SEAMS 2009: software engineering for adaptive and self-managing systems," The 31st International Conference on Software Engineering, 2009.

[21] F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "Software self-reconfiguration: a BDI-based approach," The 8th International Joint Conference on Autonomous Agents and Multiagent Systems 2009.

[22] M. Salifu, Y. Yu, and B. Nuseibeh, "Specifying monitoring and switching problems in context," The 15th IEEE International Requirements Engineering Conference, 2007.

[23] S. M. Sadjadi, P. K. McKinley, R. E. K. Stirewalt, and B. H. C. Cheng, "Generation of self-optimizing wireless network applications," The 1st International Conference on Autonomic Computing, 2004.

[24] L. Chung, "Dealing with security requirements during the development of information system," Advanced Information Systems Engineering, 1993.

[25] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Formal reasoning techniques for goal models," Journal on Data Semantics I, vol. 1, 2003.

[26] E. Letier, and A. v. Lamsweerde, "Reasoning about partial goal satisfaction for requirements and design engineering," The 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2004.

[27] C. Ghezzi, and G. Tamburrelli, "Reasoning on non-functional requirements for integrated services," The 17th IEEE International Requirements Engineering Conference, 2009.

[28] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-aware systems: a research agenda for RE for self-adaptive systems," The 18th IEEE International Requirements Engineering Conference, 2010.

[29] J. Whittle, P. Sawyer, N. Bencomo, and B. H. C. Cheng,"RELAX: incorporating uncertainty into the specification of self-adaptive systems," The 17th IEEE International Requirements Engineering Conference, 2009.

[30] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy goals for requirements-driven adaptation", The 18th IEEE International Requirements Engineering Conference, 2010.

[31] J. Gordijn, E. Yu, and B. van der Raadt, "E-service design using i* and e3 value modeling," IEEE Software, vol. 23, 2006.

[32] D. A. Menascé, D. Barbará, and R. Dodge, "Preserving QoS of e-commerce sites through self-tuning: a performance model approach," The 3rd ACM Conference on Electronic Commerce, 2001.

[33] A. Kamra, V. Misra, and E. M. Nahum, "Yaksha: a self-tuning controller for managing the performance of 3-tiered Web sites," The 12th IEEE International Workshop on Quality of Service, 2004.

[34] B. Chen, X. Peng, Y. Yu, and W. Zhao, "Survivability-oriented self-tuning of Web systems," The 20th International Conference on World Wide Web, 2011. Poster, in press.