View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Open Research Online

Open Research Online

The Open University's repository of research publications
and other research outputs

Uncertainty handling in goal-driven self-optimization – limiting the negative effect on adaptation

Journal Item

How to cite:

Chen, Bihuan; Peng, Xin; Yu, Yijun and Zhao, Wenyun (2014). Uncertainty handling in goal-driven self-optimization – limiting the negative effect on adaptation. Journal of Systems and Software, 90 pp. 114–127.

For guidance on citations see FAQs.

© 2014 Elsevier Inc.

Version: Accepted Manuscript

Link(s) to article on publisher's website:
http://dx.doi.org/doi:10.1016/j.jss.2013.12.033

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online's data policy on reuse of materials please consult the policies page.

oro.open.ac.uk

Uncertainty Handling in Goal-Driven Self-Optimization – Limiting the Negative Effect on Adaptation

Bihuan Chen[a,b], Xin Peng[a,b,*], Yijun Yu[c], Wenyun Zhao[a,b]

[a] *School of Computer Science, Fudan University, Shanghai 200433, China*
[b] *Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 200433, China*
[c] *Department of Computing, The Open University, Milton Keynes, UK*

Abstract

Goal-driven self-optimization through feedback loops has shown effectiveness in reducing oscillating utilities due to a large number of uncertain factors in the runtime environments. However, such self-optimization is less satisfactory when there contains uncertainty in the predefined requirements goal models, such as imprecise contributions and unknown quality preferences, or during the switches of goal solutions, such as lack of understanding about the time for the adaptation actions to take effect. In this paper, we propose to handle such uncertainty in goal-driven self-optimization without interrupting the services. Taking the monitored quality values as the feedback, and the estimated earned value as the global indicator of self-optimization, our approach dynamically updates the quantitative contributions from alternative functionalities to quality requirements, tunes the preferences of relevant quality requirements, and determines a proper timing delay for the last adaptation action to take effect. After applying these runtime measures to limit the negative effect of the uncertainty in goal models and their suggested switches, an experimental study on a real-life online shopping system shows the improvements over goal-driven self-optimization approaches without uncertainty handling.

Keywords: uncertainty, goal-driven self-optimization, requirements goal models, earned value

*Corresponding author. Tel.: +86 21 51355343
Email addresses: bhchen@fudan.edu.cn (Bihuan Chen), pengxin@fudan.edu.cn (Xin Peng), y.yu@open.ac.uk (Yijun Yu), wyzhao@fudan.edu.cn (Wenyun Zhao)

1. Introduction

Self-optimization enables a system to continually seek opportunities to improve its runtime qualities by adapting its structure and behavior (Kephart and Chess, 2003). With well-understood alternative functionalities and their different contributions to relevant quality requirements, requirements goal models (Mylopoulos et al., 1992), where functional and quality requirements are respectively modeled as hard and soft goals, have been used to reason about the optimal reconfigurations for a running system (Lapouchnian et al., 2005; Wang and Mylopoulos, 2009; Welsh et al., 2011; Peng et al., 2012). A quality attribute is correlated to a soft goal to measure its satisfaction based on the corresponding quality value (Letier and van Lamsweerde, 2004). The reconfigurations are often triggered periodically at a fixed time interval, or by the violations of the predefined constraints on the quality attributes. Mostly, when goal reasoning (Giorgini et al., 2002; Sebastiani et al., 2004) is used in decision making for self-optimization, the goal models are usually predefined with the assumption that they are perfectly understood and will not frequently change at runtime, and the self-optimization is often conducted with predefined adaptation rules (Baresi et al., 2010).

However, the changing environments may bring diverse uncertainty as compounding factors of goal models. For instance, *contribution uncertainty*, i.e., different operationalizations contribute to soft goals at an uncertain level due to imperfect knowledge about the runtime environments (Welsh et al., 2011), may lead to suboptimal reconfigurations. *Preference uncertainty*, i.e., the relative importance of soft goals for a system may be uncertain under different runtime environments (Peng et al., 2012; Liaskos et al., 2010), affects the results of goal reasoning when not all the desired soft goals can be fully satisfied. *Effect uncertainty*, i.e., the timing delay for the switches of goal solutions, or the adaptation actions, to take effect in the running system may be uncertain (Cheng, 2008), may trigger a premature reconfiguration before the last one has taken effect and thus cause the problem of oscillation. Such negative effect resulting from the uncertainty will make the self-optimization less satisfactory.

Notice that contribution and preference uncertainty lie in the goal models, and effect uncertainty originates from the switches of goal solutions. All of them can be regarded as uncertainty at design time as well as uncertainty at runtime, because contributions, preferences and timing delays are hard to be precisely specified at design time, and they may also change under different

runtime environments.

While design-time uncertainty handling in requirements has been widely addressed through better approaches in elicitation, disambiguation and inconsistency check (Liaskos et al., 2012; Yang et al., 2012; Arora et al., 2012), these approaches more or less involve human intervention. Hence, runtime adaptations through unsupervised feedback loops present little time to apply them, while quickly responding to changes in the environments.

On the other hand, considering alternative system configurations, it is common that quality requirements conflict with each other and the configuration better in certain quality dimensions can be worse in others (Clements et al., 2001). However, predefined adaptation rules (e.g., if average response time is larger than a threshold, then switch from multimedia mode to textual mode) usually do not involve the full facets but a single or several facet(s) of the quality requirements. Hence, such rules are not flexible enough and may trigger premature reconfigurations.

Therefore, one problem with goal-driven self-optimization is that, if the understanding of the goal models and their suggested switches contains uncertainty, the resulting adaptations will be less satisfactory. In principle, it is possible to limit their negative effect by adapting the goal models and their suggested switches at runtime, which is also known as models@run.time (Blair et al., 2009; Morin et al., 2009). The other problem is that adaptation rules of self-optimization only capture partial facets of the quality requirements and are difficult to be fully predefined. In practice, a global indicator should be used to comprehensively guide the self-optimization process.

To address such problems, in this paper, we propose a combined approach to handle diverse uncertainty in goal-driven self-optimization. Taking the monitored quality values as the feedback, and the estimated earned value as the global indicator of self-optimization, our approach handles the three kinds of uncertainty to limit their negative effect. For contribution uncertainty, we conduct a probabilistic analysis at runtime to update the quantitative contributions from alternative functionalities to soft goals, based on a probabilistic model for the satisfaction and denial of goals proposed by Giorgini et al. (2002). For preference uncertainty, we integrate the dynamic quality tradeoff mechanism proposed in our previous work (Peng et al., 2012) to tune the preferences of relevant soft goals. For effect uncertainty, we introduce a heuristics-based technique to determine a proper timing delay for the last adaptation action to take effect.

To evaluate the effectiveness of the proposed approach, we conducted an

experimental study on a real-life online shopping system. The study analyzed and compared the earned value, relevant quality values and adaptation frequency of goal-driven self-optimization approaches with and without uncertainty handling. The study shows that, by combining the mechanisms for uncertainty handling, goal-driven self-optimization becomes more effective in terms of both earned value and stability.

The rest of the paper is structured as follows. Section 2 introduces the required preliminaries. Section 3 illustrates our motivation. Section 4 presents the proposed approach, including an overview and the techniques for handling the three kinds of uncertainty. Section 5 reports our implementation. Section 6 evaluates the proposal and makes some discussion. Section 7 introduces a number of existing proposals and compares them with ours before the conclusions in Section 8.

2. Preliminaries

Our approach is built on top of goal-oriented requirements modeling (Mylopoulos et al., 1992) and value-based software engineering (Boehm, 2003). In this section, we briefly explain the basics and then introduce the problem of uncertainty with soft goal satisfaction.

2.1. Goal-Oriented Requirements Modeling

In goal-oriented requirements analysis, functional requirements are modeled as *hard goals*, while quality requirements are modeled as *soft goals* (Mylopoulos et al., 1992). Goals can be refined into subgoals through AND/OR decompositions until reaching leaf-level *tasks* that can be accomplished by software components or human agents. To satisfy an AND/OR-decomposed goal, all/at least one of its subgoals must be satisfied. Hence, goal models capture the space of alternative goal solutions (i.e., sets of leaf-level tasks) satisfying the top-level goals in the form of OR-decompositions.

Following Giorgini et al. (2002), goals can be related to each other through quantitative contribution links, i.e., w+S, w−S, w+D, w−D, w+, w−, ++S, −−S, ++D, −−D, ++ and −−. The weight $w \in [0, 1]$ numerically expresses the strength by which the satisfaction/denial of the source goal contributes to the satisfaction/denial of the target goal. Here we adopt a probabilistic model proposed by Giorgini et al. (2002) to represent the evidence of the satisfaction/denial of a goal as the probability that the goal is satisfied/denied.

Hence, $G_1 \xrightarrow{w+S} G_2$ (respectively $G_1 \xrightarrow{++S} G_2$) can be interpreted as that, if the probability of the satisfaction of G_1 is p, then the probability of the satisfaction of G_2 is $w \times p$ (respectively $1 \times p$), i.e., $P(G_1\ is\ satisfied \wedge G_2\ is\ satisfied) = w \times P(G_1\ is\ satisfied)$; but if the probability of the denial of G_1 is p', then nothing can be inferred about the probability of the denial of G_2. $G_1 \xrightarrow{w-S} G_2$ (respectively $G_1 \xrightarrow{--S} G_2$) can be interpreted as that, if the probability of the satisfaction of G_1 is p, then the probability of the denial of G_2 is $w \times p$ (respectively $1 \times p$), i.e., $P(G_1\ is\ satisfied \wedge G_2\ is\ denied) = w \times P(G_1\ is\ satisfied)$; but if the probability of the denial of G_1 is p', then nothing can be inferred about the probability of the satisfaction of G_2. w+D, $w-$D, ++D and $--$D are respectively dual w.r.t. to w+S, $w-$S, ++S and $--$S by inverting satisfaction and denial, and thus their probabilistic interpretations are omitted. $w+$, $w-$, ++ and $--$ are respectively shorthand for w+S and w+D, $w-$S and $w-$D, ++S and ++D, and $--$S and $--$D.

Figure 1 depicts a goal model of an online shopping system, which is used to illustrate our approach throughout the rest of the paper. Hard goals, soft goals and tasks are shaped as rounded rectangles, clouds and hexagons. The tasks and soft goals are annotated with a symbol for the sake of reference simplicity. Each soft goal is annotated with a number (dashed circle) to represent stakeholders' preference over different quality requirements. This goal model reflects the following requirements: after searching the desired products, a customer can view the details about the interested products, and add the wanted products to the cart; the stock has to be checked before the customer places and pays the order.

Details Be Viewed, *Stock Be Checked* and *Out of Stock Items Be Ordered* are OR-decomposed goals. Product details can be viewed in textual mode (*t2*, e.g., only descriptive texts) or in multimedia mode (*t3*, e.g., texts and pictures). *t2* helps to reduce response time but hurts to enhance usability, and *t3* contributes to them in a reverse manner. Once the products in the shopping cart are out of stock, there are two available strategies: remove them (*t8*), or order them from retailers (*t6*) or wholesalers (*t7*). *t8* hurts to reduce order canceling rate because the customers are apt to cancel the order when some wanted products are removed. Reversely, *t6* and *t7* hurt to reduce ordering cost because extra orders have to be placed to retailers or wholesalers for out of stock products, with the difference being that *t6* hurts more than *t7* because retailers usually charge a higher price than wholesalers.

Based on these OR-decomposed goals, the following alternative goal so-

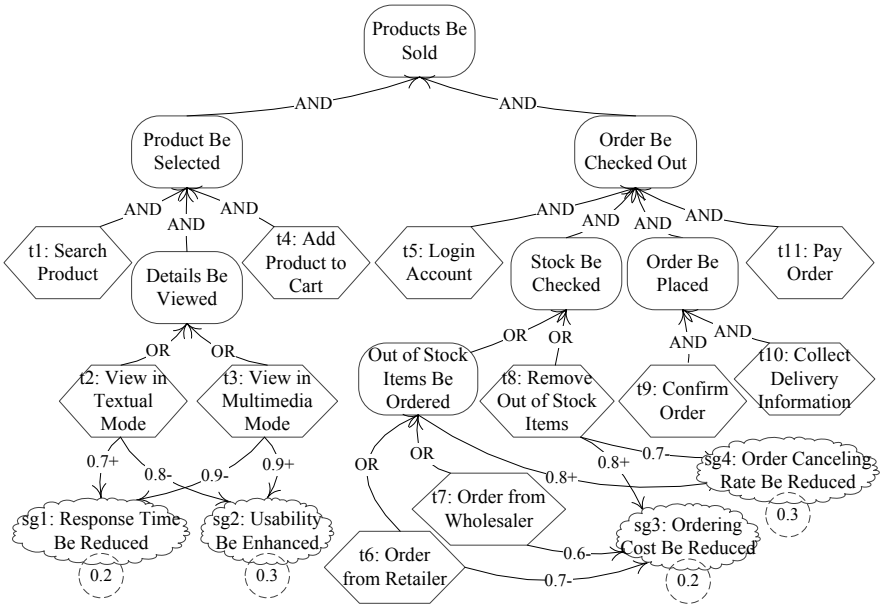

Figure 1: A Goal Model of an Online Shopping System.

lutions that satisfy the top-level goal *Product Be Sold* can be derived. The overall contribution of each solution to the soft goals is calculated as a score being the weighted sum of the satisfaction/denial levels of the soft goals. And the satisfaction/denial levels of the soft goals can be calculated by label propagation algorithms (Giorgini et al., 2002).

$$
\begin{aligned}
s1 &: [t1,\ t2,\ t4,\ t5,\ t7,\ t9,\ t10,\ t11],\ score\ =\ 0.02\\
s2 &: [t1,\ t3,\ t4,\ t5,\ t7,\ t9,\ t10,\ t11],\ score\ =\ 0.21\\
s3 &: [t1,\ t2,\ t4,\ t5,\ t6,\ t9,\ t10,\ t11],\ score\ =\ 0.00\\
s4 &: [t1,\ t3,\ t4,\ t5,\ t6,\ t9,\ t10,\ t11],\ score\ =\ 0.19\\
s5 &: [t1,\ t2,\ t4,\ t5,\ t8,\ t9,\ t10,\ t11],\ score\ =\ -0.15\\
s6 &: [t1,\ t3,\ t4,\ t5,\ t8,\ t9,\ t10,\ t11],\ score\ =\ 0.04
\end{aligned}
$$

Example 2.1. *For the solution s1, "View in Textual Mode" is chosen, which respectively has a 0.7+ and a 0.8− contribution link to "Response Time Be Reduced" and "Usability Be Enhanced". Hence, their satisfaction/denial levels are respectively 0.7/0.0 and 0.0/0.8. Similarly, the satisfaction/denial levels of "Ordering Cost Be Reduce" and "Order Canceling Rate Be Reduced" are respectively 0.0/0.6 and 0.8/0.0. Given the preferences to these*

soft goals as shown in Figure 1, the score of s1 is calculated as 0.2 × (0.7 − 0.0) + 0.3 × (0.0 − 0.8) + 0.2 × (0.0 − 0.6) + 0.3 × (0.8 − 0.0) = 0.02.

2.2. Value-Based Software Engineering

Value-based software engineering (VBSE) (Boehm, 2003) proposes to integrate value consideration into software engineering principles and practices. Different from traditional earned value systems based on cost and schedule estimation for project management, VBSE focuses on the real stakeholder value being earned and emphasizes the incorporation of business-value and mission-value achievement into feedback control systems. Earlier we have successfully applied the principles of VBSE in dynamic quality trade-offs (Peng et al., 2012) and runtime survivability assurance (Chen et al., 2011).

On the one hand, systems are usually designed to create value for stakeholders, and thus earned value is an appropriate indicator of self-optimization. On the other hand, compared with adaptation rules based on quality violations (e.g., if average response time is larger than a threshold, then switch from multimedia mode to textual mode), earned value reflects a global synthesis of the full facets of quality requirements. Hence, earned value here is used as the global indicator of self-optimization.

Further, an application-specific value proposition should be predefined to estimate the value earned from every successful transaction in a time interval (e.g., 30 seconds). Earned value estimation could be so complicated that it should be accomplished by business experts on the basis of business market analysis, risk analysis, business losing trend analysis, etc. We assume that a formula for earned value measurement can be predefined by business experts. For instance, the value proposition for the online shopping system can be defined as follows by considering both short-term and long-term influence factors.

$$\begin{aligned} value = sales \times 20\% - sales \times 5\% \ (or\ 6.5\%\ or\ 7.5\%) - \\ sales \times (1.0 - usability) \times 0.02 \end{aligned}$$

The first term means 20 percent of the sales are earned, which is influenced by the short-term factors such as response time and order canceling rate. The second term is the cost for processing the orders (e.g., stocking, transportation, etc.). Depending on the strategy for dealing with the out of stock products, 5, 6.5 and 7.5 percent of the sales are respectively expended for removing, ordering from wholesalers and ordering from retailers. The last

term takes into account the long-term factor usability because low usability will result in customer losing and it usually takes a long time to be reflected in earned value if not considered explicitly in the value proposition.

Example 2.2. *If sales and usability in a time interval is 200 dollars and 0.8, and the out of stock products are removed, then earned value is 29.2 dollars.*

2.3. Uncertainty with Soft Goal Satisfaction

Hard goals and tasks usually have clear-cut satisfaction criteria. For instance, the task *Pay Order* is satisfied when an order has been paid by a customer. On the other hand, soft goals usually have no clear-cut satisfaction criteria. For instance, it is difficult to definitely determine whether or not the soft goal *Response Time Be Reduced* is satisfied. Hence, it is reasonable to determine *to what extent* a soft goal meets its *criterion* rather than to determine whether or not a soft goal is satisfied.

A quality attribute can be identified for and correlated to a soft goal (Letier and van Lamsweerde, 2004). Thereby the satisfaction criterion of a soft goal can be seen as respecting the *expected* quality value. Then a soft goal is satisfied/denied to some extent means that the *actual* quality value is *roughly* larger/smaller (for positive quality attributes, i.e., the larger the better, e.g., usability) or smaller/larger (for negative quality attributes, i.e., the smaller the better, e.g., response time) than expected.

To handle such uncertainty with soft goal satisfaction, the theory of fuzzy sets (Zadeh, 1965) can be adopted to provide a gradual measurement for the satisfaction/denial level of soft goals (Baresi et al., 2010; Whittle et al., 2009). Specifically, a membership function is designed for each soft goal to assign a degree of truth to the satisfaction criterion. Figure 2 shows two membership functions for the fuzzy relational operators $\succeq$ and $\preceq$. The X axis denotes the actual value of a quality attribute, while the Y axis denotes the membership value (i.e., how closely the relation is respected). In the case of *Actual Value* $\succeq$ *Expected Value*, its membership function is absolutely true when the actual value is larger than the expected value, has a degree of truth between 0 and 1 when the actual value is smaller than the expected value and larger than 90% of the expected value, and is absolutely false when the actual value is smaller than 90% of the expected value. Based on such membership functions, the satisfaction level of a soft goal is a membership value s, whilst its denial level is $1.0 - s$.

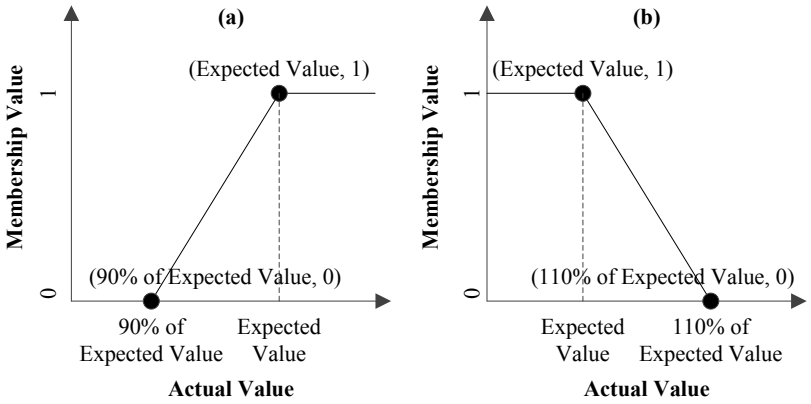

Figure 2: Membership Functions for Fuzzy Relational Operators: (a) Actual Value $\succeq$ Expected Value, (b) Actual Value $\preceq$ Expected Value.

Example 2.3. *The satisfaction level of "Response Time Be Reduced" can be measured by the membership function (b) in Figure 2. Given that the expected response time is 1000ms, "Response Time Be Reduced" is fully satisfied when the actual response time is smaller than 1000ms, has a degree of satisfaction between 0 and 1 when the actual response time is smaller than 1100ms and larger than 1000ms, and is fully denied when the actual response time is larger than 1100ms.*

3. Motivation

This section illustrates the motivation of handling the three kinds of uncertainty in goal-driven self-optimization.

3.1. Contribution Uncertainty

The weights of contribution links in a goal model are usually given by business analysts based on their assumptions on the runtime environments (Welsh et al., 2011). However, it is too difficult for a business analyst to precisely and quantitatively estimate the weight of a contribution link, and the uncertainty with the runtime environments may break their assumptions (Example 3.1). As a result, the weights of contribution links may be overestimated or underestimated. This kind of contribution uncertainty, if not properly handled, may cause goal reasoning produce suboptimal goal reconfigurations (Example 3.2).

Example 3.1. *An online shopping system usually orders out of stock products from wholesalers rather than from retailers based on the assumption that wholesalers usually provide lower prices than retailers. However, wholesalers sometimes may quote a higher price due to inventory drop and retailers may quote a lower price for sales promotion. Without updating the weights of relevant contribution links, the adaptation mechanism may make wrong decisions on choices of suppliers.*

Example 3.2. *Given the weights as shown in Figure 1, multimedia-mode product viewing will be chosen. However, if the weights of multimedia-mode and textual-mode product viewing to usability were respectively underestimated to 0.5+ and 0.5−, textual-mode product viewing would be chosen by goal reasoning because it has a higher contribution score to "Usability Be Enhanced" and "Response Time Be Reduced".*

3.2. Preference Uncertainty

The preferences of soft goals in a goal model are usually decided by business analysts through static trade-off decisions based on their assumptions on the runtime environments (Peng et al., 2012; Liaskos et al., 2010). However, due to the uncertainty with the runtime environments, the appropriate preference for each soft goal will change at runtime (Example 3.3). This kind of preference uncertainty, if not properly handled, may also cause suboptimal goal reconfigurations especially when not all the desired quality requirements can be satisfied.

Example 3.3. *In normal situations, usability is preferred to response time in order to provide attractive interfaces for customers, justifying multimedia mode being chosen for viewing product details. In a high-load scenario, however, response time should be given a higher preference over usability to ensure the availability of the functionality of viewing product details, thus making textual mode be chosen.*

3.3. Effect Uncertainty

In goal-driven self-optimization, goal reasoning is triggered when requirements are deviated, possibly adapting the requirements problem to an alternative solution and the system to a corresponding configuration. Ideally, the adaptation action should take effect immediately, i.e., the deviated requirements are satisfied on the spot. However, due to the additional processing

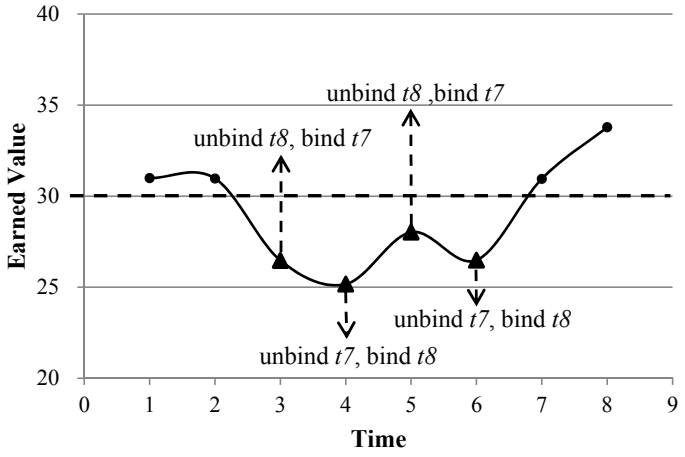

Figure 3: Oscillation of Adaptation Actions due to Effect Delay.

required for reconfiguration such as initialization and data migration, there is often a timing delay for an adaptation action to take effect in the running system (Cheng, 2008). As a result, the deviated requirements may still be violated or even get worse during the timing delay. Example 3.4 illustrates a common case for such effect delay. Further, Example 3.5, taken from our experimental study, illustrates its resulting oscillation problem from the aspect of earned value changes as shown in Figure 3.

Example 3.4. *The adaptation mechanism of the online shopping system decides to switch from textual mode to multimedia mode for the goal "Details Be Viewed". Then a series of architectural reconfigurations are enforced accordingly. A typical set of reconfigurations can be: firstly, some component instances for multimedia-mode product viewing are created and loaded; secondly, these new component instances are initiated, e.g., by assigning buffer and loading some initialization data; thirdly, the adaptation mechanism waits for the existing component instances for textual-mode product viewing to enter a safe mode and then deactivates them, and in the meantime new client requests are blocked and buffered; and finally, the component instances for multimedia-mode product viewing are activated and begin to process client requests. From the above process, it can be seen that it still takes some time for the optimization to take effect after an adaptation decision is made, during which the earned value may even be degraded (Example 3.5).*

Example 3.5. *It can be observed from Figure 3 that a deviation of earned value was detected at time 3 (the dashed line represented the expected earned value) and then the adaptation mechanism decided to replace t8 with t7. Due to the effect delay, the deviation of earned value got even worse at time 4, and the solution was changed back from t7 to t8. In the following time 5 and 6, similar oscillation continued, making the earned value stand at a low level.*

A simple way to handle effect delay is to designate a fixed timing delay for each adaptation action as suggested in (Cheng, 2008). However, due to the uncertain runtime environments, effect delay itself can also be uncertain, i.e., we cannot assume that an adaptation action will definitely take effect in a fixed length of timing delay. Hence, this kind of effect uncertainty, if not properly handled, may trigger a premature reconfiguration before the last one has taken effect, and thus cause oscillation.

4. Our Approach

In this section, we first give an overview of uncertainty handling in goal-driven self-optimization, and then present the techniques for handling the three kinds of uncertainty.

4.1. Overview of Uncertainty Handling

In order to handle the uncertainty underlying goal-driven self-optimization, our approach maintains a live goal model as the knowledge base for self-optimization, updates the weights of contribution links, tunes the preferences of soft goals, and determines a proper timing delay for the last adaptation action to take effect. Figure 4 depicts an overview of goal-driven self-optimization, which follows the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) control loop (Kephart and Chess, 2003), and highlights *where the uncertainty lies* (indicated by question marks) and *where the uncertainty is handled* (indicated by check marks).

In this overview, the *Monitor* step is achieved by using *Sensors* to regularly collect and aggregate runtime data in order to capture the properties that are of interest to self-optimization. The interested properties in our online shopping system are earned value and quality attributes such as response time, usability, order canceling rate and ordering cost. To automatically collect such data, predefined specific sensors should be first instrumented into the target system, which can be accomplished in an engineering or reengineering manner (Salehie and Tahvildari, 2009).

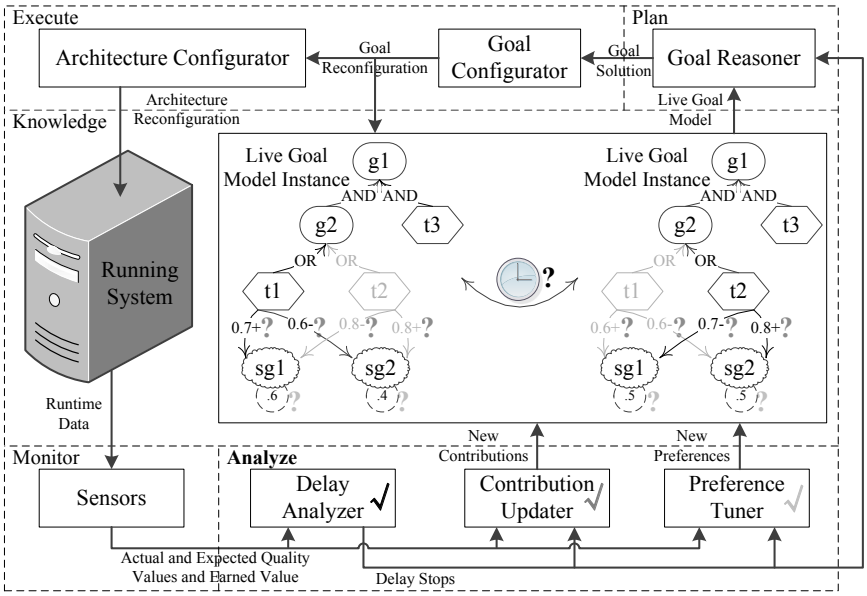

Figure 4: Overview of Goal-Driven Self-Optimization.

The *Analyze* step consists of *Delay Analyzer*, *Contribution Updater* and *Preference Tuner*, sub-steps that respectively handle effect uncertainty, contribution uncertainty and preference uncertainty on the basis of earned value and quality values. Notice that preferences and contributions will not be tuned and updated when the last adaptation action is still in its timing delay since the system is in a disturbance stage during the delay.

The *Plan* step is achieved by a quantitative *Goal Reasoner*, which is to find among all the goal solutions the one with the highest score (score calculation is introduced in Section 2.1). This reasoner is similar to the one proposed by Wang and Mylopoulos (2009) except that here we consider quantitative rather than qualitative contribution links.

The *Execute* step is realized by first using *Goal Configurator* to reconfigure the live goal model (specifically, the OR-decomposed goals) according to the planned goal solution and then using *Architecture Configurator* to map from goal reconfiguration to architecture reconfiguration of the running system.

Instead of detailedly exploring each step of the self-optimization control loop, we will put emphasis upon the *Analyze* step since in this paper we focus on uncertainty handling. In the following three sections, we will respectively introduce the handling techniques of contribution uncertainty, preference un-

certainty and effect uncertainty.

4.2. Handling Contribution Uncertainty

Let us first consider the contribution link sourced from an OR-decomposed sub-goal. For an OR-decomposed goal, assume that there are n sub-goals contributing to m soft goals. Recall the probabilistic interpretation of the contribution links introduced in Section 2.1, the weight w of a contribution link sourced from hg_i $(1 \leq i \leq n)$ to sg_k $(1 \leq k \leq m)$ can be interpreted as a conditional probability depending on the type of the contribution link. For instance, in $hg_i \xrightarrow{w+S} sg_k$, w can be interpreted as the conditional probability $P(sg_k\ is\ satisfied \mid hg_i\ is\ satisfied)$.

$$
\begin{aligned}
hg_i \xrightarrow{w+S} sg_k : w &= P(sg_k\ is\ satisfied \mid hg_i\ is\ satisfied) \\
&= \frac{P(hg_i\ is\ satisfied\ \wedge\ sg_k\ is\ satisfied)}{P(hg_i\ is\ satisfied)} \\
hg_i \xrightarrow{w-S} sg_k : w &= P(sg_k\ is\ denied \mid hg_i\ is\ satisfied) \\
&= \frac{P(hg_i\ is\ satisfied\ \wedge\ sg_k\ is\ denied)}{P(hg_i\ is\ satisfied)} \\
hg_i \xrightarrow{w+D} sg_k : w &= P(sg_k\ is\ denied \mid hg_i\ is\ denied) \\
&= \frac{P(hg_i\ is\ denied\ \wedge\ sg_k\ is\ denied)}{P(hg_i\ is\ denied)} \\
hg_i \xrightarrow{w-D} sg_k : w &= P(sg_k\ is\ satisfied \mid hg_i\ is\ denied) \\
&= \frac{P(hg_i\ is\ denied\ \wedge\ sg_k\ is\ satisfied)}{P(hg_i\ is\ denied)}
\end{aligned}
$$

For each sub-goal hg_i, $hg_i\ is\ satisfied$ means hg_i is chosen for the satisfaction of its parent OR-decomposed goal and is included in the current goal solution. On the other hand, $hg_i\ is\ denied$ means hg_i is not included in the current goal solution but one of its sibling goals $hg_j (j \neq i)$ is chosen for the satisfaction of their parent OR-decomposed goal. Based on this observation, the weight w of $w + D$ and $w - D$ can be further transformed into the followings. For instance, in $hg_i \xrightarrow{w+D} sg_k$, w can be interpreted as the probability of the denial of sg_k under the condition that hg_i is denied, which can be seen as the sum of the probabilities of the denial of sg_k under the condition that

one of the hg_i's sibling goal $hg_j (j \neq i)$ is satisfied.

$$hg_i \xrightarrow{w+D} sg_k : w = \sum_{j=1, j\neq i}^{n} P(sg_k \; is \; denied \mid hg_j \; is \; satisfied)$$
$$= \sum_{j=1, j\neq i}^{n} \frac{P(hg_j \; is \; satisfied \; \wedge \; sg_k \; is \; denied)}{P(hg_j \; is \; satisfied)}$$
$$hg_i \xrightarrow{w-D} sg_k : w = \sum_{j=1, j\neq i}^{n} P(sg_k \; is \; satisfied \mid hg_j \; is \; satisfied)$$
$$= \sum_{j=1, j\neq i}^{n} \frac{P(hg_j \; is \; satisfied \; \wedge \; sg_k \; is \; satisfied)}{P(hg_j \; is \; satisfied)}$$

In order to calculate these conditional probabilities, each pair (hg_i, sg_k) is associated with a couple $[Sat_{ik}, Den_{ik}]$, representing the sum of satisfaction and denial level of sg_k in all the past time intervals under the condition that hg_i is satisfied. $[Sat_{ik}, Den_{ik}]$ denotes the relative frequency of the occurrence of satisfaction and denial of sg_k when hg_i is satisfied.

Considering the uncertainty with soft goal satisfaction introduced in Section 2.3, similar handling techniques proposed by Baresi et al. (2010) and Whittle et al. (2009) are adopted, i.e., the membership functions in Figure 2 are used to fuzzify the satisfaction and denial level of soft goals (Example 4.1). Hence, our approach not only handles soft goal satisfaction uncertainty based on fuzzy set theory but also handles contribution uncertainty based on probability theory. Notice that the actual value can be monitored, and the expected value, in our work, is the average of the last 10 actual values. This is because the expected value, if set fixed at design-time, contains uncertainty itself.

Example 4.1. *Table 1 lists the fuzzified satisfaction and denial level of soft goal "Response Time Be Reduced" (k = 1) in each of the past 10 time intervals. Product details are viewed in multimedia mode (i = 1) in the first 5 time intervals, and then in textual mode (i = 2) in the last 5 time intervals. Thus,* Sat_{11}*, the satisfaction frequency of "Response Time Be Reduced" when multimedia mode is chosen, is 0.3 + 0.2 + 0.15 + 0.2 + 0.1 = 0.95, and* Den_{11}*, the denial frequency of "Response Time Be Reduced" when multimedia mode is chosen, is 0.7 + 0.8 + 0.85 + 0.8 + 0.9 = 4.05. Similarly,* Sat_{21} *is 0.7 + 0.8 + 0.75 + 0.8 + 0.7 = 3.75, and* Den_{21} *is 0.3 + 0.2 + 0.25 + 0.2 + 0.3 = 1.25.*

Table 1: Fuzzified Satisfaction and Denial Level of *Response Time Be Reduced*.

Time	**Details Be Viewed**	**Response Time Be Reduced**	
		Sat	**Den**
1	*View in Multimedia Mode*	0.3	0.7
2	*View in Multimedia Mode*	0.2	0.8
3	*View in Multimedia Mode*	0.15	0.85
4	*View in Multimedia Mode*	0.2	0.8
5	*View in Multimedia Mode*	0.1	0.9
6	*View in Textual Mode*	0.7	0.3
7	*View in Textual Mode*	0.8	0.2
8	*View in Textual Mode*	0.75	0.25
9	*View in Textual Mode*	0.8	0.2
10	*View in Textual Mode*	0.7	0.3

Based on the couples $[Sat_{ik}, Den_{ik}]$, the conditional probabilities can be calculated as follows, and we call them the posterior knowledge of the contribution links.

$$hg_i \xrightarrow{w+S} sg_k : w = \frac{Sat_{ik}}{Sat_{ik} + Den_{ik}}$$

$$hg_i \xrightarrow{w-S} sg_k : w = \frac{Den_{ik}}{Sat_{ik} + Den_{ik}}$$

$$hg_i \xrightarrow{w+D} sg_k : w = \frac{\sum_{j=1,j\neq i}^{n} Den_{jk}}{\sum_{j=1,j\neq i}^{n} Sat_{jk} + \sum_{j=1,j\neq i}^{n} Den_{jk}}$$

$$hg_i \xrightarrow{w-D} sg_k : w = \frac{\sum_{j=1,j\neq i}^{n} Sat_{jk}}{\sum_{j=1,j\neq i}^{n} Sat_{jk} + \sum_{j=1,j\neq i}^{n} Den_{jk}}$$

The weight of contribution link $w+$ $(w-)$ is calculated as the average of the weights of contribution links $w+S$ and $w+D$ ($w-S$ and $w-D$). Rather than dropping the initial contribution links elicited by domain experts during requirements analysis (the prior knowledge), we update the contribution links as the weighted sum of the prior and posterior knowledge. A higher weight to the prior knowledge represents higher confidence on the priori.

Example 4.2. *Following Example 4.1, the weight of* $hg_1 \xrightarrow{w+S} sg_1$ *is* $\frac{0.95}{0.95+4.05}$ *= 0.19, the weight of* $hg_1 \xrightarrow{w-S} sg_1$ *is 0.81, the weight of* $hg_1 \xrightarrow{w+D} sg_1$ *is*

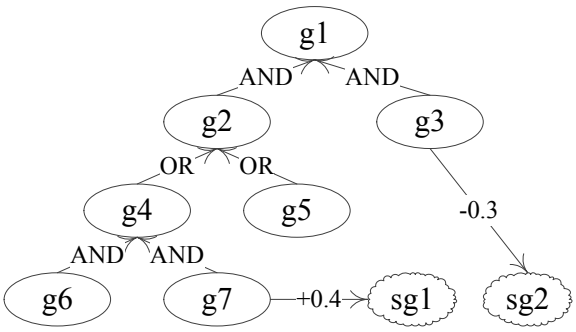

Figure 5: An Example of Contribution Links Sourced from AND-Decomposed Sub-Goals.

$\frac{1.25}{3.75+1.25} = 0.25$, *and the weight of* $hg_1 \xrightarrow{w-D} sg_1$ *is 0.75. Hence, the weight of* $hg_1 \xrightarrow{w-} sg_1$ *is* $\frac{0.81+0.75}{2} = 0.78$. *Considering that the initial contribution link sourced from* hg_1 *to* sg_1 *in Figure 1 is 0.9– and assuming that the confidence on the prior is 0.4, the weight is updated to* $0.4 \times 0.9 + (1 - 0.4) \times 0.78 = 0.828$.

Now let us consider the contribution link sourced from an AND-decomposed sub-goal. Notice that if such an AND-decomposed sub-goal is included (or not included) in a goal solution, its closest OR-decomposed ancestor sub-goal if exists is also included (or not included) in this goal solution. Based on this observation, the contribution link sourced from an AND-decomposed sub-goal can be handled using the proposed technique by moving the source of such a contribution link to the closest OR-decomposed ancestor sub-goal if exists (Example 4.3). If such an ancestor goal does not exist, the uncertainty in this contribution link can be neglected since its source goal is always included in all goal solutions and thus will not produce negative effect.

Example 4.3. *Figure 5 shows an example of the contribution links sourced from AND-decomposed sub-goals. The uncertainty in the contribution link sourced from "g7" to "sg1" can be handled using the proposed technique by moving its source to "g4". However, the uncertainty in the contribution link sourced from "g3" to "sg2" can be neglected because "g3" is always included in all the goal solutions.*

Notice that the handling of contribution uncertainty may be less effective when a soft goal is simultaneously contributed by the sub-goals of multiple OR-decomposed goals (or multiple AND-decomposed (sub-)goals). This is

because the proposed technique cannot differentiate which OR-decomposed goal's sub-goal (or AND-decomposed (sub-)goal) contributes more or less to the satisfaction and denial of the same soft goal. However, this can be solved by refining the soft goals into sub-soft goals by type or topic (Mylopoulos et al., 1992) until every soft goal is contributed by the sub-goals of one OR-decomposed goal (or one AND-decomposed (sub-)goal).

4.3. Handling Preference Uncertainty

We integrate the dynamic quality trade-off mechanism proposed in our earlier work (Peng et al., 2012) to handle preference uncertainty. It takes as input the earned value and quality values, and dynamically tunes the preferences of soft goals in response to the changing environments using a PID (proportional-integral-derivative) feedback controller.

The reason for adopting a control-theoretic technique to realize *preference tuner* is that every soft goal has an approximately proportional relationship between its preference and expectation, i.e., the better a soft goal is expected, the higher its preference should be (Peng et al., 2012). Hence, the tuner is designed to prevent a soft goal from getting worse by increasing its preference such that another goal solution with better satisfaction level of this soft goal will be chosen. For instance, if the response time is getting too long in online shopping, the preference of the soft goal *Response Time Be Reduced* should be increased such that textual mode will be chosen to make response time close to its expectation. Detailed descriptions of the preference tuning algorithm can be found in Peng et al. (2012).

4.4. Handling Effect Uncertainty

It is almost impossible to know if an adaptation action has taken effect or not before its effect has been observed from the running system. Different actions need different timing delays; even the same action may need different timing delays under different environments. To automatically determine the proper timing delay for an adaptation action, the challenge is to formalize the *observable effect*. From the perspective of VBSE, we formalize the observable effect as the satisfaction level of earned value and determine the proper timing delay by a heuristics-based technique.

Since earned value also does not have a clear-cut satisfaction criterion, its satisfaction level is fuzzified by the membership function in Figure 2 (a). The actual earned value is monitored and analyzed from the runtime data as introduced in Section 2.2, and the expected earned value is the average

of the last 10 actual earned values. Notice that earned value is used as the global indicator of self-optimization, and thus the adaptation mechanism is triggered when the satisfaction level of earned value is less than the value satisfaction threshold α.

Instead of using complex learning techniques that usually require much runtime data to be effective, we choose to adopt several simple and straightforward heuristic rules (i.e., rules **R1** to **R4**) to guide the handling of effect uncertainty because there is little available runtime data that can be used before an adaptation action is shown to be effective or ineffective. The proper timing delay of an adaptation action is dynamically determined by Algorithm 1.

$$\begin{aligned}
&\textbf{R1} : delay \geq delay_{min} \\
&\textbf{R2} : delay \leq delay_{max} \\
&\textbf{R3} : actual\ earned\ value\ (t+1) \geq actual\ earned\ value(t) \\
&\textbf{R4} : actual\ earned\ value\ (t+1) \succeq expected\ earned\ value, \\
&\qquad and\ its\ membership\ value \geq \alpha
\end{aligned}$$

Algorithm 1 The Procedure of Delay Analysis

```
1: procedure DELAY-ANALYSIS(earned value, delay)
2:     delay++
3:     if ! R1 then
4:         continue delaying
5:     else if R2 then
6:         if (R3 && R4) || delay == delay_max then
7:             stop delaying
8:         else
9:             continue delaying
10:        end if
11:    end if
12: end procedure
```

Rule **R1** indicates that the prior knowledge of the timing delay of an adaptation action can be used to determine the minimum timing delay for facilitating the analysis (Line 3-4). If no (trusted) prior knowledge is available, the minimum timing delay should be one time interval. Rule **R2** means

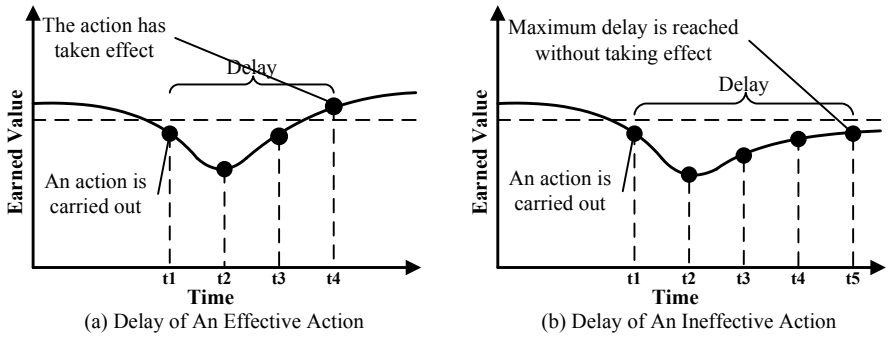

Figure 6: Delay Patterns Identified from our Experimental Study.

that delay should be stopped if it has taken a long time but the action has not brought into effect (Line 6-7), i.e., to prevent the actually ineffective actions from running for a long time and thus making the system running in a suboptimal fashion. Rules **R3** and **R4** indicate that there is an increasing trend of earned value and earned value has been satisfied to an acceptable level, meaning that the action has taken effect from the perspective of earned value and thus the delay should be stopped (Line 6-7).

Example 4.4 shows two delay instances that are actually two typical patterns of delay. All delay instances found in our experimental study are variants of these two patterns.

Example 4.4. *Figure 6 shows two delay instances of an effective action and an ineffective action. The minimum timing delay is one time interval and the maximum timing delay is four time intervals. In Figure 6 (a), at time t1, an action is carried out and delay is started because of the deviation of earned value; at time t2, delay is continued because both* ***R3*** *and* ***R4*** *are violated; at time t3,* ***R3*** *is achieved but* ***R4*** *is still violated, hence, delay is continued; at time t4, delay is stopped because all heuristic rules are achieved (i.e., the action is considered to be effective). Similarly, in Figure 6 (b), an action is carried out at time t1; at time t5,* ***R4*** *is still violated and the maximum delay is reached (i.e., the action is considered to be ineffective), hence, delay is stopped and the adaptation mechanism is re-triggered to find another goal solution.*

5. Implementation

We implement the framework in Figure 4 in an extensible way in order to provide a generic infrastructure of the goal-driven self-optimization with proposed uncertainty handling techniques. Specifically, *Delay Analyzer*, *Contribution Updater*, *Preference Tuner*, *Goal Reasoner* and *Goal Configurator* are application-independent components. They are realized in a plug-in manner so that necessary extensions or different implementations that may be later introduced can be easily integrated with the infrastructure. On the other hand, *Sensors* and *Architecture Configurator* are application-specific components. They are programmed as external RMI interfaces using RMI-IIOP technology so that the infrastructure and the running system can be decoupled, distributed and stand-alone. Notice that this implementation of the framework is a reconstruction of the implementation proposed in Peng et al. (2012) by following the MAPE-K architecture, integrating uncertainty handling and adopting quantitative goal reasoning.

In order to apply our approach, developers should provide their own implementation to the two external interfaces. The implementation to *Sensors*, i.e., to obtain application-specific earned value and quality values, depends on the business goals, strategies and policies and on the quality attributes' own characteristics. Typical techniques for realizing sensors include logging, profiling and aspect-oriented programming (Salehie and Tahvildari, 2009). For the online shopping system, we compute earned value as introduced in Section 2.2, and measure quality values by analyzing business database and log files (e.g., the success or failure of a payment transaction, the response time of a browsing request, etc.).

The implementation to *Architecture Configurator*, i.e., to map from goal reconfiguration to architectural reconfiguration, depends on the implementation and architecture styles of the target system. Typical techniques for achieving architectural reconfiguration include lightweight methods like design patterns (e.g., wrapper, proxy, or strategy) for well-designed systems, reflective component models (e.g., Fractal (Bruneton et al., 2006)) for component-based systems, service-oriented techniques (e.g., AO4BPEL (Charfi and Mezini, 2007)), and architecture-based middlewares (e.g., SM@RT (Song et al., 2011)). For the online shopping system, we realize architectural reconfiguration by Java reflection mechanism, by which components are instantiates by runtime reflection on the basis of goal-component mappings.

6. Experimental Study

To evaluate the proposed approach, we conducted an experimental study[1] on an online shopping system, whose requirements goal model is illustrated in Figure 1, to answer the following research questions:

- **Q1**: Do the three kinds of uncertainty really exist and have influences on the goal-driven self-optimization?
- **Q2**: What benefit can be achieved from the combined handling of the three kinds of uncertainty in goal-driven self-optimization?
- **Q3**: What is the scalability of our approach?

6.1. Experimental Setup

The stress testing tool JMeter 2.3.4 was used to simulate concurrent system accesses with a constant load about 25 users, and Badboy 2.1 was used to record the JMeter test plan. The experiments were conducted on a server with an Intel Core2 Quad 2.33 GHz processor and 4GB RAM, running Windows Server 2008.

In our experiments, usability was stochastically distributed with different density under textual and multimedia mode to simulate real-life customer feedbacks, order canceling rate was stochastically distributed with different density under different strategies for dealing with the out of stock products to simulate real-life customer behaviors, and response time and ordering cost were intrinsically changing under the previous two simulations.

Each experiment was executed in a continuous running of about 30 minutes. Earned value was calculated as introduced in Section 2.2, and quality values were computed by log analysis based on the collected runtime data per time interval. Notice that the weight of the prior knowledge used in contribution updating was set to 0.5, the minimum and maximum timing delay were set to one and four time intervals, the value satisfaction threshold was set to 0.5, and the time interval was set to 30 seconds.

We conducted the experiments with the following eight approaches using the same experimental setting. To extensively evaluate the proposed approach, all the possible combinations of the handling of the three kinds of uncertainty were considered.

[1] All the models and results used in our experimental study are available at `www.se.fudan.edu.cn/research/self-adaptation/JSS-UncertaintyHandling`.

- **A1**: the approach without self-optimization capability.
- **A2**: the self-optimization approach using predefined adaptation rules. One of the four adaptation rules is to switch from multimedia mode to textual mode if average response time is larger than a threshold.
- **A3–P**: the goal-driven self-optimization approach that only handles preference uncertainty.
- **A4–PE**: the goal-driven self-optimization approach that handles preference uncertainty and effect uncertainty.
- **A5–C**: the goal-driven self-optimization approach that only handles contribution uncertainty.
- **A6–CE**: the goal-driven self-optimization approach that handles contribution uncertainty and effect uncertainty.
- **A7–PC**: the goal-driven self-optimization approach that handles preference uncertainty and contribution uncertainty.
- **A8–PCE**: the goal-driven self-optimization approach that handles preference uncertainty, contribution uncertainty and effect uncertainty.

6.2. Evaluation of Handling Uncertainty (Q1)

Although the experiments of handling preference uncertainty (i.e., **A3–P**, **A4–PE**, **A7–PC** and **A8–PCE**) were conducted, the detailed evaluation of handling preference uncertainty was not considered separately in this paper since it has been demonstrated in our earlier paper (Peng et al., 2012). Here we mainly focus on the evaluation of handling contribution uncertainty and effect uncertainty.

6.2.1. Evaluation of Handling Contribution Uncertainty

Table 2 shows the updating results of contribution links when the experiments finished (i.e., after a running of 30 minutes). The first column lists the initial contribution links as shown in Figure 1. The second to fifth columns respectively report the final updated contribution links in the experiments with handling of contribution uncertainty, i.e., **A5–C**, **A6–CE**, **A7–PC** and **A8–PCE**.

Table 2: Updating Results of Contribution Links

Initial	A5–C	A6–CE	A7–PC	A8–PCE
t2 $\xrightarrow{0.7+}$ sg1	0.839	0.839	0.680	0.681
t2 $\xrightarrow{0.8-}$ sg2	0.776	0.775	0.832	0.837
t3 $\xrightarrow{0.9-}$ sg1	**0.729**	**0.707**	**0.780**	**0.781**
t3 $\xrightarrow{0.9+}$ sg2	**0.921**	**0.921**	0.882	0.887
t6 $\xrightarrow{0.7-}$ sg3	0.843	0.843	0.702	0.703
t7 $\xrightarrow{0.6-}$ sg3	0.554	0.555	**0.448**	**0.447**
t8 $\xrightarrow{0.8+}$ sg3	**0.809**	**0.825**	0.828	0.827
t8 $\xrightarrow{0.7-}$ sg4	**0.652**	**0.663**	0.680	0.685
g1* $\xrightarrow{0.8+}$ sg4	0.646	0.646	0.730	0.735

* *g1* represents goal *Out of Stock Items Be Ordered*.

It can be observed that **A5–C** and **A6–CE** get different results from **A7–PC** and **A8–PCE**. We found that only one adaptation action was triggered in the experiments with **A5–C** and **A6–CE** and thus the online shopping system was configured to the goal solution [$t1$, $t3$, $t4$, $t5$, $t8$, $t9$, $t10$, $t11$] most of the time. As a result, only the contribution links sourced from $t3$ and $t8$ (indicated in bold in the second and third columns) were well updated. On the other hand, at least eight adaptation actions were triggered in the experiments with **A7–PC** and **A8–PCE** and thus the online shopping system was configured to different goal solutions. As a result, all the contribution links were well updated and those sourced from $t3$ and $t8$ were similar to the results of **A5–C** and **A6–CE**. The above analysis also suggests that it is reasonable to provide a combined handing of preference uncertainty and contribution uncertainty, which will be detailed demonstrated in Section 6.3.

It can be seen that **A7–PC** and **A8–PCE** get almost the same results, and two contribution links are greatly overestimated (indicated in bold in the fourth and fifth columns) and others are slightly overestimated or underestimated. For instance, the contribution link sourced from *View in Multimedia Mode* (*t3*) to *Response Time Be Reduced* (*sg1*) is greatly overestimated by about 0.12, and the contribution link sourced from *Remove Out of Stock Items* (*t8*) to *Ordering Cost Be Reduced* (*sg3*) is slightly underestimated by about 0.03. This answers **Q1** positively that contribution uncertainty really exists.

Further, due to the greatly overestimated contribution link sourced from *View in Multimedia Mode* (*t3*) to *Response Time Be Reduced* (*sg1*), we found

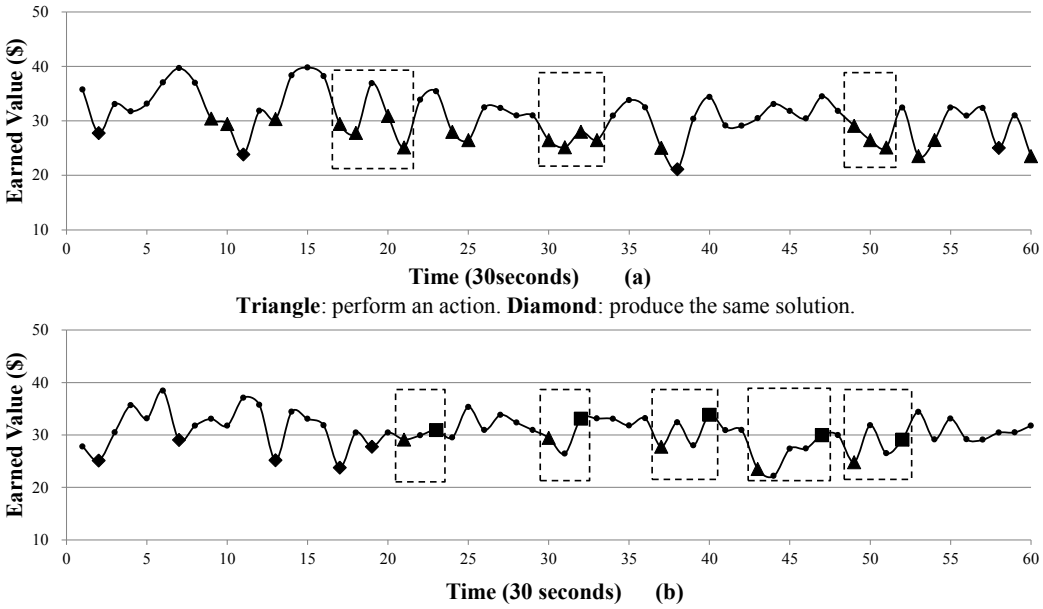

Figure 7: The Self-Optimization Process (Earned Value vs. Time).

in the experiments without handling contribution uncertainty (i.e., **A3**–**P** and **A4**–**PE**) that there happened a switch from multimedia mode to textual mode, which would not happen if contribution uncertainty were properly treated. This answers **Q1** positively that contribution uncertainty, if not properly handled, may produce suboptimal goal solutions, and it is practical to take such uncertainty into consideration to make self-optimization more satisfactory.

6.2.2. Evaluation of Handling Effect Uncertainty

Figure 7 (a) and (b) respectively show the self-optimization process without handling effect uncertainty (i.e., **A7**–**PC**) and with handling effect uncertainty (i.e., **A8**–**PCE**). The X axis denotes discrete time intervals of 30 seconds and the Y axis denotes the earned value in each time interval. The triangle point on the curve represents that an adaptation action is performed, the diamond point represents that the goal reasoner produces the same goal solution and thus no adaptation action is performed, and the square point represents that the timing delay of the last action is stopped.

It can be observed that, without handling effect uncertainty, the system

suffered 3 oscillations of adaptation actions (identified by dashed rectangles in Figure 7 (a)), performed totally 20 times of adaptation actions in 30 minutes, and had a standard deviation of earned value of 4.24. Notice that Figure 3 is taken from the second oscillation. On the other hand, by handling effect uncertainty, the system performed only 5 times of adaptation actions in 30 minutes and had a standard deviation of earned value of 3.33. This answers **Q1** positively that effect uncertainty really exists.

Further, the delay length of all the 5 adaptation actions were identified by dashed rectangles in Figure 7 (b). Among them, two actions had a delay of 2 time intervals, each avoiding 1 premature action; two actions had a delay of 3 time intervals, each avoiding 2 premature actions; and one action had a delay of 4 time intervals, avoiding 3 premature actions. For instance, an action happened due to a deviation of earned value at time 30, and a premature action would have happened at time 31 due to an even worse deviation of earned value if there were no consideration of timing delay. Hence, at least 9 premature actions were avoided through handling effect uncertainty, and all the actions took effect before reaching the maximum delay. This answers **Q1** positively that effect uncertainty, if not properly handled, may produce premature actions and cause oscillation, and it is reasonable to take timing delay into consideration to make the system more stable.

It is worth to mention that we also get similar results of effect uncertainty with **A3–P** and **A4–PE**, and with **A5–C** and **A6–CE**. It is also worth to mention that the fourth action's delay in Figure 7 (b) is a variant of the second pattern in Figure 6, with the difference being that it took effect when reaching the maximum delay; the first and second action's delay almost follows the first pattern in Figure 6; and the third and fifth action's delay are a variant of the first pattern in Figure 6, with the difference being that they first had an increase of earned value before a decrease. A wider experiment should be performed to find more delay patterns to guide the derivation of heuristic rules for determining the proper timing delay.

6.3. Effectiveness Evaluation (Q2)

Table 3 shows the means of earned value and quality values per interval and the adaptation frequency in 30 minutes under different approaches. The adaptation frequency is used as an indicator for the stability of self-adaptive systems. The first and second columns list the involved approaches and their earned values. The third to sixth columns respectively list the quality values, i.e., response time, usability, ordering cost and order canceling rate.

Table 3: Means of Earned Value, Quality Values and Adaptation Frequency

App.	Val. ($)	Res. (ms)	Usa. (%)	Cos. ($)	Can. (%)	Fre. (#)
A1	28.76	948.97	82.38	12.91	14.38	N/A
A2	29.82	924.18	83.70	11.69	15.13	29 + 0
A3–P	30.57	934.18	86.31	12.40	15.55	19 + 4
A4–PE	31.27	**887.14**	86.22	13.31	14.82	10 + 3
A5–C	29.20	1044.08	87.07	**10.09**	18.62	1 + 20
A6–CE	29.34	981.68	86.10	10.64	19.57	1 + 18
A7–PC	30.73	928.96	**87.31**	12.32	15.43	16 + 6
A8–PCE	**31.58**	889.14	86.49	13.12	**14.06**	**8 + 3**

The last column lists the adaptation frequency in the form of $a + b$, where a represents the number of *succeeded adaptation* (i.e., an adaptation action is performed) and b represents the number of *failed adaptation* (i.e., the goal reasoner cannot find a better goal solution and thus no adaptation action is performed although an adaptation action is really needed). Notice that the first row lists the average results of **A1** under all the 6 goal solutions of the online shopping system.

Firstly, it can be seen that **A3–P** has a gain of 6.29% and 2.52% in earned value over **A1** and **A2** respectively, **A5–C** has a gain of 1.53% but a loss of 2.08% in earned value over **A1** and **A2** respectively, and **A7–PC** has a gain of 6.85% and 3.05% in earned value over **A1** and **A2** respectively. This shows that handling preference or contribution uncertainty independently cannot achieve the highest earned value, but a combination of them can.

Secondly, it can be observed that **A3–P** decreases succeeded adaptation frequency over **A2** by 34.48% and has a low frequency of failed adaptation, **A5–C** has one succeeded adaptation but has a high frequency of failed adaptation, and **A7–PC** decreases succeeded adaptation frequency over **A2** by 44.83% and has a low frequency of failed adaptation. This demonstrates that handling preference or contribution uncertainty independently may suffer instability (i.e., high frequency of succeeded adaptation) or make systems run in sub-optimal manner (i.e., high frequency of failed adaptation), but a combination of them can effectively relieve such problems.

Thirdly, after further considering effect uncertainty, **A4–PE** has a gain of 2.29% in earned value and a decrease of 47.37% in succeeded adaptation frequency over **A3–P**, **A6-CE** has a gain of 0.48% in earned value and a decrease of 10.00% in failed adaptation frequency over **A4–C**, **A8–PCE** has a gain of 2.77% in earned value and a decrease of 50.00% in adaptation

frequency over **A7–PC**, and **A8–PCE** has the highest earned value and the lowest adaptation frequency. This demonstrates that the handling of effect uncertainty should be combined with the handling of preference uncertainty and contribution uncertainty in order to further improve the effectiveness of goal-driven self-optimization.

Last but not least, in terms of quality attributes, **A3–P**, **A4–PE**, **A5–C**, **A6–CE**, **A7–PC** and **A8–PCE** are better in certain dimensions but worse in others because we focus on the overall optimization from the perspective of earned value rather than on the optimization of quality attributes.

The above analysis answers **Q2** positively that it is practical and reasonable to combine the handling of the three kinds of uncertainty, and by such a combination, goal-driven self-optimization can be improved in terms of both earned value and stability.

6.4. Scalability Evaluation (Q3)

The scalability of our approach is mainly determined by the performance of the *Analyze* and *Plan* step. In the *Analyze* step, the time complexity of *Delay Analyzer*, *Preference Tuner* and *Contribution Updater* are respectively constant time, linear time with the number of soft goals and linear time with the number of contribution links. In the *Plan* step, the time complexity of *Goal Reasoner* is exponential time with the number of OR-decomposed goals.

To evaluate the scalability of our approach, we conducted a set of experiments with automatically generated goal models whose sizes vary from 30 goals to 210 goals. Table 4 reports the experimental results. The first five columns respectively list the number of goals, OR-decomposed goals, soft goals, contribution links and goal solutions in each goal model. The last four columns respectively list the performance (in milliseconds) of *Delay Analyzer*, *Preference Tuner*, *Contribution Updater* and *Goal Reasoner*.

The *Analyze* step took around 3 milliseconds in all the experiments. And the *Plan* step took around 3 seconds with the experiments on the goal model containing 18 OR-decomposed goals (with 46656 goal solutions), which is still feasible in our approach. As the number of OR-decomposed goals climbs to 21 (with 279936 goal solutions), the *Plan* step returned an "out of memory" error, which is infeasible in our approach at this scale. Hence, this answers **Q3** that our approach can be applied to real-life systems with medium-sized requirements goal models.

Table 4: Performance of Our Approach.

Goal Model (#)					**Performance (ms)**			
G	OR	SG	C	GS	DA	PT	CL	GR
30	3	4	9	6	1	1	1	1
60	6	8	18	36	1	1	1	8
90	9	12	27	216	1	1	1	20
120	12	16	36	1296	1	1	1	78
150	15	20	45	7776	1	1	1	452
180	18	24	54	46656	1	1	1	3079
210	21	28	63	279936	1	1	1	out

6.5. Discussion

The handling of contribution uncertainty may be less effective when the weights of contribution links change impulsively. In this case, our approach responds slowly to such changes because our probability-based technique needs some time of observing to precisely capture such changes. One possible remedy to this is to use the latest n monitored quality values for calculating $[Sat_{ik}, Den_{ik}]$ instead of using all the historical monitored quality values.

The handling of effect uncertainty relies on a few constants, i.e., value satisfaction threshold, and minimum and maximum timing delay. The setting of value satisfaction threshold α should consider a system's tolerance capability of earned value violation. For instance, if a system has a low tolerance capability of value violation, α should be larger (e.g., 0.9); otherwise, α should be smaller (e.g., 0.4). The minimum timing delay is actually optional with the default value being one time interval because every adaptation action needs at least one time interval to take effect, and can be set to other values if prior knowledge about the minimum timing delay is available (e.g., the historical runtime data of adaptation actions). And the maximum timing delay can be estimated by the historical runtime data of adaptation actions if available, or can be analyzed by business experts considering the maximum tolerance range of value violation from the business perspective.

Last but not least, we argue that the three kinds of uncertainty really exist and can be controlled to some extent by the proposed techniques such that the effectiveness of goal-driven self-optimization can be improved. Surely, other possible techniques should be explored to handle the uncertainty (especially the effect uncertainty) and be compared with the proposed techniques.

7. Related Work

Our work falls in the area of self-adaptive systems that are able to change their structures and/or behaviors in response to the changes in their operational environments, the system themselves and their requirements (Lemos et al., 2011). Instead of enumerating all of the related studies, we refer the readers to Cheng et al. (2009a), Sawyer et al. (2010) and Lemos et al. (2011) for a comprehensive analysis of the current state-of-the-art, limitations and challenges. Here we focus our discussion on the most closely related studies in four areas: goal-driven self-adaptation, self-adaptation under probability, runtime uncertainty handling, and design-time uncertainty handling.

7.1. Goal-Driven Self-Adaptation

Self-adaptive systems should be requirements-aware since these systems are increasingly operating in volatile and poorly-understood environments which forces requirements engineers to make assumptions about the states and events these systems may encounter at runtime (Sawyer et al., 2010). Several advances have been made on goal-driven self-adaptation. Dalpiaz et al. (2009) proposed a conceptual architecture to provide systems with self-configuration capabilities. Wang and Mylopoulos (2009) proposed a requirements monitoring and diagnosing approach to allow self-repairing in cases of failures. Chen et al. (2011) proposed a requirements-driven self-optimization approach to achieve survivability assurance for Web systems. Souza et al. (2011) proposed a systematic system identification method for self-adaptive systems to identify configuration parameters and target outputs as well as their qualitative relations, which can be used to dynamically tune these parameters in cases of requirements deviations (Souza et al., 2012). Fu et al. (2012) proposed a stateful requirements monitoring approach for decentralized self-repairing of socio-technical systems. Salehie et al. (2012) proposed a requirements-driven approach to support adaptive security in order to protect variable assets at runtime.

These studies have the commonality that they use requirements goal models as the knowledge base for self-adaptation and cover the four self-* capabilities proposed by IBM's autonomic computing (Kephart and Chess, 2003). However, all of them lack an explicit consideration of uncertainty handling, which may cause negative effect on self-adaptation.

Table 5: Literature Comparison over Supported Uncertainty Handling

Approach	Ext.				Int.
	Sat.	Pre.	Con.	Eff.	
RELAX (Whittle et al., 2009; Cheng et al., 2009b)	√	×	×	×	×
FLAGS (Baresi et al., 2010)	√	×	×	×	×
REAssuRE (Welsh et al., 2011; Ramirez et al., 2012)	√.	×	√	×	×
Bencomo and Belaggoun (2013); Bencomo et al. (2013)	√.	×	√	×	×
POISED (Esfahani et al., 2011)	×	×	×	×	√
Our Approach	√.	√	√	√	×

7.2. Self-Adaptation under Probability

Ghezzi et al. (2013) presented ADAM to support adaptation aimed at mitigating non-functional uncertainty. It exploits Markov Decision Processes (MDPs) to model self-adaptive systems which have alternative and optional functionality implementations. According to the aggregated quality metrics up to the current state, ADAM can find the execution path with the highest probability to satisfy the non-functional requirements and then enable adaptation by switching to alternative implementations or skipping the optional. They do not consider the inaccuracy of the transition probabilities in such MDP models, while we take into account the uncertainty in goal models by tuning the preference ranks of soft goals and updating the weights of contribution links.

Filieri et al. (2012) presented KAMI to provide continuous verification of reliability and performance requirements of self-adaptive systems by exploiting Discrete-Time Markov Chains (DTMCs) and Continuous-Time Markov Chains (CTMCs) respectively and using a model checking technique. KAMI can update the model parameters of DTMCs and CTMCs through Bayesian estimation based on runtime observations. Similarly, Sykes et al. (2013) proposed to enable self-adaptive systems to cope with incomplete and inaccuracy knowledge by updating their behavior models. They use a probabilistic rule learning technique to not only update transition probabilities but also discover new structures of the model. These studies aim at updating the models used for the knowledge base of planning. Different from their models, we handle the uncertainty in requirements goal models and also deal with effect uncertainty.

7.3. Runtime Uncertainty Handling

Esfahani et al. (2011) distinguish between external uncertainty and inter-

nal uncertainty in self-adaptive systems. External uncertainty arises from the environment or domain, which is still a key challenge when requirements engineering for self-adaptive systems (Sawyer et al., 2010; Cheng et al., 2009a). On the other hand, internal uncertainty is underlying adaptation decision's impact on the quality objectives.

Table 5 provides a comparison between our approach and some approaches supporting runtime uncertainty handling reported in related work in terms of satisfaction uncertainty, preference uncertainty, contribution uncertainty, effect uncertainty and internal uncertainty. Several advances have been made on external uncertainty while less attention has been paid on internal uncertainty. Besides, most studies on external uncertainty focus on one kind of uncertainty while our approach handles almost all the listed external uncertainty (although this list is not sound and complete and we partially handle satisfaction uncertainty by dealing with soft goals).

Whittle et al. (2009) developed RELAX, a formal requirements specification language, whose semantics is defined in fuzzy branching temporal logic, to support explicit expression of environment uncertainty in requirements for self-adaptive systems. This enables the self-adaptive systems to temporarily relax the satisfaction of non-critical requirements when environment changes. Following Whittle et al. (2009), Cheng et al. (2009b) integrated RELAX with goal modeling and proposed a range of tactics for adaptation to deal with uncertainty.

Baresi et al. (2010) presented FLAGS, an extension of the KAOS goal model (van Lamsweerde and Letier, 2002), distinguishing between crisp goals, whose satisfaction is Boolean, and fuzzy goals, whose satisfaction is evaluated by fuzzy logic. Similar to RELAX, FLAGS also exploits fuzziness to handle the satisfaction uncertainty of fuzzy goals with the idea of tolerating small/transient deviations. This approach has been successfully applied to self-adaptive service compositions (Baresi and Pasquale, 2010a,b).

Both RELAX and FLAGS support the handling of goal satisfaction uncertainty, while our approach partially supports it by only dealing with soft goals and further handles contribution uncertainty, preference uncertainty and effect uncertainty.

Welsh et al. (2011) proposed REAssuRE, in which claims are attached to the contribution links to record the rationale for a choice of alternative goal operationalization when there is uncertainty about the optimal choice. Further, Ramirez et al. (2012) integrated REAssuRE with RELAX to introduce a fuzzy logic layer upon the evaluation criteria of claims' validity. When a

claim is falsified at runtime, its attached contribution link is updated (e.g., converting from *help* to *neutral*), and the goal model is re-evaluated to find a better solution. Hence, these studies presented a qualitative way to handle the contribution uncertainty. Enlightened by their work, we propose a quantitative way to handle the contribution uncertainty.

Following Welsh et al. (2011), Bencomo and Belaggoun (2013) proposed to map goal models and claims to dynamic decision networks (DDNs), where each qualitative contribution link corresponds to a conditional probability and each configuration is associated with a preference. When the validity of a claim is changed at runtime, the relevant conditional probabilities will be updated and the DDN model will be re-evaluated to find a configuration with the highest probability-weighted utility (Bencomo et al., 2013). The conditional probabilities and preferences are given by experts and only some of the conditional probabilities will be updated, while our approach can update all the conditional probabilities and tune the preferences.

Esfahani et al. (2011) proposed POISED to handle the internal uncertainty, which is built upon possibility theory to assess the positive and negative consequences of internal uncertainty. POISED can find a system configuration that maximizes the satisfaction of quality objectives as well as maximizes the positive consequences and minimizes the negative consequences. Complementary to this work, our approach can utilize this technique to handle the uncertainty underlying goal reasoning.

7.4. Design-Time Uncertainty Handling

Design-time uncertainty handling in requirements engineering has been proposed in researches on requirements elicitation, disambiguation and inconsistency check. Salay et al. (2012) used partial models to manage requirements uncertainty. This study is more qualitative (e.g., are there more alternatives to achieve a goal) where ours is more quantitative (e.g., is the contribution link underestimated). Yang et al. (2012) proposed an approach to detect the uncertainty in natural language requirements. This study handles requirements uncertainty at the language level where ours is at the model level. Arora et al. (2012) focused on the uncertainty arising from inconsistent feature interactions. The difference is that our approach has an aim to improve or maintain the satisfaction level of requirements even though initially the requirements goal models are not perfectly defined.

Karlsson and Ryan (1997) used the analytic hierarchy process technique, In et al. (2002) used a multi-criteria analysis method, and Avesani et al.

(2005) used a machine learning-based technique, to prioritize requirements. Liaskos et al. (2012) used the analytic hierarchy process technique to quantitatively elicit the weights of contribution links. Cheng (2008) statically designated a fixed timing delay for the adaptation mechanism to wait and observer the effect. As an initial attempt, we studied the impact of different windows of timing delay to self-adaptive systems in Chen et al. (2011).

Letier and van Lamsweerde (2004) proposed to specify partial degrees of goal satisfaction and quantify the impact of alternative designs on the degree of goal satisfaction for guiding requirements elaboration and design decision making. The partial degree of goal satisfaction is modeled by objective function on quality variables, and the objective function is specified by probabilistic models. This study used a probabilistic technique to tackle goal satisfaction uncertainty at design time, while we focus on runtime handling of contribution uncertainty, preference uncertainty and effect uncertainty.

Esfahani et al. (2013) presented GuideArch to quantitatively guide the exploration of architectural solution space, including ranking the architectures, finding the optimal, and identifying the critical decisions, under the uncertain impact of architectural alternatives on properties of interest. This study employs fuzzy math to represent and reason about uncertainty where ours uses a probability analysis to try to eliminate the contribution uncertainty.

In summary, these design-time approaches more or less involve user study or elicitation, making them infeasible to be applied in an unsupervised self-optimization process. However, we propose to dynamically handle the uncertainty at runtime.

8. Conclusions

Goal-driven self-optimization approaches usually use requirements goal models to reason about the optimal configurations for running systems. However, current approaches often suffer from diverse uncertainty as compounding factors of the predefined goal models and their suggested switches in the changing and uncertain environments. In this paper, we have proposed an approach to provide a combined handling of the three kinds of uncertainty, i.e., proposing a probabilistic-based analysis for contribution uncertainty, integrating a dynamic quality trade-off mechanism for preference uncertainty, and proposing a heuristics-based technique for effect uncertainty. After applying these runtime measures, our approach can limit the uncertainty's neg-

ative effect on goal-driven self-optimization, e.g., avoiding suboptimal or premature reconfigurations and mitigating oscillations of adaptation actions.

Our experimental study on an online shopping system has shown that the three kinds of uncertainty really exist and have negative effect on goal-driven self-optimization, and the effectiveness of goal-driven self-optimization approaches can be further improved in terms of both earned value and stability by a combined handling of the three kinds of uncertainty.

The preference and effect uncertainty are general in self-adaptive systems although our approach is under the umbrella of goal models. Thus we plan to apply the techniques to architecture-based self-adaptation approaches for further validating the feasibility and to exploit the specific uncertainty under architecture-based self-adaptation. In addition, we hope to extend our approach to further deal with the internal uncertainty underlying goal reasoning by integrating the related work like Esfahani et al. (2011). We also plan to explore other possible techniques to handle the three kinds of uncertainty as well as to perform a wider range of experiments on more real-life systems to find more delay patterns for a better handling of effect uncertainty.

Acknowledgment

This work is supported by National Natural Science Foundation of China under Grant No. 61361120097, and National High Technology Development 863 Program of China under Grant No. 2013AA01A605.

References

Arora, S., Sampath, P., Ramesh, S., 2012. Resolving uncertainty in automotive feature interactions. In: Proceedings of the 20th IEEE International Requirements Engineering Conference. pp. 21–30.

Avesani, P., Bazzanella, C., Perini, A., Susi, A., 2005. Facing scalability issues in requirements prioritization with machine learning techniques. In: Proceedings of the 13th IEEE International Conference on Requirements Engineering. pp. 297–306.

Baresi, L., Pasquale, L., 2010a. Adaptive goals for self-adaptive service compositions. In: Proceedings of the 2010 IEEE International Conference on Web Services. pp. 353–360.

Baresi, L., Pasquale, L., 2010b. Live goals for adaptive service compositions. In: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. pp. 114–123.

Baresi, L., Pasquale, L., Spoletini, P., 2010. Fuzzy goals for requirements-driven adaptation. In: Proceedings of the 18th IEEE International Requirements Engineering Conference. pp. 125–134.

Bencomo, N., Belaggoun, A., 2013. Supporting decision-making for self-adaptive systems: From goal models to dynamic decision networks. In: Proceedings of the 19th International Working Conference on Requirements Engineering: Foundation for Software Quality. pp. 221–236.

Bencomo, N., Belaggoun, A., Issarny, V., 2013. Dynamic decision networks for decision-making in self-adaptive systems: a case study. In: Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 113–122.

Blair, G. S., Bencomo, N., France, R. B., 2009. Models@run.time. Computer 42 (10), 22–27.

Boehm, B., 2003. Value-based software engineering: Reinventing "earned value" monitoring and control. SIGSOFT Softw. Eng. Notes 28 (2), 4–.

Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.-B., 2006. The FRACTAL component model and its support in Java: Experiences with auto-adaptive and reconfigurable systems. Softw. Pract. Exper. 36 (11-12), 1257–1284.

Charfi, A., Mezini, M., 2007. AO4BPEL: An aspect-oriented extension to BPEL. World Wide Web 10 (3), 309–344.

Chen, B., Peng, X., Yu, Y., Zhao, W., 2011. Are your sites down? Requirements-driven self-tuning for the survivability of Web systems. In: Proceedings of the 19th IEEE International Requirements Engineering Conference. pp. 219–228.

Cheng, B. H., Lemos, R., Giese, H., et al., 2009a. Software engineering for self-adaptive systems: A research roadmap. In: Software Engineering for Self-Adaptive Systems. Springer-Verlag, Berlin, Heidelberg, pp. 1–26.

Cheng, B. H., Sawyer, P., Bencomo, N., Whittle, J., 2009b. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In: Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems. pp. 468–483.

Cheng, S.-W., 2008. Rainbow: Cost-effective software architecture-based self-adaptation. Ph.D. thesis, Carnegie Mellon University.

Clements, P., Kazman, R., Klein, M., 2001. Evaluating Software Architectures: Methods and Case Studies, 1st Edition. Addison-Wesley, Boston, MA, USA.

Dalpiaz, F., Giorgini, P., Mylopoulos, J., 2009. An architecture for requirements-driven self-reconfiguration. In: Proceedings of the 21st International Conference on Advanced Information Systems Engineering. pp. 246–260.

Esfahani, N., Kouroshfar, E., Malek, S., 2011. Taming uncertainty in self-adaptive software. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. pp. 234–244.

Esfahani, N., Malek, S., Razavi, K., 2013. GuideArch: guiding the exploration of architectural solution space under uncertainty. In: Proceedings of the 2013 International Conference on Software Engineering. pp. 43–52.

Filieri, A., Ghezzi, C., Tamburrelli, G., 2012. A formal approach to adaptive software: continuous assurance of non-functional requirements. Form. Asp. Comput. 24 (2), 163–186.

Fu, L., Peng, X., Yu, Y., Zhao, W., 2012. Stateful requirements monitoring for self-repairing socio-technical systems. In: Proceedings of the 20th IEEE International Requirements Engineering Conference. pp. 121–130.

Ghezzi, C., Pinto, L. S., Spoletini, P., Tamburrelli, G., 2013. Managing non-functional uncertainty via model-driven adaptivity. In: Proceedings of the 2013 International Conference on Software Engineering. pp. 33–42.

Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R., 2002. Reasoning with goal models. In: Proceedings of the 21st International Conference on Conceptual Modeling. pp. 167–181.

In, H. P., Olson, D., Rodgers, T., 2002. Multi-criteria preference analysis for systematic requirements negotiation. In: Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment. pp. 887–892.

Karlsson, J., Ryan, K., 1997. A cost-value approach for prioritizing requirements. IEEE Software 14 (5), 67–74.

Kephart, J. O., Chess, D. M., 2003. The vision of autonomic computing. Computer 36 (1), 41–50.

Lapouchnian, A., Liaskos, S., Mylopoulos, J., Yu, Y., 2005. Towards requirements-driven autonomic systems design. In: Proceedings of the 2005 workshop on Design and Evolution of Autonomic Application Software. pp. 1–7.

Lemos, R., Giese, H., Müller, H., et al., 2011. Software engineering for self-adaptive systems: A second research roadmap. In: Software Engineering for Self-Adaptive Systems. Schloss Dagstuhl, Dagstuhl, Germany.

Letier, E., van Lamsweerde, A., 2004. Reasoning about partial goal satisfaction for requirements and design engineering. In: Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 53–62.

Liaskos, S., Jalman, R., Aranda, J., 2012. On eliciting contribution measures in goal models. In: Proceedings of the 20th IEEE International Requirements Engineering Conference. pp. 221–230.

Liaskos, S., McIlraith, S. A., Sohrabi, S., Mylopoulos, J., 2010. Integrating preferences into goal models for requirements engineering. In: Proceedings of the 18th IEEE International Requirements Engineering Conference. pp. 135–144.

Morin, B., Barais, O., Jezequel, J.-M., Fleurey, F., Solberg, A., 2009. Models@run.time to support dynamic adaptation. Computer 42 (10), 44–51.

Mylopoulos, J., Chung, L., Nixon, B., 1992. Representing and using nonfunctional requirements: A process-oriented approach. IEEE Trans. Softw. Eng. 18 (6), 483–497.

Peng, X., Chen, B., Yu, Y., Zhao, W., 2012. Self-tuning of software systems through dynamic quality tradeoff and value-based feedback control loop. J. Syst. Software 85 (12), 2707–2719.

Ramirez, A. J., Cheng, B. H. C., Bencomo, N., Sawyer, P., 2012. Relaxing claims: Coping with uncertainty while evaluating assumptions at run time. In: Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems. pp. 53–69.

Salay, R., Chechik, M., Horkoff, J., 2012. Managing requirements uncertainty with partial models. In: Proceedings of the 20th IEEE International Requirements Engineering Conference. pp. 1–10.

Salehie, M., Pasquale, L., Omoronyia, I., Ali, R., Nuseibeh, B., 2012. Requirements-driven adaptive security: Protecting variable assets at runtime. In: Proceedings of the 20th IEEE International Requirements Engineering Conference. pp. 111–120.

Salehie, M., Tahvildari, L., 2009. Self-adaptive software: Landscape and research challenges. ACM Trans. Auton. Adapt. Syst. 4 (2), 14:1–14:42.

Sawyer, P., Bencomo, N., Whittle, J., Letier, E., Finkelstein, A., 2010. Requirements-aware systems: A research agenda for RE for self-adaptive systems. In: Proceedings of the 18th IEEE International Requirements Engineering Conference. pp. 95–103.

Sebastiani, R., Giorgini, P., Mylopoulos, J., 2004. Simple and minimum-cost satisfiability for goal models. In: Proceedings of the 16th International Conference on Advanced Information Systems Engineering. pp. 20–35.

Song, H., Huang, G., Chauvel, F., Xiong, Y., Hu, Z., Sun, Y., Mei, H., 2011. Supporting runtime software architecture: A bidirectional-transformation-based approach. J. Syst. Softw. 84 (5), 711–723.

Souza, V. E. S., Lapouchnian, A., Mylopoulos, J., 2011. System identification for adaptive software systems: a requirements engineering perspective. In: Proceedings of the 30th International Conference on Conceptual Modeling. pp. 346–361.

Souza, V. E. S., Lapouchnian, A., Mylopoulos, J., 2012. Requirements-driven qualitative adaptation. In: On the Move to Meaningful Internet Systems: OTM 2012. pp. 342–361.

Sykes, D., Corapi, D., Magee, J., Kramer, J., Russo, A., Inoue, K., 2013. Learning revised models for planning in adaptive systems. In: Proceedings of the 2013 International Conference on Software Engineering. pp. 63–71.

van Lamsweerde, A., Letier, E., 2002. From object orientation to goal orientation: A paradigm shift for requirements engineering. In: Proceedings of the 9th International Workshop on Radical Innovations of Software and Systems Engineering in the Future, Revised Papers. pp. 325–340.

Wang, Y., Mylopoulos, J., 2009. Self-repair through reconfiguration: A requirements engineering approach. In: Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering. pp. 257–268.

Welsh, K., Sawyer, P., Bencomo, N., 2011. Towards requirements aware systems: Run-time resolution of design-time assumptions. In: Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering. pp. 560–563.

Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. C., Bruel, J.-M., 2009. RELAX: Incorporating uncertainty into the specification of self-adaptive systems. In: Proceedings of the 17th IEEE International Requirements Engineering Conference. pp. 79–88.

Yang, H., De Roeck, A., Gervasi, V., Willis, A., Nuseibeh, B., 2012. Speculative requirements: Automatic detection of uncertainty in natural language requirements. In: Proceedings of the 20th IEEE International Requirements Engineering Conference. pp. 11 –20.

Zadeh, L. A., 1965. Fuzzy sets. Information and Control 8 (3), 338–353.