

# Self-Tuning of Software Systems through Goal-based Feedback Loop Control

Xin Peng<sup>1</sup>, Bihuan Chen<sup>1</sup>, Yijun Yu<sup>2</sup>, Wenyun Zhao<sup>1</sup>

<sup>1</sup> School of Computer Science, Fudan University, Shanghai, China

<sup>2</sup> Department of Computing, The Open University, Milton Keynes, UK

pengxin@fudan.edu.cn, 09210240005@fudan.edu.cn, y.yu@open.ac.uk, wyzhao@fudan.edu.cn

**Abstract**—Quality requirements of a software system cannot be optimally met, especially when it is running in an uncertain and changing environment. In principle, a controller at runtime can monitor the change impact on quality requirements of the system, update the expectations and priorities from the environment, and take reasonable actions to improve the overall satisfaction. In practice, however, existing controllers are mostly designed for tuning low-level performance indicators rather than high-level requirements. By linking the overall satisfaction to a business value indicator as feedback, we propose a control-theoretic self-tuning method that can dynamically adjust the tradeoff decisions among different quality requirements. A preference-based reasoning algorithm is involved to configure hard goals accordingly to guide the following architecture reconfiguration.

**Keywords**—control theory; goal reasoning; self-tuning

## I. INTRODUCTION

Once deployed, software systems in production are no longer as easy to configure and change as they were inside the development houses. Therefore, *self-tuning* mechanisms with little human intervention are more practical to find optimal configurations for dynamic quality requirements. Considering alternative architectures and designs, different quality attributes require different optimizations, and often the better one in certain quality dimensions is the worse in others [1]. Therefore, quality tradeoff decisions should be adjusted dynamically at runtime to maximise the overall satisfaction.

After a decade of exploration, an emerging architecture for self-managing systems was proposed by software engineers [2, 3], which separates the design of such systems into three vertical layers: *goals*, *plans* and *components*. Instead of using high-level requirements, however, existing self-tuning mechanisms were using low-level performance indicators such as load, memory, bandwidth, and even characteristics of user profiles such as operational error rates [4]. There is clearly an uncertain gap between goal satisfaction and quantifiable metrics [5].

To link the satisfaction of high-level requirements down to the tuning parameters of the running systems, one key mechanism is to consider using feedback loops as proposed in software cybernetics [6]. Earlier we have reported a framework to optimize software quality at runtime [7]. The self-tuning method proposed there was based on low-level control parameters. In this work, we combine goal models with feedback loop controllers to make dynamic trade-offs among conflicting softgoals (i.e.,

the goals with no binary satisfaction criteria). Reflecting the business value of customers, our controller adjusts the preference ranks of softgoals on basis of runtime feedback. The *preference rank* of a softgoal  $S_1$  is defined by a priority number  $R_1$ , indicating its importance relative to another softgoal  $S_2$  of rank  $R_2$ :  $S_1$  is preferred to  $S_2$ , if and only if  $R_1 > R_2$ . Intuitively, when it is impossible to meet all the expectations on the individual quality requirements, the softgoals with lower preference ranks will be conceded until the remaining softgoals are achievable.

Having the traceability from goal models to design alternatives, *intentional variability* expressed by OR-refinement of goals is used in our approach to support dynamic quality tradeoffs among the alternatives, in response to the changing environment.

## II. PRELIMINARIES

Our method is founded on goal-oriented reasoning and feedback control theory.

### A. Goal-Oriented Modelling and Reasoning

Goal-oriented modelling methodologies such as [8, 9] elicit and refine goals into requirements at the leaf level. Some requirements have clear-cut satisfaction criteria (modelled as hard goals) whilst others do not (modelled as softgoals). After obtaining the goal models, softgoals can be evaluated formally through reasoning. A general way of goal-based reasoning considers the entire goal model as a set of logic constraints in conjunctive normal form, which encodes both refinement and contribution rules. The goal-based reasoning discretises a softgoal into propositions of four levels of satisfaction such that they can be reasoned together with the hard goals in proposition logic [10].

### B. Feedback Control Theory and PID Controller

The basic structure of a feedback control loop is presented in Figure 1. Technical experts can specify a *set point*, the desired characteristics of the system or the monitored *process*. An *output*, or the controlled behaviour of a system, provides feedback to the *controller*. The controller tries to minimise the *error*, or delta, between the *set point* and *output*, to control the *process* such that the *output* is close to the *set point*.

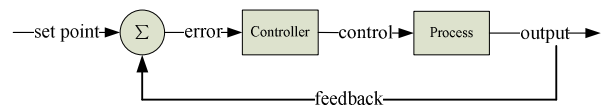


Figure 1. Basic Structure of Feedback Control Loop

Core to the design of a feedback control loop is the choice of the *controller*. We adopt the PID (proportional-integral-derivative) controller [11] in consideration of its precision, stability, and responsiveness. By precision the average error is low; by stability the standard deviation of errors is low; and by responsiveness the delay to obtain output from the set points and feedback is fast. In the absence of knowledge of the underlying process, PID controllers usually are the best [12]. This is consistent to our problem due to the uncertainty of a changing environment and a series of conflicting quality requirements. To achieve a desired level of satisfaction for quality requirements, PID controller takes into account the current behaviour through proportional control, the past behaviour through integral control, and future behaviour through derivative control. In other words, it characterises the temporal behaviour of a process. PID controller synthesizes the benefits from P, PI and PD controllers and overcomes the drawbacks such as P controller’s low stability, PI controller’s high overshoot, and PD controller’s slow response [11]. In the domain of software tuning, PID controllers have already been applied successfully in scheduling in real-time operating systems [13], automatic stress and load testing tools [14] and performance optimization [7].

The incremental formulation of a PID controller can be specified as follows:

$$u(t) = u(t-1) + Kp * (e(t) - e(t-1)) + Ki * e(t) + Kd * (e(t) - 2 * e(t-1) + e(t-2))$$

where  $u(t)$  is the control variable of the controller at time  $t$ ,  $e(t) = \text{set point} - \text{output}$  is the error signal at time  $t$ ,  $Kp$ ,  $Ki$ , and  $Kd$  are the parameters of proportional control, integral control and derivative control respectively. All these control parameters are constants specified at the design time, which vary from one control system to another [15].

### III. OUR METHOD

#### A. Method Overview

To implement a requirements-driven self-tuning system, our method needs to continuously seek opportunities to improve the overall performance. Figure 2 presents an overview: a running system together with the goal reasoner and the architecture configurator form the process under the control. Due to conflicts among softgoals, it is often impossible to optimise all the softgoals individually. Therefore we need to dynamically adjust the tradeoff decisions, i.e. the preference ranks of related softgoals, to maximise the overall satisfaction. In order to provide such runtime feedback reflecting an overall satisfaction, we choose a measurable value indicator from the business perspective and monitor it at runtime.

Intuitively, the controller is designed to prevent a softgoal from getting worse by increasing its preference rank such that different choice in the plans may be found. In other words, every softgoal has an approximately proportional relationship between the preference rank and the satisfaction. For example, if the response time is

getting too long, the rank of the softgoal “minimal response time” should be increased. By defining a value indicator reflecting the overall satisfaction as the set point and the feedback, our PID controller is designed to adjust the process by tuning the preference ranks of related softgoals to guide the goal reasoning and the following architecture reconfiguration.

Note that both the preference ranks of softgoals and the monitored satisfaction level of a softgoal are changeable in a runtime system. Following the dynamically tuned preference ranks, our goal reasoner first generates a set of configurations that optimise the achievement of high-ranked softgoals. Each configuration is a selection of leaf-level goals. One “best” configuration is then chosen by the “minimum distance” principle to make the adaptation smoother. The distance between the next goal configuration and the current one is calculated based on the similarity of the configurations, i.e., minimal or no change is required if the two configurations deliver the same level of overall satisfaction. Next the architecture configurator executes the adaptation by reconfiguring the runtime architecture according to the selected goals and the mappings between the goals and the architectural components. The components corresponding to newly selected goals are bound and integrated, while the components corresponding to those eliminated goals are removed. Such architecture reconfigurations are supported by the service-oriented architecture and a reflective component model.

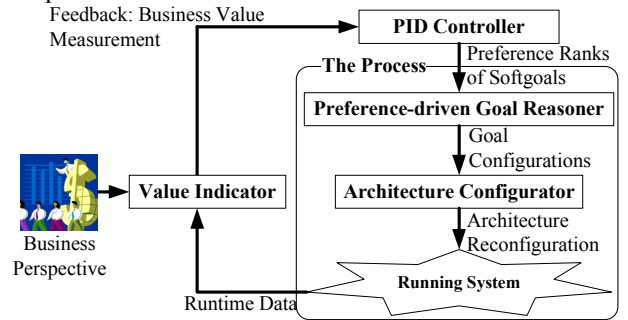


Figure 2. An Overview of Our Method

#### B. A Running Example and Its Requirements Model

The running example to illustrate our method is an online course registration system. It provides a Web-based public service for examinees to register for periodically held public examinations in China. Some functional and quality requirements of the system are shown in the goal model (Figure 3). In a typical course registration scenario, an examinee first selects the courses to study (“COURSE SELECTION”), and then pays registration fees for the selected courses (“PAYMENT”). Each course selection will be checked by examination rules, and only students who have their course registration fees successfully paid will be accepted by the system.

To achieve these goals, there are multiple alternatives. For the course selection, an examinee can submit a course registration form either by choosing from the course listed

in a complex form (“LIST AND CHOOSE”) or by simply entering the course code in a text field (“INPUT COURSE ID”). For the course payment, two third-party payment services “ABBPAY” and “UNLPAY” are currently available in China, and either of them supports online payment by credit or debit cards.

The system runs in a *Software as a Service* (SaaS) fashion: the service provider runs the system under a contract with education administration and gains profits from every successful course registration. According to this business value proposition, four quality requirements have been elicited, shown as softgoals in Figure 3. These softgoals contribute to the business value from different perspectives: “minimal response time” is demanded for higher system throughput under a given bandwidth; “minimal error rate” is aimed for higher rate of valid course selection; “minimal cost” is aimed for the lower third-party cost; and “availability” is targeted at higher success rate of payment transactions. The alternatives identified for the “COURSE SELECTION” or “PAYMENT” goals can exert different influences (e.g., Help, Hurt) on these softgoals.

Here the system has an explicit indicator for its business value, i.e. the course registration profit in unit time, which amounts to the gain of fees from successful course registrations subtracting the cost paid to third-party payment providers.

Considering environmental changes to the accessing load and the quality of third-party services, dynamic tradeoff decisions must be made to maximise the business value. For example, when the load is heavy, this goal model may be configured to ensure “minimal response time” over “minimal error rate”; when the load is light, adverse tradeoff decision can be better for a better business value. Since it is hard to specify precise relationships between the changes of preference ranks of related softgoals and the fluctuation of the business value, we use a PID controller to tune the preference ranks dynamically, with the business value as the set point and the feedback.

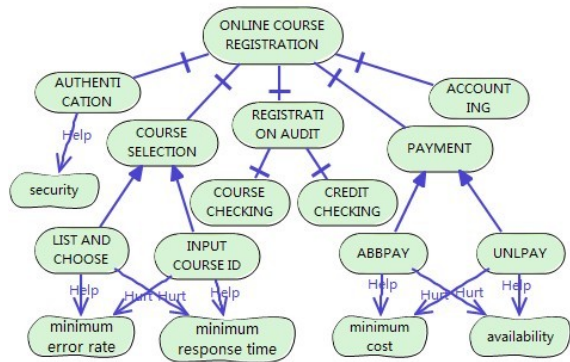


Figure 3. Requirements of the Online Course Registration System

### C. Preference-based Goal Reasoning

One contribution of this work is the adaptation of existing qualitative goal reasoning frameworks to business value propositions that are quantitative. The preference

ranks assigned to the softgoals form a partial ordering. The Preference Based Goal Reasoning algorithm has been implemented in the new version of OpenOME. The procedure takes as input the given goal model, including a set of hard goals and softgoals, the refinement and contribution relationships, as well as the quantitative labels of expected satisfaction. In addition to these input, dynamic quality tradeoff decisions are expressed by preference ranks of the softgoals because the algorithm does not differentiate two softgoals of the same rank. The preference ranks are changeable input tuned by the feedback controller. The algorithm has an iterative step to invoke a SAT solver for finite times, depending on the number of ranks needed to express the partial ordering. The procedure tries initially to find a configuration that can accommodate all softgoals to reach their expectations. If not possible, the lowest ranked softgoals will be removed from the encoding so as to accommodate the remaining softgoals, and so on, until either a viable configuration is found or all softgoals have been removed. The algorithm terminates with a valid configuration because we assume the root hard goal is satisfied by design.

Our encoding extends the original goal reasoning work [16] by discretising both satisficing and denial of softgoals of 3 labels (fully, partially, unknown) into any  $N > 3$  labels, depending on the level of details one would tune. In practice, we find  $N = 5$  a reasonable value, which means that the interval in  $[0, 1]$  is 0.1. As reported earlier [5], since goals are expressed as desired states of subject domains, different domains may require different degree of discrimination. In this experiment, we aim to see whether a relatively small discrimination can still be helpful when the reasoning is combined with the PID controllers.

**Algorithm 1.** Preference Based Goal Reasoning

**input:** a goal model with expectations, plus the feedback in terms of updated preference ranks of some softgoals

- elements  $g, s$ : set of hard goals/softgoals
- decomposition relations  $d: g \times g \times \{ \text{AND, OR} \}$
- contribution relations  $c: g \times s \times \{ +, - \} \times (0, 1]$
- expectations  $e: g \cup s \times [0, 1]$
- labels  $l: s \times [0, 1]$
- preference ranks  $r: s \times Z$

**output:** a set of “good enough” configurations

- If any, output configurations  $o: 2^{g \cup s}$

**procedure**

**begin**

1. Encode  $g, s, d, c, e, l$  into a CNF proposition formula  $\Phi$  w.r.t. [12]
2. solved = false; rank = 0
3. **while not solved and rank < MAX(r)** **do**
4.   solved = solve  $\Phi$  using SAT solver
5.   **if not solved then**
6.      $\Phi$  = remove rules in  $\Phi$  concerning  $q \in s$  where  $r(q) < rank$
7.   **else**
8.      $o$  = decode propositions from  $\Phi$  concerning satisfied elements in  $g \cup s$
9.   **end if**
10. **end do**
11. **if solved then return o**
12. **else return nil end if**

**end**

#### D. Modified-PID-based Self-tuning Algorithm

The challenge for the feedback controller is to balance the preference ranks of related softgoals while making the adaptation results smoother with less turbulence. To address the challenge, our Modified-PID-based Self-tuning algorithm returns the tuned preference ranks of softgoals in response to the changing environment.

We made three adaptations to the PID control model introduced in Section II to fit our specific control problem. First, set points are initially unspecified, and then moved gradually from old values to newly specified ones that reflecting the actual status of the runtime environment. Thus, we evaluate set points as the average of all the past values of corresponding performance indicators (Lines 3 and 12). Second, the delta of business value is calculated as the percentage of increment/decrement (Line 2), which determines whether any self-tuning is needed. The error signal of softgoal represents the percentage of its deviation from set points (Line 8), and the control variable represents the percentage of increment/decrement of the preference rank based on the past error signals (Lines 9). Third, a dead band of the delta of the business value, or a tolerable range, is used to avoid oscillations in frequent architecture reconfigurations. More specifically, if the delta of the business value is within the defined dead band, then the preference ranks will not be tuned (Line 4). Otherwise, the preference ranks should be tuned based on the control variables (Line 14).

```

Algorithm 2 Modified-PID-based Self-tuning
input: array curVal
output: array ranks
procedure
begin
1. calculate the delta of the business value:
2.  $\text{delta} = (\text{curVal}[0] - \text{set-points}[0]) / (\text{set-points}[0]) * \text{marks}[0]$ 
3. calculate the set-points of business value (average of the past values)
4. if ( $\text{delta} < \alpha$ ) then  $\text{ranks}[0] = -1$  //no self-tuning is needed
5. else  $\text{ranks}[0] = -2$  //self-tuning is needed end if
6. calculate error signal and control variables:
7. for ( $i = 1; i < \text{curVal.length}; i++$ ) do
8.  $e(k)[i] = (\text{set-points}[i] - \text{curVal}[i]) / \text{set-points}[i] * \text{marks}[i]$ 
9.  $\text{conVar}[i] = \text{conVar}[i] + \text{KP} * (e(k)[i] - e(k-1)[i]) + \text{KI} * e(k)[i]$ 
    $+ \text{KD} * (e(k)[i] - 2 * e(k-1)[i] + e(k-2)[i])$  //PID
10.  $e(k-2)[i] = e(k-1)[i]$ 
11.  $e(k-1)[i] = e(k)[i]$ 
12. calculate set-points[i] as average of the past values
13. if ( $\text{ranks}[0] == -2$ ) then
14.  $\text{ranks}[i] = \text{round}(\text{conVar}[i] * \text{ranks}[i]) + \text{ranks}[i]$ 
15. end if
16. if ( $\text{ranks}[i] > 10$ ) then  $\text{ranks}[i] = 10$ 
17. else if  $\text{ranks}[i] < 1$  then  $\text{ranks}[i] = 1$  end if
18. end do
19. return ranks
end

```

#### IV. CONCLUSIONS

Runtime quality tradeoff algorithms have been proposed to improve the overall satisfaction of the requirements. Our experimental study on a Web-based system has validated that combining PID control theory

with preference-based goal reasoning is effective in runtime self-tuning for a real-life software system.

#### ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China under Grant No. 90818009, Shanghai Committee of Science and Technology, China under Grant No. 08DZ2271800 and 09DZ2272800, Shanghai Leading Academic Discipline Project under Grant No. B114.

#### REFERENCES

- [1] R. Kazman, M. Klein, and P. Clements, "ATAM: Method for Architecture Evaluation," CMU/SEI-2000-TR-004 ESC-TR-2000-004, August 2000.
- [2] B. H. C. Cheng et al, "Software Engineering for Self-Adaptive Systems: A Research Roadmap," in Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar], 2009.
- [3] J. Kramer and J. Magee, "A Rigorous Architectural Approach to Adaptive Software Engineering," J. Comput. Sci. Technol., vol. 24, pp. 183-188, 2009.
- [4] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," Computer, vol. 36, pp. 41-50, 2003.
- [5] E. Letier and A. v. Lamsweerde, "Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering," in Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2004.
- [6] K. Cai, J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur, "An Overview of Software Cybernetics," 11th International Workshop on Software Technology and Engineering Practice, 2004.
- [7] B. Chen, X. Peng, and W. Zhao, "Towards Runtime Optimization of Software Quality Based on Feedback Control Theory," in Proceedings of the First Asia-Pacific Symposium on Internetwork, 2009.
- [8] A. v. Lamsweerde and E. Letier, "From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering," in Radical Innovations of Software and Systems Engineering in the Future, 2002.
- [9] J. Castro, M. Kolp, and J. Mylopoulos, "Towards Requirements-driven Information Systems Engineering: the Tropos Project," Inf. Syst., vol. 27, pp. 365-389, 2002.
- [10] R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Simple and Minimum-Cost Satisfiability for Goal Models," in Advanced Information Systems Engineering, 16th International Conference, CAiSE 2004.
- [11] G. F. Franklin, J. D. Powell, and A. E. Naeini, Feedback Control of Dynamic Systems, 5th ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2006.
- [12] S. Bennett, A History of Control Engineering 1930 - 1955, 1st ed., Hitchin, Herts., UK, UK: Peter Peregrinus, 1993.
- [13] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A Feedback-driven Proportion Allocator for Real-rate Scheduling," In Operating Systems Design and Implementation, pp. 145-158, 1999.
- [14] M. Bayan and J. W. Cangussu, "Automatic Feedback, Control-based, Stress and Load Testing," 23rd Annual ACM Symposium on Applied Computing, 2008.
- [15] J. A. Shaw, "Pid Algorithms and Tuning Methods," <http://www.jashaw.com/pid/tutorial/>.
- [16] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Formal Reasoning Techniques for Goal Models," Journal on Data Semantics I, vol. 1, pp. 1-20, 2003.