Autotuning control structures for reliability-driven dynamic binding

Antonio Filieri, Carlo Ghezzi, Alberto Leva and Martina Maggio

***Abstract*— This paper explores a formally grounded approach to solve the problem of dynamic binding in service-oriented software architecture. Dynamic binding is a widely adopted mean to automatically bind exposed software interfaces to actual implementations. The execution of an operation on one or another implementation, though providing the same result, could turn out in different quality of service, e.g. due to failure occurrence. Dynamic binding is thus of primary importance to achieve what in the Software Engineering domain is called "self-adaptiveness", the capability to preserve a desired quality of service, if this is feasible. It is important to reach this goal also in the presence of environmental fluctuations – a route congestion increase – or even abrupt variations – a server breakdown. A quite general dynamic binding problem is here reformulated as a discrete-time feedback control one, and the use of autotuning techniques is discussed, extending previous research, in a view to guaranteeing the desired quality of service without the need for computationally-intensive optimisations.**

I. Introduction

Software systems nowadays are required to live in highly dynamic environments. Deployment infrastructures, usage profiles, and the behaviour of often crucial third-party components, are subject to continuous and unpredictable changes. Software is required to survive such changes maintaining the required Quality of Service (QoS), by adapting its behavior online to compensate possible environmental threats. Self-adaptive software is therefore a growing research field in Software Engineering [4].

Many proposed approaches are grounded on Service Oriented Architectures (SOA), that is, software is built by composing abstract services, that provide functionalities contributing to the achievement of the global goal. Each abstract service could be provided by multiple implementations. The binding of a concrete implementation to the abstract service is a key factor in the adaptation of SOA systems, and the subject of this paper.

More specifically, the focus here is on the use of simple but effective control-theoretical tools to support binding decisions driven by "reliability", defined as the probability of successfully accomplishing an assigned task, whatever the term "success" means in the particular application at hand—e.g., it could mean to enforce a response time compliant with the service level agreement. The encountered problems can be formulated as set point tracking ones in the presence of disturbances, operating in the discrete time domain. Such an approach requires an initial modelling effort, since that class of systems does not appear – at least, at a first glance – to be the most natural one. However, the resulting problem formulation allows for significantly simpler solutions with respect to alternatives such as Markov chains and/or queue networks, without giving up on generality.

The companion paper [5] views the matter essentially from the software engineering side, evidencing the mentioned increased simplicity by means of a quite extensive review of related work, and discussing various implementation-related facts. This paper takes conversely the control engineering point of view, recalling the major modelling aspect for the reader's convenience, and then concentrating on the application of autotuning by discussing the technique choice and the overall procedure structuring. Application examples are finally described, where an autotuning mechanism is applied to both the two- and the multiple-alternatives dynamic binding cases. Dynamic binding is the ability to route an arriving request to different services that provide the correct answer to the user. Using one service or another could depend in principle on a variety of different reasons. To start, the cost of the service can be taken into account; the load of the service infrastructure can be another deciding variable; the reliability of each service implementation is here taken into account to offer an overall reliable experience to the end user.

A. Filieri, C. Ghezzi, A. Leva, Politecnico di Milano, Dipartimento di Elettronica e Informazione, Via Ponzio 34/5, 20133 Milano, Italy, {filieri,ghezzi,leva}@elet.polimi.it

M. Maggio, Department of Automatic Control, Lund University, Ole Römers väg 1, 223 63, Lund, Sweden, martina.maggio@control.lth.se

II. Two-level dynamic binding via the autotuning of a SISO discrete time controller

To address dynamic binding in its generality, it is convenient to first break it down to its most elementary setting, namely the *two-level dynamic binding problem*. This problem can be formulated as follows: a certain service S has to be provided, two software components (possibly operated by third parties) s_1 and s_2 are available for that purpose. During the execution the QoS of the two components may change. This means that their reliability – in the sense of "success" probability – may change over time, due to service failures, overload and so forth. Service S should however offer a certain reliability level R for its clients, whenever possible. The goal is thus to dynamically distribute the requests for S between s_1 and s_2 so as to attain R, if feasible, and in the opposite case, to get as close as possible to R. To model this scenario, in the Software Engineering field, it is common to adopt a convenient Markov Chain, as the one in Figure 1, where requests enter the system at the initial node n_i (representing the entry point of the service S), with a rate of w_i requests per time unit, and are then routed to either of two implementations, s_1 and s_2. Each node s_j has a

success probability p_{sj}, and thus a failure one $p_{fj} = 1 - p_{sj}$. Nodes n_f and n_s represent the failure and the success state, corresponding to the happening of a failure or the successful completion of the task respectively. The control objective is to continuously adapt p_1 , to match or overcome an overall reliability goal (which is the global probability of reaching state n_s from state s_i). It is assumed – consistently with

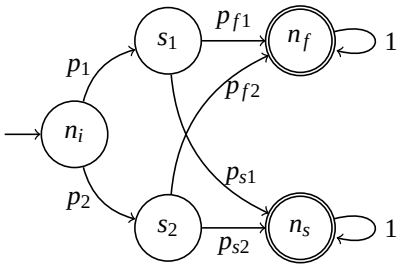


Fig. 1: Model representation for the basic load balancing example.

common practice – that reliability R is measured periodically, each T_s time units. The choice is here made to handle the problem with a discrete-time controller of sampling period T_s, and thus the first task to carry out is to devise, in this setting, a convenient model of the process to be controlled. Notice that, one can generalise the proposed case study to the *n-level dynamic binding* problem, where the choice is to be made among more than two components, the interested reader is referred to [5] and the papers quoted therein for an extensive problem formulation.

In the rest of this manuscript, first the two-level case is treated, introducing the used modelling and control framework without loss of generality; then, building on the found solution, the n-level extension is dealt with.

A. The controlled system's model

Assume, quite realistically, that each node is endowed with a request queue, thus the number of requests in all the queues is the controlled system's state vector $\mathbf{n}(k)$. One dispatch probability – in the example, $p_1(k)$ – is the control variable, while input rates – in the example the sole $w_i(k)$, but his causes no generality loss – act as disturbances; the controlled variable is the overall reliability. Other probabilities – in the example, p_{s1} and p_{s2} are considered time-varying parameters. Also, suppose that p_{s1} and p_{s2} undergo variations that viewed the T_s time scale are modest and slow, interspersed with large and abrupt – but sporadic – changes caused e.g. by node failures. Thus, quite accurate estimates of p_{s1} and p_{s2} are most frequently available based on the observed nodes' success and failure rates, said estimates becoming highly unreliable only in the presence of the mentioned sporadic events. Finally, suppose that each node i has a maximum throughput of t_{mi} requests per control step of length T_s.

Denoting by m the number of nodes in the chain – five in the example – the (SISO) controlled system has the state equation

$$\begin{aligned} \mathbf{n}(k) &= \mathbf{n}(k-1) - \mathbf{r}(k-1) \\ &\quad + \mathscr{P}(k-1)\cdot\mathbf{r}(k-1) + \mathbf{w}(k-1) \\ \mathbf{r}(k) &= \min\{\mathbf{t_m}, \mathbf{n}(k)\} \end{aligned} \tag{1}$$

where boldface letters denote vectors. Each element of $\mathbf{w}$ is the number of requests entering the corresponding node in the control step. Vector $\mathbf{n} = [v_i\, v_1\, v_2\, v_s\, v_f]'$ is the state, while vector $\mathbf{r}$ contains the number of requests served by all the nodes at time k, $\mathbf{t_m}$ being the vector of the maximum node throughputs (possibly different for each node). Matrix $\mathscr{P}$ is finally the transition matrix of the chain. For the case of Figure 1, $\mathscr{P}$ takes the form

$$\mathscr{P}(k) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ p_1(k) & 0 & 0 & 0 & 0 \\ 1-p_1(k) & 0 & 0 & 0 & 0 \\ 0 & 1-p_{s1} & 1-p_{s2} & 1 & 0 \\ 0 & p_{s1} & p_{s2} & 0 & 1 \end{bmatrix} \tag{2}$$

while defining $\mathbf{y} = v_s$, the controlled system's output equation is

$$\mathbf{y}(k) = \mathbf{C}\mathbf{n}(k) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{n}(k). \tag{3}$$

For the moment, suppose that p_{s1} and p_{s2} are constant, for example at the values stipulated with the nodes via Service Level Agreements (SLAs). In this case (1,3) is a time-invariant SISO model, hereinafter indicated with $\mathscr{M}$, that however is nonlinear owing to the rate saturation and to an input-by-state product observed in (1).

To complete the model, it is necessary to represent the *measurement dynamics*, i.e., the output-to-metric model $\mathscr{M}_m$ cascaded to the "physical" system's one $\mathscr{M}$. Expressing the measured reliability as

$$q(k) = \frac{v_s(k) - v_s(k-1)}{v_s(k) + n_f(k) - v_s(k-1) - v_f(k-1)} \tag{4}$$

i.e., as the percentage of successful requests in the last control step, $\mathscr{M}_m$ is a system with input $\mathbf{u}_m$ and state $\mathbf{x}_m$ both given by $\mathbf{y}$, having the (linear) equation

$$\mathbf{x}_m(k) = \mathbf{u}_m(k-1) \tag{5}$$

as the state one, and (4) as the (nonlinear) output one. Given the assumptions made on parameter changes, the problem is naturally cast – except for when an abrupt event occurs – in the framework of control in the vicinity of an equilibrium for a system with moderate and slow variability,

Observing $\mathscr{M}$, it can be however verified that no equilibrium exists. This is correct, since output nodes apparently accumulate served requests indefinitely. If thus the equilibrium search is repeated neglecting said accumulation, i.e., replacing $\mathscr{P}$ with the reduced matrix

$$\mathscr{P}_{red} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ p_1 & 0 & 0 & 0 & 0 \\ 1-p_1 & 0 & 0 & 0 & 0 \\ 0 & 1-p_{s1} & 1-p_{s2} & 0 & 0 \\ 0 & p_{s1} & p_{s2} & 0 & 0 \end{bmatrix} \tag{6}$$

then an equilibrium is found as

$$\begin{aligned}\overline{\mathbf{n}} &= [\overline{\nu}_i\,\overline{\nu}_1\,\overline{\nu}_2\,\overline{\nu}_s\,\overline{\nu}_f]' = (I - P_{red}(\overline{\mathbf{u}}))^{-1}\,\overline{\mathbf{d}} \\ &= \begin{bmatrix} 1 \\ p_1 \\ 1-p_1 \\ p_1(1-p_{s1})+(1-p_1)(1-p_{s2}) \\ p_1 p_{s1}+(1-p_1)p_{s2} \end{bmatrix} \overline{w}_i \end{aligned} \tag{7}$$

if this satisfies the maximum throughput constraints, while in the opposite case there is no equilibrium because at least one queue grows indefinitely. Notice that (7) holds also for the original system $\mathscr{M}$, under the same feasibility hypothesis, just (re-)defining the reliability in the last control period by replacing (4) with

$$q(k) = \frac{v_s(k)}{v_s(k)+v_f(k)}. \tag{8}$$

This re-definition also makes $\mathscr{M}$ algebraic, so that the complete linearised model is

$$\begin{cases} \delta\mathbf{n}(k) &= \mathbf{A}\delta\mathbf{n}(k-1) \\ & \quad +\mathbf{B}_u\delta\mathbf{u}(k-1)+\mathbf{B}_d\delta\mathbf{d}(k-1) \\ \delta q(k) &= \mathbf{C}_m\delta\mathbf{n}(k) \end{cases} \tag{9}$$

where

$$\begin{aligned}\mathbf{C}_m &= \mathbf{D}_m\mathbf{C} \\ &= \begin{bmatrix} 0 & 0 & 0 & \frac{\overline{\nu}_f}{(\overline{\nu}_s+\overline{\nu}_f)^2} & -\frac{\overline{\nu}_s}{(\overline{\nu}_s+\overline{\nu}_f)^2} \end{bmatrix}. \end{aligned} \tag{10}$$

The transfer function from δp_1 to δq is thus readily computed as

$$\begin{aligned} P(z) &= \mathbf{C}_m(z\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}_u \\ &= \overline{\nu}_1\frac{\overline{\nu}_s(p_{s2}-p_{s1})-\overline{\nu}_f(p_{f2}-p_{f1})}{(\overline{\nu}_s+\overline{\nu}_f)^2}\frac{1}{z^2} \end{aligned} \tag{11}$$

and with trivial computations it reduces to

$$P(z) = \frac{p_{s2}-p_{s1}}{z^2}. \tag{12}$$

This reveals that the structure of the controlled dynamics is invariantly that of a two-steps delay, where however the sign of the gain may change. Assuming – it is worth stressing – that parameter variations are small and slow with respect to T_s, and that reliable estimates for said parameter are available, one can thus reason for control as follows:

- if estimates foresee no gain sign modification, apply a simple fixed-parameter controller suitably (auto)tuned when a new operating condition (i.e., a new desired reliability) is established, if the transient from the previous to the new condition did not prove satisfactorily shaped (plenty of methods for retuning necessity detection are available in the literature, see e.g. [3]);
- if conversely the gain is "dangerously" approaching zero – i.e., if binding variations loose efficacy as with equal success probabilities no routing modification can alter the overall reliability – refrain from altering the routing until a "sufficiently nonzero" gain is forecast, and then re-tune the controller.

B. Controller tuning

The requirement of reaching at least a reliability level of $\overline{q}$ in $[0,1]$ at time k can be attained by a feedback controller aiming at a reliability set point $\overline{q}(k)$, typically set slightly above $\overline{q}$ to accommodate for the unavoidable small fluctuations at operation time. Zero steady-state error and a high degree of stability can be achieved by a PI controller, here written as

$$\begin{aligned} u_i(k) &= u_i(k-1)+a(1-b)e(k-1) \\ p_1(k) &= u_i(k)+ae(k) \end{aligned} \tag{13}$$

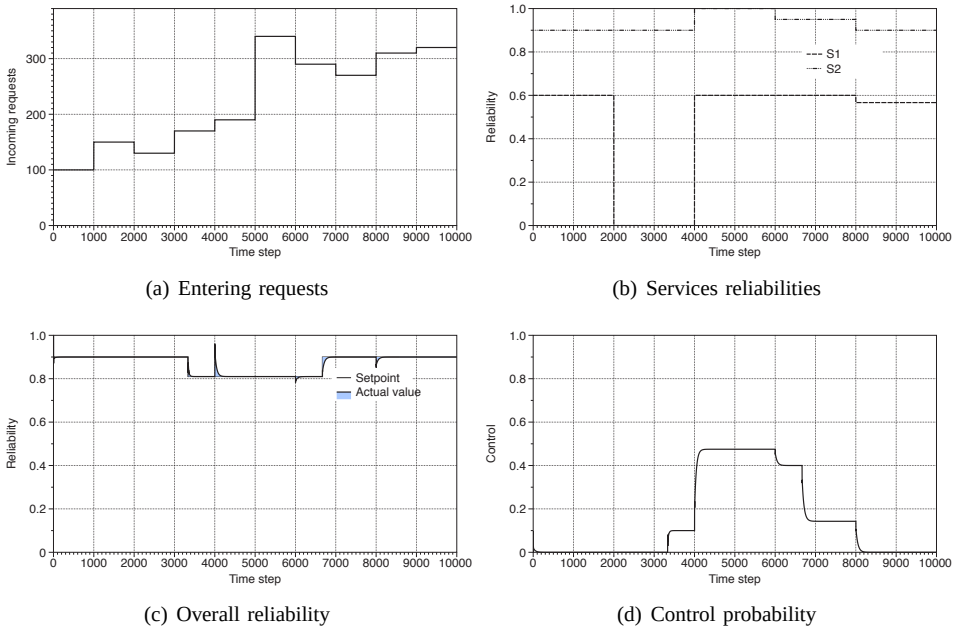
where $e(k) = \overline{q}(k)-q(k)$ is the error. Once a and b are chosen, the Jury criterion [10] reveals that asymptotic stability of the closed-loop system composed of (12) and (13) holds for any value d of $p_{s2}-p_{s1}$, obviously in the range $(-1,1)$, such that

$$\begin{cases} \dfrac{1-abd}{1+abd} & > 0 \\ \dfrac{(a^2b^2d^2-abd+ad-1)(a^2b^2d^2+abd-ad-1)}{(1-abd1)(1+abd)} & > 0 \\ \dfrac{ad(1-b)(abd+ad+2)(a^2b^2d^2-abd+ad-1)}{a^2b^2d^2+abd-ad-1} & > 0 \end{cases} \tag{14}$$

Studying (14) it can be noticed that for a wide range of (a,b) values, stability is preserved under the sole condition $ad > 0$, thus that even relevant estimation errors for d do not produce disrupting effects if at least the sign is caught (under the assumptions above, remember). Of course, this is not true for control *performance*: the time required to recover from a disturbance can degrade significantly if the estimation of d is not good enough, for example, whence the need for retuning if the detected error is too large.

Given the remarks just made, and also to allow the goal to be attained by system administrators who quite often have hardly any knowledge of control, an autotuning PI controller is here employed. The reason why autotuning was preferred to continuous adaptation resides in the type of encountered variations (see again the considerations reported before), and also in the lower computational effort. As anticipated, a logic may be activated to force autotuning if the error is "too large", and requesting the operator's intervention if the tuning operation was not successful. In the used procedure, autotuning is simply triggered when the integrated absolute error, at constant set point, exceeds a pre-specified threshold, which is quite common practice, or when a set point variation causes a controlled variable overshoot that is by another threshold bigger than the expected one (see below for the meaning of "expected"). Here too, in any case see [3] for a vast choice of alternative methods.

To choose the autotuning technique so as to structure and finalise the procedure, the following considerations were made. First, for a number of reasons too long to report here but inessential for this particular paper, it is preferable to keep the loop always somehow closed, thus relay-based



(a) Entering requests

(b) Services reliabilities

(c) Overall reliability

(d) Control probability

Fig. 2: Sample of the simulation tests for two-level control.

identification is used: in particular, to also avoid stepwise *stimuli*, the (linearised) process frequency response point with phase $-\pi/2$ is found with a relay-plus-integrator test. Second, to quantify how "slightly above" the set point has to be with respect to $\overline{q}$, reliable forecasts have to be provided for the closed-loop controlled variable's transients' shape, with particular reference to overshoots caused by set point steps, and peak deviations caused by step-like disturbances.

The above remarks oriented the choice to the recently introduced *contextual tuning* relay-based autotuning approach [9]. Details on that can be found in the quoted reference, and are omitted here for brevity. Suffice to say that the chosen autotuner provides the PI parameters and at the same time a first-order model with delay that precisely describes the process in the vicinity of the achieved cutoff frequency, the model's frequency response being exact at that frequency. This apparently allows to reliably estimate the main characteristics (peak and duration, typically) of the closed-loop transients of interest.

In practice, denoting by N the number of samples (at sampling time T_s) of the induced oscillation and by A the measured magnitude of the process Nyquist curve point with phase $-\pi/2$, (13) is tuned by setting

$$\begin{aligned} a &= \frac{2\pi\tan\varphi_m}{AT_s\left(2\pi+N\tan\varphi_m\right)\sqrt{1+\tan^2\varphi_m}} \\ b &= \frac{N\tan\varphi_m}{2\pi+N\tan\varphi_m} \end{aligned} \tag{15}$$

where φ_m is the desired phase margin in radians. As explained in the quoted reference, by just reformulating its results in the discrete time, also an approximated process model precise near the cutoff is obtained, that joined to the tuned PI allows to forecast the closed-loop transients of the controlled variable based on an estimated complementary sensitivity function given by

$$\widehat{T}(z) = \frac{2\pi}{Nz^q - Nz^{q-1} + 2\pi} \tag{16}$$

where q is the nearest integer to the quantity $N(1/4-\varphi_m/2\pi)$ —an acceptable approximation according to experience. Also, the settling time of the so tuned closed loop is readily estimated, in samples, as $5N/2\pi$.

C. Validation tests

Prior to implementing the control policy in a real software system, a simulation campaign was conducted in MATLAB; a sample of the results is shown in Figure 2. The test spans 10000 steps, each node can serve a maximum of 100 requests per step, and a reliability of 0.9 is desired. The initial failure probability of the first service is 0.4, while that of the second one is 0.1.

To see how the controller reacts to changes in the set point, the simulations run is divided into three parts, and the requested reliability is diminished to 0.8 in the second one. Also, the success and failure probabilities of the services were changed so as to divide the run in five parts, for example simulating the failure of the first service between steps 2000 and 4000. Finally, the number of requests entering the system is changed according to a predefined pattern. As can be seen in Figure 2(c), and in many other tests here omitted, the system is actually capable of withstanding the envisaged upset whenever feasible, and in general also more severe stimulations than the discussed theory rigorously allows to consider acceptable.

III. Extension to n-level dynamic binding

The two-level case just treated is naturally keen to become the basic brick for a modular construction of the *n*-level one.

To this end, consider the structure shown in Figure 3, where the added "intermediate" nodes are fictitious, since they are used only to compute the probabilities of routing from the single input node to the n possible targets.

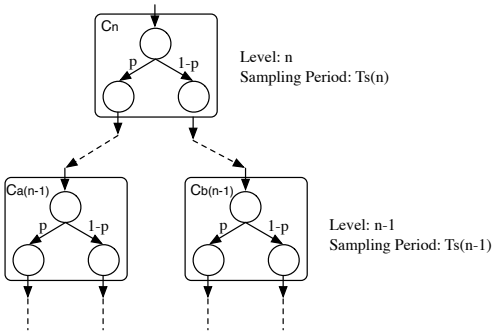


Fig. 3: Example of n-level binding structure.

To ensure proper operation, tuning starts from the output layer (that leading to the physical service nodes). Once all that layer is tuned, which can be done simultaneously for all its two-level blocks, the contextual tuning forecasts allow to estimate the settling time for the entire layer as the maximum among those of the blocks. The preceding layer is then tuned, after modifying its sampling time to be a multiple of that for the output layer, chosen as the smallest multiple greater than the estimated common settling time.

Repeating the procedure up to the input node, a multirate multivariable controller is therefore obtained. At present a complete analysis of that scheme is still underway, thus no rigorous guarantees on its characteristics are available nor can one forecast any upper limit for the ratio between the highest and the lowest sampling times. In all the encountered cases, however, the scheme proved to work satisfactorily, and no sampling time ratios were observed that are so large to impair an effective operation of the overall system. Hence, the so devised multirate control and tuning procedure seem to naturally generalise to more complex selection trees, which is the ultimate goal for the addressed application.

A. Validation tests

Also in this case, a couple of samples from a simulation campaign is reported. More specifically, Figure 4 shows the response of a five-level binder to a set point step variation and its rapid convergence.

Figure 5 conversely shows the behaviour of a 4-level binder subject to different stimuli. The effect of autotuining operations is shown to evidence how acceptable the system upset is. Specifically, the four services reliabilities are

$$\begin{aligned} av_1 &= 1-(0.4+0.6\cdot stp(k-9000)-0.6\cdot stp(k-9500) \\ &\quad +0.6\cdot stp(k-7000)-0.6\cdot stp(k-7500)) \\ av_2 &= 1-(0.1+0.3\cdot stp(k-2000)-0.3\cdot stp(k-2500)) \\ av_3 &= 1-(0.9+0.1\cdot stp(k-2000)-0.15\cdot stp(k-4500)) \\ av_4 &= 1-(0.7+0.1\cdot stp(k-1900)-0.1\cdot stp(k-4400)) \end{aligned} \tag{17}$$

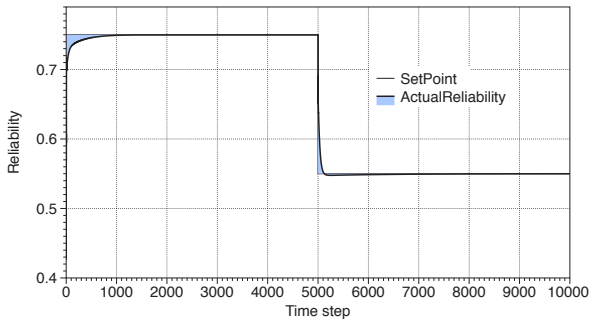


Fig. 4: Step response of a 5-level binder.

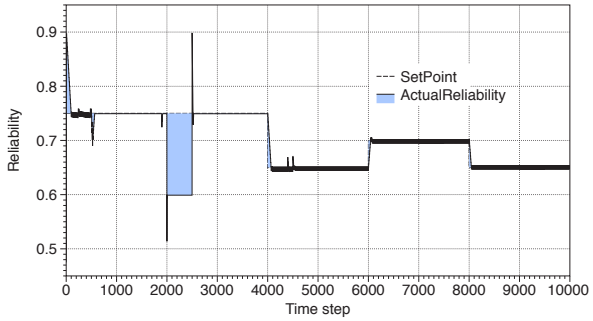


Fig. 5: Response of a 4-level binder with autotuning and different *stimuli*.

where av_x is the availability (or reliability) of service x and stp is the step function. This results in the control objective not being feasible in the interval between time step 2000 and 2500, as can be seen in the figure. Also, the input rate varies according to

$$\begin{aligned} w_1 &= (25+15\cdot stp(k-1000)-5\cdot stp(k-2000) \\ &\quad +0.006\cdot rmp(k-3000)-0.012\cdot rmp(k-4000) \\ &\quad +0.006\cdot rmp(k-5000)-15\cdot stp(k-6000) \\ &\quad +25\cdot stp(k-7000)+15\cdot stp(k-7025) \\ &\quad -5\cdot stp(k-9000)). \end{aligned} \tag{18}$$

IV. A COMPLETE IMPLEMENTATION

To support the applicability of our approach we implemented two different Java-based prototypes: a web application based on Spring and a classic stand-alone application.

The web application has been developed in the Spring Framework [1]. Spring is one of the most popular application development framework for enterprise Java applications. The purpose of this implementation is to show the impact of moving existing enterprise projects toward our approach. We assume an underlying service-oriented architecture [11], that is, all the functionalities needed for the execution of the software are conveniently grouped in functional components that expose a web-service interface. An application may also invoke services offered by third parties, e.g. may use the map service from Google or Yahoo. The invocation of a

service may result in a correct execution or may lead to an exception. Collections of success and failure events can be processed through established statistical methods to online estimate the current reliability of a service [7]. If two or more implementations expose the same interface they are functionally equivalent and any of the two can be used to perform the exposed operations (e.g. we can assume that the map service can be interchangeably gathered from either Google or Yahoo). In this scenario, we assume as the driver of choice the actual reliability of equivalent alternatives, and we want to redirect the calls to either of them according to a desired reliability for the abstract operation. In Spring this can be done, for example, by exploiting *aspect-orientation* capabilities of the framework. Aspect-Oriented Programming (AOP) is a paradigm enabling an explicit separation of concerns while developing software [8]. AOP is an effective mean to implement our approach in a seamless way on existing enterprise projects, with almost no impact on the existing code but the addition of a new aspect. This allows to delegate the application of the method to a single expert, with low impact on the development cycle and project management too. At present, in Spring we implemented the autotunable 2-level case only, a screenshot of the running implementation can be seen in Figure 6.

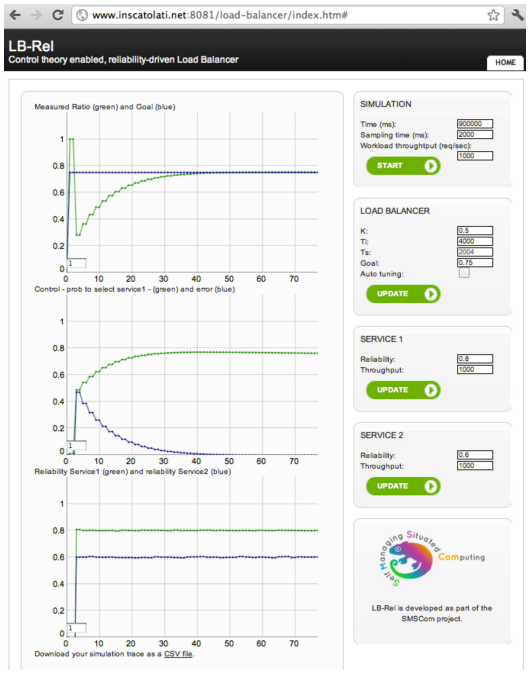


Fig. 6: Screenshot of the Spring implementation.

A stand-alone Java application has instead been developed with Maven [2], a state-of-the-art project management and comprehension tool developed by the Apache foundation. Maven is based on the concept of *project object model* and provides support for easy distribution and integration of the developed artifacts. This is the fully-fledged implementation of the concepts in this paper, supporting n-level autotuning too. With this implementation we aim at providing a proof of concept of the complexity of both coding and executing the controllers in a general purpose environment. The same Java code can be adapted to extend the Spring implementation too. Both the implementations can be downloaded from [6], as well as the Matlab scripts to reproduce our data.

V. Conclusions and Future Work

This paper dealt with the problem of dynamic binding in the context of software services composition, to satisfy reliability requirements. The problem was formulated as a discrete-time feedback control one, and autotuning techniques were devised and applied, both in the two-level and the n-level case, to guarantee the desired reliability without the need for computationally-intensive optimisations that could degrade the performance of the software system. The proposed control strategy was validated through Matlab simulations and subsequently implemented in Java within the Spring framework, and as a standalone application. Experimental results confirm the validity of the control solution in matching the reliability requirements through dynamic binding. Future work will address different performance/quality metrics, and the implementation of the designed solutions in distributed environments hosting mutually related software components. This will most likely lead to address the problem by means of multivariable control methodologies, thereby extending the research beyond the composition of single-variable elements.

Acknowledgment

This research has been partially funded by the European Commission, Programme IDEAS-ERC, Project 227977-SMScom. This work was supported by the Swedish Research Council through the LCCC Linnaeus Center.

References

[1] `http://www.springsource.org`.
[2] `http://maven.apache.org`.
[3] Karl Johan Åström and Tore Hägglund. *Advanced PID control*. ISA - the Instrumentation, Systems, and Automation Society, Research Triangle Park, NY, 2006.
[4] L. Baresi, E. Di Nitto, and C. Ghezzi. Toward open-world software: Issue and challenges. *Computer*, 39(10):36 –43, oct 2006.
[5] Anonio Filieri, Carlo Ghezzi, Alberto Leva, and Martina Maggio. Reliability-driven dynamic binding via feedback control. In *Proc. SEAMS 2012*, page to appear, 2012.
[6] Antonio Filieri, Carlo Ghezzi, Alberto Leva, and Martina Maggio. `http://filieri.dei.polimi.it/publications/2012-cdc`.
[7] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. A formal approach to adaptive software: continuous assurance of non-functional requirements. *Formal Aspects of Computing*, pages 1–24, 2011.
[8] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Aksit and Satoshi Matsuoka, editors, *ECOOP'97 Object-Oriented Programming*, volume 1241 of *LNCS*, pages 220–242. Springer Berlin / Heidelberg, 1997.
[9] Alberto Leva, Sara Negro, and Alessandro V. Papadopoulos. PI/PID autotuning with contextual model parametrisation. *Journal of Process Control*, 20(4):452–463, 2010.
[10] Messaoud and Benidir. On the root distribution of general polynomials with respect to the unit circle. *Signal Processing*, 53(1):75 – 82, 1996.
[11] R. Perrey and M. Lycett. Service-oriented architecture. In *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on*, pages 116 – 119, jan. 2003.