Reliability-Driven Dynamic Binding via Feedback Control

Antonio Filieri, Carlo Ghezzi, Alberto Leva
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza L. da Vinci, 32; 20133 Milano, Italy
{*filieri, ghezzi, leva*}*@elet.polimi.it*

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

***Abstract*—We are concerned with software that can self-adapt to satisfy certain reliability requirements, in spite of adverse changes affecting the environment in which it is embedded. Self-adapting software architectures are heavily based on dynamic binding. The bindings among components are dynamically set as the conditions that require a self-adaptation are discovered during the system's lifetime. By adopting a suitable modeling approach, the dynamic binding problem can be formulated as a discrete-time feedback control problem, and solved with very simple techniques based on linear blocks. Doing so, reliability objectives are in turn formulated as set point tracking ones in the presence of disturbances, and attained without the need for optimization. At design time, the proposed formulation has the advantage of naturally providing system sizing clues, while at operation time, the inherent computational simplicity of the obtained controllers results in a low overhead. Finally, the formulation allows for a rigorous assessment of the achieved results in both nominal and off-design conditions for any desired operation point.**

***Keywords*-Self-adaptive software; reliability requirements; dynamic binding; discrete-time feedback control**

I. Introduction

Modern software systems live in highly dynamic environments and must survive changes while they are operational. Changes may occur because the requirements they should satisfy evolve over time. They may also occur because the environment in which the system is embedded changes and the environment assumptions made when the system was originally defined—and upon which design decisions affecting the implementation were made—are not valid any more, and lead to requirements violations. In these cases, self-adaptation becomes a key goal for the implementation. If the system can self-adapt at run-time to achieve continuous requirements satisfaction, it can run continuously and continuously provide service. This is, in turn, a requirement that many modern systems must satisfy.

From a very abstract viewpoint, self-organization at the software architecture level must leverage dynamic binding among the components of the architecture. Dynamic binding is the enabling feature that supports dynamic configurations. In this paper, we explore how continuous reconfigurations through dynamic binding can be obtained as the solution of a discrete-time feedback control problem. The software system may be viewed as a broken down into constituent blocks having no hard-wired connectors. Connectors result dynamically when the binding between two blocks is established. Our final goal is to apply control theory to automatically derive how the bindings must evolve over time to achieve self-adaptation. The changes that may lead to new bindings are treated as *disturbances* in control theory terminology.

The aforementioned abstract viewpoint is specialized in this paper to the context of requirements that specify the expected *quality of service*. (QoS). Even more specifically, we will focus on the *reliability* requirements that the system as a whole must satisfy. Reliability is here broadly defined as the probability of successfully accomplishing an assigned task when it is requested. The meaning of *success* is domain dependent. We give to the term a pretty wide scope, which can be summarized by stating that the execution of the task satisfies convenient properties: e.g., it has been completed without exceptions, within an acceptable time-out, occupying less than a certain amount of memory, etc. We also position this work in a setting where the system we build is a composition of parts that are quite autonomous, possibly developed and managed by independent entities, as in the case of service-oriented systems. The independent parts are characterized by their own reliability properties, which concur to the reliability of the composition defined by the established bindings in place at any given time.

To address the dynamic binding problem in its full generality, as it was stated above, we need to narrow it down first to its most elementary setting. This is exactly the purpose of this paper. The most elementary problem frame we need to study is what we call the *two-alternatives dynamic binding problem*. The problem can be formulated as follows: We need to provide a certain service S, and two components $C1$ and $C2$ are available which provide that service, each exhibiting a varying QoS. Their reliability may for example change over time because of changes in the load conditions of the host on which they run. Service S must satisfy certain reliability requirements R for its clients. The solution is to dynamically distribute the requests for service S among $C1$ and $C2$ (i.e., to follow a certain dynamic binding law) in such a way that we satisfy the requirements R. If this is not feasible, the solution ensures that we get as close as possible to R. Once this elementary problem frame is solved in the

control theory framework, we show how we can generalize it to the case in which the choice is among any number of components, $C1$, $C2$, ... CN. This binding problem is the basis upon which the mode general setting can be addressed.

The paper is organized as follows. Section II starts by formulating the addressed problem as a discrete-time dynamic control one, detailed as set point tracking in the presence of disturbances. A suitable feedback controller structure is then devised, attaining the desired objectives and allowing for the assessment of local stability and robustness in the vicinity of a generic operation point. An automatic procedure is finally devised to determine the controller parameters on line, once its structure is dictated by the performed (off line) analysis, and the used dynamic models are employed to validate the control by extensive simulation. A two-alternatives case is used in section II for simplicity, while section III extends the obtained results to the n-alternatives case. Section IV discusses implementation issues concerning the application to service-oriented and Java-based applications. Section V presents related work and Section VI concludes the paper.

II. Two-Alternatives Online Dynamic Binding

Dynamic binding is the native elementary mechanism that enables dynamic architectural reconfigurations, through which one can achieve the required QoS levels by adapting the application to changing and variable user behaviors and environmental conditions. This section explains how dynamic binding can be treated as a discrete-time feedback control problem, by going through the typical control synthesis approach. In our initial formalization, the problem is reduced to the dynamic decision of directing requests to one out of two possible servers (*two-alternatives dynamic binding*), with the goal of optimizing the satisfaction of reliability requirements. To achieve this goal, first, a (possibly simplified) dynamic model of the uncontrolled system is written and described in Section II-A. Subsequently, a regulator is designed to fulfill the required goals, as shown in Section II-B. The analytical formulation of the controller allows rigorous convergence analysis to be performed on the closed-loop system. The avoidance of oscillations, biases and unnecessary extra quality – that would be costly – is a result of this modeling and synthesis process.

A. The Modeling Paradigm

The case study used to present the method is shown in Figure 1, where requests enter the system through the initial node n_i, at rate w_i, and are then re-routed to different nodes to be served – in this basic case just two service nodes are presented, marked with s_1 and s_2. Each service node s_j has a success probability p_{sj}, thus a failure one $p_{fj} = 1 - p_{sj}$. The control objective is to continuously adapt the probability p_1 of routing to s_1, thus also the probability $p_2 = 1 - p_1$ of routing to s_2, to match or overcome an overall reliability goal. Nodes n_f and n_s respectively represent the failure and the success state.

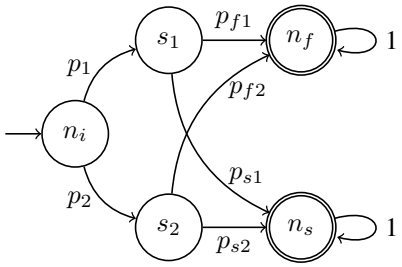


Figure 1. Model representation for the basic load balancing example.

The controller is supposed to act periodically, at a fixed time step (or *sampling period*) T_s chosen as the period on which reliability needs to be quantified. Once T_s is selected based on the particular problem at hand, the treatment can be performed entirely in the discrete-time domain, that is, introducing a time index k that counts the controller interventions, and interpreting any quantity $x(k)$ "at (discrete) time k", whatever x is, as the value of x in the (continuous) time span from kT_s to $(k+1)T_s$, when a new value will become available.

Furthermore, each node j is supposed to encompass a request queue. In system-theoretical terms each queue is called a *storage*, and the values of all storages at time k (i.e., the number ν_j of requests in each queue j[1]) form the controlled system's *state vector* $\mathbf{n}(k)$. Also, any quantity that is exogenous for the controlled system is termed an *input*. Inputs can either be a *control variable* if the controller can prescribe it – in the example, the manipulated probability $p_1(k)$, or a *disturbance* if the controller can possibly measure it, but not prescribe it – in the example, the input rate $w_i(k)$. Finally, any other quantity pertaining to the system – in the example, p_{s1} and p_{s2} – is referred to as a *parameter*. Parameters can be constant over time and known, leading to a *time-invariant system without uncertainty*. Also, they can be constant over time but only known as nominal values, providing a *time-invariant system with uncertainty*. Also, parameters may vary over time, resulting in a *time-varying system*. In the case of uncertain or varying parameters, there may or may not be the possibility of *estimating* their current values on-line, based on the available measurements.

For the considered case study, we suppose that p_{s1} and p_{s2} are "moderately varying with sporadic steps", i.e., that their value undergoes, in each control step of duration T_s, only small variations around a nominal value, while from time to time – but sporadically with respect to the control steps – there may be a single, large and abrupt variation. This models for example the case of a node failure, whenever

[1]Notice that only the length of the queue matters in our model; i.e. the possible parameters carried as part of the requests can be ignored.

a service node may not be available for some time due to external and a priori unpredictable causes. We also suppose that p_{s1} and p_{s2} can be estimated by observing the service nodes' success and failure rates. As a last hypothesis, each node j is supposed to have a maximum throughput of t_{mj} requests per control period of length T_s. Given these assumptions, we can now write models for the system.

1) The Controlled System's Model: To obtain a control-theoretical model of the system to be controlled, one first needs to write its *state equations*, i.e., to express the state at time k as a function of the state and the inputs at time $k-1$. In a previous work this was done starting from Discrete Time Markov Chain (DTMC) models [1], while here the formalism is extended to consider the queuing mechanism induced by the throughput saturation. Denoting by m the number of nodes in the chain – in the example $m = 5$ - the state equations are

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

where bold letters denote vectors. Each element of $\mathbf{w}$ represents the number of requests entering the corresponding node in the control step: in our case study there is only one entry point, thus $\mathbf{w} = [w_i\, 0\, 0\, 0\, 0]'$, but the model already takes into account the possibility of having multiple ones (it would suffice that $\mathbf{w}$ had more than one nonzero element). Vector $\mathbf{n} = [\nu_i\, \nu_1\, \nu_2\, \nu_s\, \nu_f]'$ is the state, while $\mathbf{r}$ represents the number of requests actually served by each node at time k, which is the minimum between the number of enqueued ones and $\mathbf{t_m}$, the vector of maximum node throughputs. Each node is supposed to have a possibly different maximum throughput, taking into account the differences in the implementations and capacity of each component of the chain. Finally, $\mathcal{P}$ is the transition matrix of the chain, that for the case study of Figure 1 takes the form

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

where for the moment p_{s1} and p_{s2} are supposed constant, although this assumption will be relaxed. In other words, default values could be assumed for the probability of success and failure of the service nodes, for example based on Service Level Agreements.

The next modeling step is to write the *output equation*, instantaneously relating the quantities needed to produce the metric(s) of interest – here, reliability – to the system state (and possibly, which does not happen here, its input). Said quantities – the system *output* in control-theoretical terms – are ν_f and ν_s, thus defining $\mathbf{y} = [\nu_f\, \nu_s]'$, the output equation is

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

This makes (1,3) a *time-invariant nonlinear model* – $\mathcal{M}$ from now on – owing to the rate saturation and to an input-by-state product – the term $\mathcal{P}\cdot\mathbf{r}$ in (1) – that causes the state equation not to be linear in both the state and the input vectors (while the output equation apparently is).

Finally comes the *measurement dynamics*, i.e., the output-to-metric relationship cascaded to the system model. Assuming – quite naturally – that the measured reliability q be expressed as

$$q(k) = \frac{\nu_s(k) - \nu_s(k-1)}{\nu_s(k) + n_f(k) - \nu_s(k-1) - \nu_f(k-1)}, \qquad (4)$$

which means that the measured reliability is the percentage of successful requests in the last time interval, the measurement dynamics is described by a system with input $\mathbf{u}_m$ and state $\mathbf{x}_m$ both given by $\mathbf{y}$, the state equation

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

and the output equation (this time containing the input) given by (4). Also model (4,5) – $\mathcal{M}_m$ from now on – is nonlinear, owing in this case to the output equation only.

2) The Linearized Model: Since the required reliability is assumed to vary sporadically, the problem is naturally cast in the framework called *control in the vicinity of an equilibrium*, indicating that the system needs to be brought to the equilibrium first and subsequently kept in its proximity. In this case it is possible to first analyze the equilibria of the system for constant inputs, and then obtain a linear model for it valid in the vicinity of the generic equilibrium. Finally a controller suitable for any equilibrium is devised. More precisely, we write a different controller *parametrization* for any equilibrium in the vicinity of which the system needs maintaining.

Observing that $\mathcal{M}$ and $\mathcal{M}_m$ are cascaded, it is convenient to treat them separately, and then join the results. Starting with $\mathcal{M}$, equation (1) can be written in the form

$$\mathbf{n}(k) = \mathbf{\Phi}\left(\mathbf{n}(k-1), \mathbf{u}(k-1), \mathbf{d}(k-1)\right) \qquad (6)$$

where $\mathbf{u} = p_1$ is the input and $\mathbf{d} = w_i$ the disturbance. Assuming to receive constant inputs $\overline{\mathbf{u}}$ and $\overline{\mathbf{d}}$, the corresponding equilibrium states $\overline{\mathbf{n}}$ are the solutions of

$$\overline{\mathbf{n}} = \mathbf{\Phi}\left(\overline{\mathbf{n}}, \overline{\mathbf{u}}, \overline{\mathbf{d}}\right) \qquad (7)$$

that, specialized to our case, becomes

$$(\mathcal{P}(\overline{\mathbf{u}}) - \mathbf{I})\min\{\mathbf{t_m}, \overline{\mathbf{n}}\} + [w_i\, 0 \cdots 0]' = \mathbf{0} \qquad (8)$$

where $\mathbf{I}$ represents the identity matrix.

Matrix $\mathcal{P}(\overline{\mathbf{u}}) - \mathbf{I}$ is structurally singular, and it can be verified that no equilibrium exists. This is correct, as the output

nodes in Figure 1 apparently accumulate (served) requests indefinitely. If however one repeats the equilibrium search neglecting the output nodes accumulation, i.e., replacing *for this modeling purpose* $\mathcal{P}$ with the reduced matrix

$$\mathcal{P}_{red} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ p_1 & 0 & 0 & 0 & 0 \\ 1-p_1 & 0 & 0 & 0 & 0 \\ 0 & 1-p_{s1} & 1-p_{s2} & 0 & 0 \\ 0 & p_{s1} & p_{s2} & 0 & 0 \end{bmatrix}, \quad (9)$$

then an equilibrium is always found as

$$\begin{aligned} \overline{\mathbf{n}} &= [\overline{\nu}_i\, \overline{\nu}_1\, \overline{\nu}_2\, \overline{\nu}_s\, \overline{\nu}_f]' = (I - P_{red}(\overline{\mathbf{u}}))^{-1}\, \overline{\mathbf{d}} \\ &= \begin{bmatrix} 1 \\ p_1 \\ 1-p_1 \\ p_1(1-p_{s1}) + (1-p_1)(1-p_{s2}) \\ p_1 p_{s1} + (1-p_1) p_{s2} \end{bmatrix} \overline{w}_i \end{aligned} \quad (10)$$

if this satisfies the maximum throughput constraints, i.e., if $\mathbf{r} = \mathbf{n}$. In the opposite case there is clearly no equilibrium, as one or more queues will grow indefinitely if a non-feasible number of requests are injected into the system. Notice that the equilibrium (10) is valid also for the original system, under the same feasibility hypothesis, by just interpreting ν_s and ν_f as the successful and failed requests in the last period, thus (re-)defining the reliability in the same period as their ratio by replacing (4) with

$$q(k) = \frac{\nu_s(k)}{\nu_s(k) + \nu_f(k)} \quad (11)$$

Defining $\delta\mathbf{n} = \mathbf{n} - \overline{\mathbf{n}}$, $\delta\mathbf{u} = \mathbf{u} - \overline{\mathbf{u}}$, $\delta\mathbf{d} = \mathbf{d} - \overline{\mathbf{d}}$ and $\delta\mathbf{y} = \mathbf{y} - \overline{\mathbf{y}}$, the linearized model of the system (with matrix $\mathcal{P}_{red}$ in accordance with the interpretation above) is

$$\begin{cases} \delta\mathbf{n}(k) &= \mathbf{A}\delta\mathbf{n}(k-1) + \\ & \quad \mathbf{B}_u\delta\mathbf{u}(k-1) + \mathbf{B}_d\delta\mathbf{d}(k-1) \\ \delta\mathbf{y}(k) &= \mathbf{C}\delta\mathbf{n}(k) \end{cases} \quad (12)$$

where

$$\mathbf{A} = \tfrac{\partial \mathbf{\Phi}_{red}}{\partial \mathbf{n}}\big|_{\overline{\mathbf{n}},\overline{\mathbf{u}},\overline{\mathbf{d}}}, \quad \mathbf{B}_u = \tfrac{\partial \mathbf{\Phi}_{\mathbf{red}}}{\partial \mathbf{u}}\big|_{\overline{\mathbf{n}},\overline{\mathbf{u}},\overline{\mathbf{d}}}, \quad \mathbf{B}_d = \tfrac{\partial \mathbf{\Phi}_{\mathbf{red}}}{\partial \mathbf{d}}\big|_{\overline{\mathbf{n}},\overline{\mathbf{u}},\overline{\mathbf{d}}} \quad (13)$$

are respectively the Jacobian matrices of $\mathbf{\Phi}_{red}$ (same expression as $\mathbf{\Phi}$ with $\mathcal{P}$ replaced by $\mathcal{P}_{red}$) with respect to $\mathbf{n}$, $\mathbf{u}$ and $\mathbf{d}$, computed at the equilibrium, and $\mathbf{C}$ is defined by equation (3). Matrix $\mathbf{A}$ simply (and expectedly) equals $\mathcal{P}$, while

$$\begin{aligned} \mathbf{B}_u &= \begin{bmatrix} 0 & \overline{\nu}_1 & -\overline{\nu}_1 & 0 & 0 \end{bmatrix}', \\ \mathbf{B}_d &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}'. \end{aligned} \quad (14)$$

Coming to $\mathcal{M}_m$, the re-definition of q given by (11) makes it merely algebraic, any constant input $\overline{\mathbf{u}}_m$ yielding an equilibrium output $\overline{q} = \overline{\nu}_s/(\overline{\nu}_s + \overline{\nu}_f)$. Then, following the same procedure used for the output equation of $\mathcal{M}$, the linearized (algebraic) model of $\mathcal{M}_m$ is

$$\delta q(k) = \mathbf{D}_m \delta\mathbf{u}_m(k) \quad (15)$$

where $\delta\mathbf{u}_m = \mathbf{u}_m - \overline{\mathbf{u}}_m$, $\delta q = q - \overline{q}$, and

$$\mathbf{D}_m = \begin{bmatrix} \frac{\overline{\nu}_f}{(\overline{\nu}_s+\overline{\nu}_f)^2} & -\frac{\overline{\nu}_s}{(\overline{\nu}_s+\overline{\nu}_f)^2} \end{bmatrix}. \quad (16)$$

Putting it all together, the complete linearized model is then

$$\begin{cases} \delta\mathbf{n}(k) &= \mathbf{A}\delta\mathbf{n}(k-1) \\ & \quad +\mathbf{B}_u\delta\mathbf{u}(k-1) + \mathbf{B}_d\delta\mathbf{d}(k-1) \\ \delta q(k) &= \mathbf{C}_m\delta\mathbf{n}(k) \end{cases} \quad (17)$$

where

$$\begin{aligned} \mathbf{C}_m &= \mathbf{D}_m\mathbf{C} \\ &= \begin{bmatrix} 0 & 0 & 0 & \frac{\overline{\nu}_f}{(\overline{\nu}_s+\overline{\nu}_f)^2} & -\frac{\overline{\nu}_s}{(\overline{\nu}_s+\overline{\nu}_f)^2} \end{bmatrix}. \end{aligned} \quad (18)$$

Based on (17), the z-domain transfer function from δp_1 to δq is readily computed as

$$\begin{aligned} P(z) &= \mathbf{C}_m\,(z\mathbf{I} - \mathbf{A})^{-1}\,\mathbf{B}_u \\ &= \overline{\nu}_1\,\frac{\overline{\nu}_s(p_{s2} - p_{s1}) - \overline{\nu}_f(p_{f2} - p_{f1})}{(\overline{\nu}_s + \overline{\nu}_f)^2}\,\frac{1}{z^2} \end{aligned} \quad (19)$$

that bringing in all the relationships among the involved quantities, simply reduces to

$$P(z) = \frac{p_{s2} - p_{s1}}{z^2} \quad (20)$$

and reveals some control-relevant facts. First, the gain is the difference of the service nodes' success probabilities, thus (correctly) zero if they are equal, since in that case no routing action can alter the overall reliability. Second, and most important, the structure of the controlled dynamics is invariantly that of a two-steps delay, allowing for a simple control law as that employed in the following. On the other hand, since the *sign* of the controlled system's gain can change, most likely no single controller parametrization will be suitable for all situations, and an on-line estimation of the service nodes success probabilities is required. Notice however, that in general, the only estimation needed is the sign of the mentioned difference.

B. Control Synthesis

The requirement of reaching at least a reliability level of $\bar{q}$ in $[0, 1]$ at time k can be expressed as

$$q(k) \geq \bar{q} \quad (21)$$

and for the example of Figure 1 the number ν_s of successfully served requests and the number ν_f of failures experienced are represented by elements of the vector $\mathbf{n}(k)$. The simplest way to attain such a goal in a control-theoretical manner is to design a feedback controller that at each step k computes the control signal – in the example, $p_1(k)$ – based on the desired reliability $\bar{q}(k)$ – typically set slightly above

the desired threshold to accommodate for the unavoidable small fluctuations at operation time – and its measured value $q(k)$. Based on equation (20) and an analysis too long to report here, it can be concluded that zero steady-state error and a high degree of stability can be achieved by the PI (Proportional plus Integral) controller

$$\begin{array}{rcl} u_i(k) & = & u_i(k-1) + a(1-b)e(k-1) \\ p_1(k) & = & u_i(k) + ae(k) \end{array} \tag{22}$$

where $e(k) = \bar{q}(k) - q(k)$ is the error. Notice that u is in this case the control signal, therefore the probability of routing to a specific service. The reader interested in additional information on PI(D) control can refer e.g. to [2] and the vast bibliography provided therein.

Coming back to the addressed problem, the Jury criterion [3] reveals that once a and b are chosen, asymptotic stability of the closed-loop system composed of (20) and (22) holds for any value d of the difference $p_{s2} - p_{s1}$, obviously in the range $(-1, 1)$, such that

$$\begin{cases} \dfrac{1-abd}{1+abd} & > \quad 0 \\ \dfrac{(a^2b^2d^2 - abd + ad - 1)(a^2b^2d^2 + abd - ad - 1)}{(1-abd1)(1+abd)} & > \quad 0 \\ \dfrac{ad(1-b)(abd + ad + 2)(a^2b^2d^2 - abd + ad - 1)}{a^2b^2d^2 + abd - ad - 1} & > \quad 0 \end{cases} \tag{23}$$

Studying (23) 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. Of course, this is not true for control *performance*: for example, the time required to recover from a disturbance can degrade significantly if the estimation of d is not good enough.

C. Auto-Tuning

One could relate the previously defined a and b to d and some performance specification. However, this does not seem a user-friendly approach. Therefore, to facilitate usability, an *auto-tuning* mechanism was introduced, as shown in the block diagram of Figure 2. The purpose of the auto-tuning mechanism is to automatically update the controller parameters to the current conditions. This means that if the reliabilities of the different services radically change, the parameters of the controller need to be tuned accordingly. In fact, as previously stated, the sign of the differences between p_{s2} and p_{s1}, i.e., the sign of d, is crucial for the control procedure.

To this end, a procedure is introduced to select the PI parameters. The discrete-time transfer function (20) is re-interpreted as continuous-time and sampled at period T_s, yielding:

$$P_c(z) = d\, e^{-2T_s\, s}, \quad R_c(s) = K\left(1 + \frac{1}{sT_i}\right) \tag{24}$$

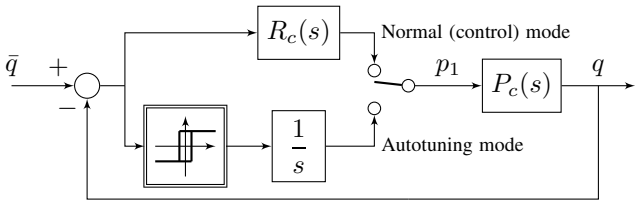


Figure 2. Basic scheme for relay-based (PI) auto-tuning.

where s is the Laplace transform complex variable, d is the difference defined above, and

$$a = K\left(1 + \frac{T_s}{T_i}\right), \quad b = \frac{T_i}{T_i + T_s}. \tag{25}$$

The applied methodology is based on relay feedback, see e.g. [4] for background material. More in detail, by replacing the feedback controller with a relay cascaded to an integrator, the point of the frequency response $P_c(j\omega)$ – where j is the imaginary unit and ω the frequency – with phase -90° is easily found from the characteristics (frequency and amplitude) of the sustained oscillation induced on the controlled variable. This technique, commonly referred to as *relay feedback identification*, is known to provide useful auto-tuning information rapidly and with a very modest system upset. Once the mentioned frequency response point is determined, taking as control specification a desired phase margin $\overline{\varphi}_m$ (in degrees), the parameters of R_c in (24) – thus a and b via (25) – are obtained by solving the complex equation

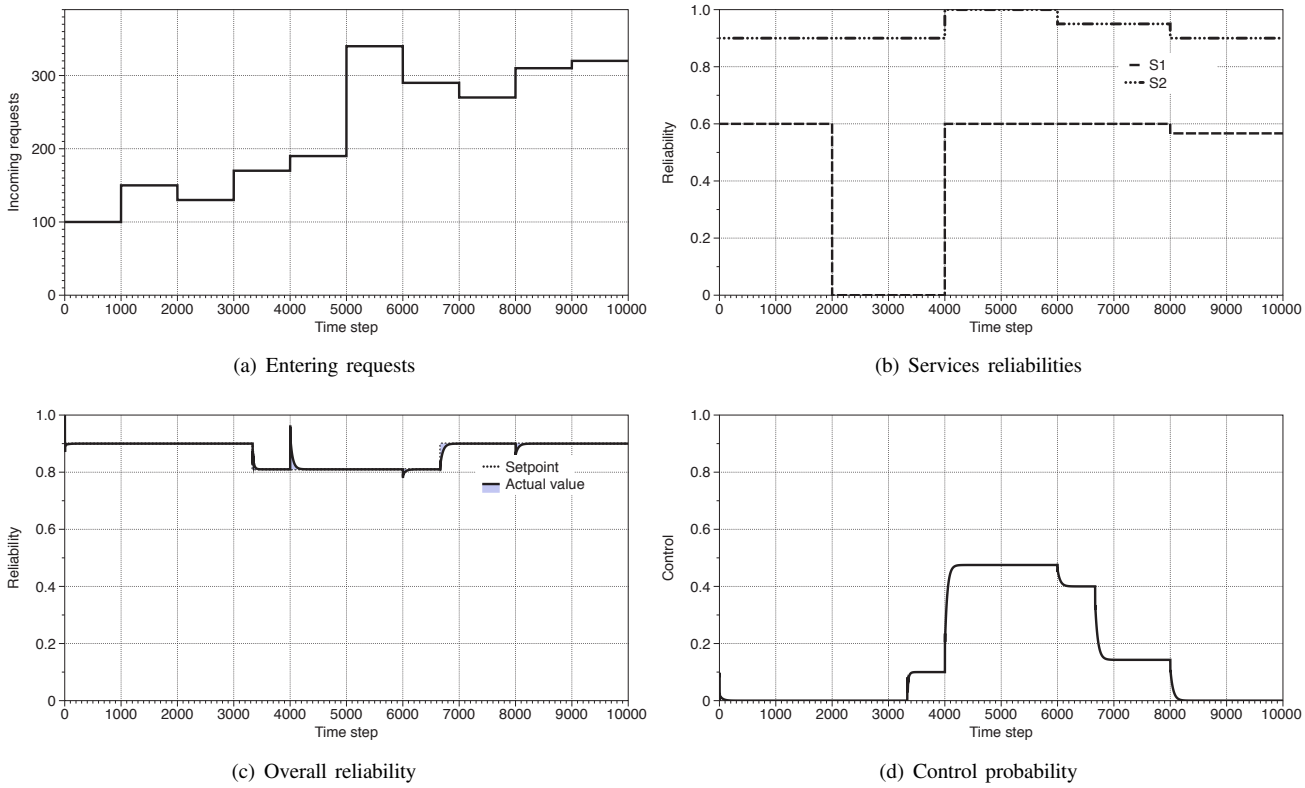
$$R_c(j\overline{\omega}) \cdot \overline{P}_\omega\, e^{-j90^\circ} = e^{j(180^\circ - \overline{\varphi}_m)} \tag{26}$$

where $\overline{\omega}$ is the oscillation frequency, and $\overline{P}_\omega$ the correspondingly measured frequency response magnitude, see e.g. [5] for details that would stray from the scope of this work. Suffice to say that parameter $\overline{\varphi}_m$ is of course positive and less than 90°, that lower values privilege response speed versus absence of oscillations and degree of stability, that higher values do the reverse, and that in virtually any case 60° is a reasonable default value.

For the convenience of non-specialist users, one could then provide a "desire knob" graduated from 0 to 1, 0 corresponding to the request of minimum time for both the response to desired reliability variations and the rejection of disturbances at the possible cost of oscillatory transients and diminished stability degree, while 1 calls for maximum stability and transients' smoothness, at the potential cost of response time, and have the required phase margin vary in accordance with the user choice, say from 40° to 80°.

D. Control Validation

A simulation campaign was conducted, prior to the implementation of the control policy in a real software system.



(a) Entering requests

(b) Services reliabilities

(c) Overall reliability

(d) Control probability

Figure 3. Simulation of two-alternatives selector.

The results of one of the simulations from the campaign are reported in Figure 3. The MATLAB simulator is started asking for 10000 simulation steps, each node can serve maximum 100 of requests per step and the system is supposed to maintain a reliability of 0.9. The initial failure probability of the first service is 0.4 while the one for the second service is 0.1.

A few variations and disturbances are injected into the system. First, to see how the controller reacts to changes in the set point, the simulations step are divided into three different parts and the requested reliability is diminished to 0.8 in the second part of the simulation. Second, the simulation is divided into five different parts, in each of these parts the success and failure probabilities of the services were changed, for example simulating the complete failure of the first service between time units 2000 and 4000, as can be seen in Figure 3(b); the figure also shows the complete pattern. Third, the number of requests entering the system is changed according to a predefined pattern, to see the reaction of the load balancer to different loads. As can be seen in Figure 3(c), at time unit 4000, i.e., when the first service node is back to its normal operations, there is a spike in reliability that is immediately compensated by the control action shown in Figure 3(d).

The MATLAB implementation which has been used to perform the experiments will be discussed later in Section IV.

III. N-Alternatives Online Dynamic Binding

Thanks to its inherent modularity, the control approach presented in the previous section for the two-alternatives case can be extended to the n alternatives in a natural manner. In 1976 D. Knuth proved that every multinomial distribution can be equivalently reproduced by conveniently combining binary probabilistic choices [6]. In a similar fashion, to build a n-alternatives selector, it suffices to apply the scheme of Section II-B hierarchically in order to build a binary selection tree whose leaves are the n components that can be targets of the binding and internal nodes are two-alternatives selectors. This leads to a structure like the one shown in Figure 4, composed of controllers like the one we devised for the two-alternatives case. Notice that the "intermediate" nodes we generated can be considered as fictitious, since they are used only to compute the probabilities of routing from the input node to the possible targets.

Combining PI controllers, therefore creating a hierarchical control structure, requires careful setting of the parameters for the different control elements. To understand the issue, assume the structure of Figure 4 is implemented and both arrows exiting from C_1 are connected to concrete executors,

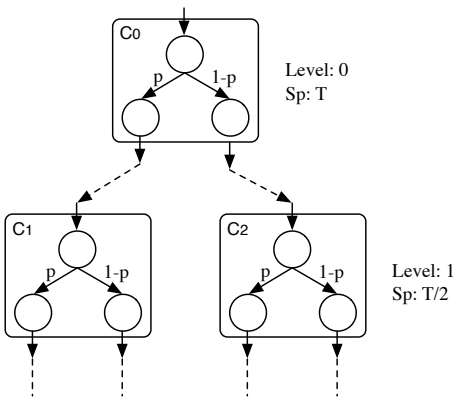


Figure 4. N-alternatives binding structure.

S_1 and S_2. These executors have their own reliabilities, respectively r_1 and r_2. The reliability r_{C_1} provided by C_1 can be computed as $p_1 \cdot r_1 + (1 - p_1) \cdot r_2$, where p_1 is the value produced by the controller. When r_{C_1} does not meet the reliability requirement, the controller C_1 can only adjust the value of p_1 in order to get as close as possible to the target. This adjustment typically takes a few time steps to be completed, depending on the configuration of the controller (i.e., the values of K and T_i).

Suppose now that the system is running and satisfies the overall reliability target. Consider a scenario where r_1 decreases sharply, for example due to a complete failure of the service S_1. Assuming that all two-alternatives controllers adopt the same time-step to query their siblings, the violation of the requirement due to S_1's failure is propagated upwards from C_1 and therefore it is perceived by both C_1 and C_0 instantaneously triggering a reaction. Simultaneous changes in the decision of C_0 and C_1 could interfere with one another, delaying the solution and possibly introducing oscillations in the global reliability of the system. A better solution for the problem is to allow the controller which is closer to the source of the violation to react first. In this case, if possible, C_1 would compensate the failure of S_1 by redirecting the load to S_2. If the compensation is not possible, it would still provide the best guarantees that could possibly been obtained at that level in the tree. Only at this point, if still needed, the intervention of the higher level controller C_0 should be triggered.

This scenario naturally generalizes to more complex selection trees and can be solved applying a *multirate* control strategy. The term means that for each level in the hierarchy, the corresponding controllers act with different time constants, i.e., at different rates. Precisely, higher-level nodes in the routing tree would intervene slower with respect to lower ones and their control period would just need to be changed accordingly, intuitively being larger. To simplify the design of n-alternatives selectors, the PI controllers can still share the parameters, introducing a further "scaling" factor, identified with the integer parameter r_{T_s}. This means that each tree level exerts its control action every r_{T_s} steps with respect to the lower one. r_{T_s} can be interpreted as the number of time steps required by a controller node to stabilize the control signal and therefore the reliability of the correspondent part of the tree. Multirate systems are a very well established research domain in control, and powerful analysis and synthesis techniques are available [7]. In this case, the use of such a strategy allows to avoid the issues generated by mutual interference of the controllers.

The response to a step variation of a five alternatives binder is shown in Figure 5, where the short convergence time required to meet the goals can be visually appreciated.

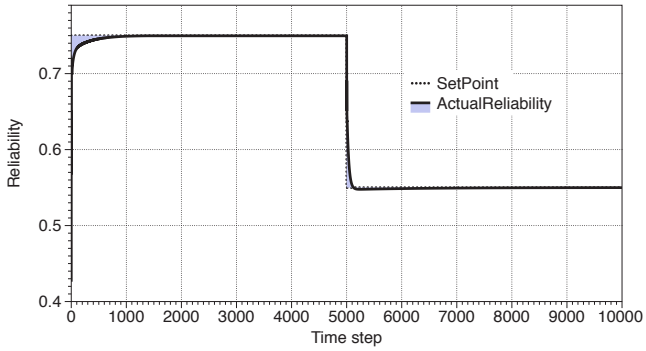


Figure 5. Step response of a 5-alternatives binder.

Concluding, we can state that the two-alternatives case scales up quite seamlessly, thanks to the adopted control-based approach. On the other hand, however, the tuning problem in the n-alternatives case is definitely more complex than in the two-alternatives case, since in the former case a multivariable and interacting system needs managing. Experiments have shown that the auto-tuning procedure devised for the two-alternatives case is not replicable *as is* in the n-alternatives context, and more advanced synthesis techniques need to be devised. The presented *preliminary* solution based on multirate control already shows that the system can work satisfactorily also with "hand made" tuning, thereby proving that the extension is practically feasible, and the tuning problem is well posed from the system-theoretical standpoint. As for now, the problem stands however open, and is being addressed with methodologies specifically aimed at multivariable control. Also, more advanced adaptation mechanisms are being studied, grounded on well established control-theoretical methods such as that proposed in [8]. The results of this ongoing research will be presented in future works.

IV. Implementation

To validate our approach and demonstrate its applicability, we implemented the control algorithm in three different

platforms. The three artifacts can be downloaded from the website http://home.dei.polimi.it/filieri/seams2012.

A Matlab implementation is available for simulation purposes[2]. Numerical mathematic programming is an established instrument for control experts to study controller's performance by simulating disturbances and process dynamics.

The second implementation is based on the Spring Framework [10]. Spring is considered one of the most complete lightweight container for J2EE applications. We exploited Aspect Oriented Programming (AOP) functionalities in order to show how a simple way our methodology can be integrated in real life applications with low impact on development organization. Monitoring and control can be defined as a specific aspect of the application, requiring no changes on the existing code. Indeed, the around advice of AOP allows one to perform custom behavior before and after invocation of an existing method, resulting in a natural environment to engraft our monitoring and control methodology. The impact on performance is definitely low thanks to the complete integration of our framework in Spring (further details in [10] or on the Spring Framework website). A running instance of the Spring implementation is also accessible from the aforementioned url, with a web-interface to ease the demonstration.

Finally, to show how to build an n-alternatives selector in Java, we implemented a simple prototype which automatically arranges the available executors in a balanced binary tree and applies to it the control laws presented in Section III. Notice that, though it is theoretically possible to obtain effective control with any binary tree, the choice of a balanced (or almost balanced) tree proved in our experiments to be easier to configure ad more efficient. Indeed the tree is set up by applying to each control node a sampling time of $r^d_{T_s}$, where $d \geq 0$ is the distance by the leaves.

The Matlab and the Spring implementations support the auto-tuning procedure sketched in Section II.

V. Related Work

Dynamic binding for Web services is emblematic of many situations in which multiple implementations for the same abstract operation are available and the actual execution of incoming requests has to be delegated to one of them. The selection criteria is usually based on cost, provided QoS, or both.

Most of the current approaches address this problem by setting a convenient optimization problem, where different qualities are traded-off, looking for an optimal, or at least satisfactory, solution [11], [12]. Earlier approaches allow the formulation of an optimization problem for each operation in order to select the best candidate with respect to a local objective function [13], [14]. Local approaches are usually efficient because in most of the practical cases the number of candidates for each single operation are not many. On the other hand, most of the QoS requirements are expressed at application level, hence shrinking the scope down to single operations my produce sub-optimal solutions with respect to the global system, or, conversely, they may overshoot producing (possibly costly) better-than-required solutions. Subsequent approaches allow for managing the optimization of a global objective, spreading over the entire design-space [15], [16]. The immediate negative effect is in terms of complexity: considering in a single optimization all the possible alternative bindings of each operation leads to a combinatorial explosion. In practice, these approaches are either unfeasible for even small cases or too complex to provide binding control. Besides the growth in the exploration space, the non linearities of the global problem may be untreatable with standard mathematical procedure and may require the adoption of soft computing techniques, such as genetic algorithms [17].

Some recent approaches combine both local and global techniques to so to improve the performance of global search by feeding in locally optimal bindings of all or part of the operation level selections (e.g. [18], [19]). Most of the optimization-based approaches consider multiple QoS metrics simultaneously. In this work we focused on reliability (with the generality provided by the domain-specific notions of *success* and *failure*), though our methodology can be adapted to control other quantitative QoS properties too, while keeping the same controller structure.

Another approach to dynamic binding has been investigated in our group in the case of service-oriented architectures [20]. The problem setting in that case, however, is quite different and refers to the case where multiple clients dynamically bind to functionally equivalent services with the goal of optimizing response time. The paper introduces and compares several predefined binding strategies, both on a theoretical and experimental grounds, and without feedback control.

We are not aware of other dynamic binding approaches based on control-theory. Our previous work [1] introduced a control-theory enabled adaptation mechanism for the control of systems modeled through Discrete Time Markov Chains. The goal of the controller was to continuously ensure satisfaction of a goal expressed as the probability to reach a desired success state from the initial one. The controller was capable of trading off reliability and costs by solving an optimization problem. The complexity of the adaptation mechanism did not depend on the size of the DTMC but only on the type of objective function and on the number of controlled variables and disturbances (i.e. transitions which values can change due to external factors that can be observed but not influenced). Such an optimization problem could be complex enough to make certain systems loose the

[2]We also implemented a Scilab version. Scilab is a widely used open-source Matlab counterpart [9].

ability to timely adapt when their requirements are violated. Thus, though [1] can be used to solve the problem of dynamic binding, this problem can be mapped to a subset of DTMCs (those having a tree-shaped connection graph) leading to the more efficient solution proposed in this paper.

The adoption of a simpler controller allows for more efficient and timely adaptations even on low-end or mobile devices. For the proposed controller is also possible a formal assessment of its effectiveness (cfr. Section II). Finally, from an architectural viewpoint, in [1] the control problem is formulated starting from a DTMC model of the entire system. Every structural change, such as the addition of a new state, would invalidate the dynamic model of the systems and consequently the one of the controller. In the approach we proposed in this paper, adding or removing a new alternative would require a very low effort because of the "boxed" hierarchical structure of the n-alternatives controller.

Concerning the application of control theory to achieve continuous QoS assurance, in the field of load-balancing, a comparison between optimization based and control-theory based techniques has been performed in [21]. Though the MIMO controllers used to balance the load among DB2 instances in [21] was more complex than a PI, the effectiveness of the feedback loop overwhelmed the optimization-based techniques, particularly in the situation of highly variable loads, where efficient continuous adjustments leaded to a smother performance curve, with reduced outliers and faster convergence time. More related applications in the field of load-balancing have to be further investigated to identify possible connections and shared controller structures.

VI. Conclusions and Future Work

In this paper, the dynamic selection of alternatives in dynamic binding problems has been addressed in its full generality, starting from its most elementary setting. First, the choice of dynamically binding a service request to one of two available alternatives has been addressed by means of control-theoretical analysis and synthesis. An autotuning procedure has been devised to automatically select the most suitable controller configuration even at run-time. Subsequently, the solution has been generalized to selecting among an arbitrary number of components.

Both a simulation environment in Matlab and a real implementation in Java, within the Spring framework and based on Maven, are discussed and extensively tested. The results of our tests allow us to conclude that the control-theoretical approach is a feasible decision mechanism for achieving a specific reliability. Additional quantitative QoS properties could also be formalized and managed within the same framework.

We are currently exploring the adoption of more complex control strategies to enhance the expressiveness of our models and the possibility of defining trade-off conditions. We firstly aim at modeling finite capacity queues and multiple types of requests, with different QoS requirements, in a unifying framework. We are also investigating the application of our dynamic binding mechanism in a fully distributed environment, with multiple selectors for a set of executors. In such setting, we aim at extending the scope of our methodology by devising a formal framework to solve the problem of parameterization of each single controller and possible stability issues.

Acknowledgment

This research has been partially funded by the European Commission, Programme IDEAS-ERC, Project 227977-SMScom.

References

[1] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio, "Self-adaptive software meets control theory: a preliminary approach supporting reliability requirements," in *ASE*, 2011, pp. 283–292.

[2] K. J. Åström and T. Hägglund, *Advanced PID control*. Research Triangle Park, NY: ISA - the Instrumentation, Systems, and Automation Society, 2006.

[3] Messaoud and Benidir, "On the root distribution of general polynomials with respect to the unit circle," *Signal Processing*, vol. 53, no. 1, pp. 75 – 82, 1996.

[4] A. Leva, "PID autotuning algorithm based on relay feedback," *IEE Proceedings-D*, vol. 140, no. 5, pp. 328–338, 1993.

[5] A. Leva, S. Negro, and A. V. Papadopoulos, "PI/PID autotuning with contextual model parametrisation," *Journal of Process Control*, vol. 20, no. 4, pp. 452–463, 2010.

[6] D. Knuth and A. Yao, *Algorithms and Complexity: New Directions and Recent Results*. Academic Press, 1976, ch. The complexity of nonuniform random number generation.

[7] M. Araki and K. Yamamoto, "Multivariable multirate sampled-data systems: State-space description, transfer characteristics, and nyquist criterion," *IEEE Transactions on Automatic Control*, vol. 31, no. 2, pp. 145 – 154, feb 1986.

[8] A. Leva, C. Maffezzoni, and R. Scattolini, "Self-tuning PID regulators for stable systems with varying delay," *Automatica*, vol. 30, no. 7, pp. 1171–1183, 1994.

[9] Scilab Consortium, *Scilab: The free software for numerical computation*, Scilab Consortium, Digiteo, Paris, France, 2011. [Online]. Available: http://www.scilab.org

[10] R. Johnson, J. Hoeller, A. Arendsen, T. Risberg, and D. Kopylenko, *Professional Java Development with the Spring Framework*. Birmingham, UK, UK: Wrox Press Ltd., 2005.

[11] D. Ardagna and R. Mirandola, "Per-flow optimal service selection for web services based processes"," *Journal of Systems and Software*, vol. 83, no. 8, pp. 1512 – 1523, 2010.

[12] N. Ben Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny, "Qos-aware service composition in dynamic service oriented environments," in *Middleware 2009*, ser. Lecture Notes in Computer Science, J. Bacon and B. Cooper, Eds. Springer Berlin / Heidelberg, 2009, vol. 5896, pp. 123–142.

[13] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for web services selection with end-to-end qos constraints," *ACM Transactions on the Web*, vol. 1, May 2007.

[14] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 369 –384, june 2007.

[15] M. Jaeger, G. Mhl, and S. Golze, "Qos-aware composition of web services: An evaluation of selection algorithms," in *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds. Springer Berlin / Heidelberg, 2005, vol. 3760, pp. 646–661.

[16] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, may 2004.

[17] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for qos-aware service composition based on genetic algorithms," in *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, ser. GECCO '05. New York, NY, USA: ACM, 2005, pp. 1069–1075.

[18] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient qos-aware service composition," in *Proceedings of the 18th international conference on World wide web*, ser. WWW '09. New York, NY, USA: ACM, 2009, pp. 881–890.

[19] Q. Liang, X. Wu, and H. Chuin Lau, "Optimizing service systems based on application-level qos," *IEEE Transactions on Services Computing*, vol. 2, no. 2, pp. 108–121, april-june 2009.

[20] C. Ghezzi, A. Motta, V. P. L. Manna, and G. Tamburrelli, "Qos driven dynamic binding in-the-many," in *QoSA*, 2010, pp. 68–83.

[21] D. Yixin, W. W. Chai, J. Hellerstein, A. Storm, M. Surenda, S. Lightstone, S. Parekh, C. Garcia-Arellano, M. Carroll, C. Lee, and J. Colaco, "Comparative studies of load balancing with control and optimization techniques," in *Proceedings of the American Control Conference*, june 2005, pp. 1484 – 1490 vol. 2.