JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering. ©JOT, 2007

Vol. 6, No. 11, Special Issue on Advances in Quality of Service Management December 2007

Model Driven Prediction and Control

Assel Akzhalova, Department of Computer Science,King's College London, Strand, London, UK, WC2R2LS, assel.akzhalova@kcl.ac.uk
Assel Altayeva, Department of Mathematics, King's College London, Strand, London, UK, WC2R2LS, assel.altayeva@kcl.ac.uk
Nurzhan Duzbayev, Department of Computer Science,King's College London, Strand, London, UK, WC2R2LS, nurzhan.duzbayev@kcl.ac.uk

Self-adaptive systems are capable of changing their behaviour at runtime to meet target constraints. An important research question is how quality of service models can inform runtime adaptation. We sketch one solution to this question by application of control theory to improve performance of queued systems by means of architectural adaptation.
Previous research by our group has shown how Auto-Regressive Integrated Moving Average techniques can be utilized to forecast how Quality of Service (QoS) characteristics are likely to evolve in the near future. This is particularly important in cases where systems can be adapted to counter QoS constraint violations. In this paper, we show how, given a similar type of QoS characteristic forecasts, strategies of architectural adaptation can be implemented that pre-emptively avoid QoS violations. The novelty of our approach is that we use classical control theory to ensure that our adaptation strategies are stable, in the sense that they do not oscillate between choices. We provide a description of how our control theoretic model can be implemented using context-based interception in .NET via model driven engineering.

1 INTRODUCTION

Self-adaptive systems are capable of changing their behaviour at runtime to meet target behavioural constraints. An important research question is how quality of service models can inform runtime adaptation. This paper presents a step towards solving this problem by application of classical control theory.

We show how Auto-Regressive Moving Average (ARMA) techniques can be use to forecast how QoS characteristics are likely to evolve in the near future. This result is useful in and of itself, as it allows us to detect warning signs that a system is tending towards violation of desired QoS levels. A warning can be given to a human administrator, who might then decide to reconfigure the system in such a way as to pre-empt the violation occurring.

However, clearly there is benefit from an automated means of controlling system parameters to provide such reconfiguration. The theory of dependability offers a range of possible solutions for automated control for load balancing and fault tolerance. In this paper, we show how, given ARMA predictions of likely future QoS characteristics, pre-emptive controllers can be developed by application of classical control theory. Our approach determines a control strategy that is optimal with respect to resource cost.

We focus on the problem of queued stability and utilization. The idea is that, if our system detects a dangerous trend towards a communication queue becoming unmanageably long, then a controller component should adapt the architecture of the system to improve influencing factors, such as service rate and the average number of calls entering the queue. For example, if the queue is becoming too long, the controller could create an alternative server and divert some calls to it, halving the service rate. In a world of limitless resources, the controller could solve the problem of QoS violation by creating a thousand such servers and, when the queue becomes shorter, the thousand servers may be replaced by one again. The problem is that each server has a cost, as does the adaptation action. The controller should adapt the architecture to provide the best QoS and yet be optimal with respect to cost.

A common approach is to perform adaptation by means of some form of a policy-based controller. Determining the optimal control strategy is difficult. A bad strategy yields a oscillating feedback problem: if the utilization improves significantly, so (to minimize cost) influencing parameters are changed again (by removing some servers for instance), and, consequently, the utilization worsens significantly, resulting in influencing parameters needing to be changed again, the utilization improving signficantly again, and so on. The best controller is one that provides an appropriate adaptation as quickly as possible but do not result in radical oscillations.

We develop the controller using root locus techniques to determine optimal non-oscillating control strategies. We also provide a description of how our control theoretic model can be implemented using context-based interception in .NET via model driven engineering.

This paper extends work developed by Altayeva, Akzhalova, Duzbayev and Poernomo some of which was previously presented at the QoSA 2006 and 2007 conferences [7].

The paper proceeds as follows:

- Section 2 summarizes relevant notions from queuing theory.
- Section 3 shows how to apply classical control theory to develop a controller providing optimal adaptation.
- Section 4 provides overview of how we employ model driven techniques to implement our control systems.
- An illustrate example is provided in section 5.
- Conclusions and related work are discussed in the final section.

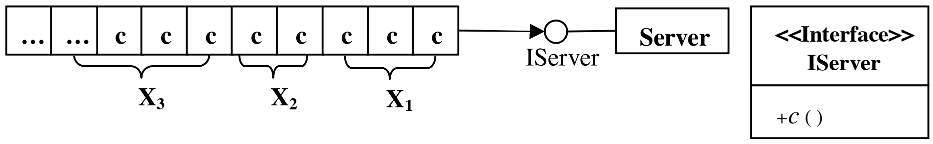

Figure 1: A queued service. Invocations of the service's methods are queued, where each X_i is the number of invocations at unit time i

2 BACKGROUND

Queued communication

Queuing theory enables the mathematical analysis of queued communication between clients and a server (or a set of servers) (see, for example, [13, 5]). Such communication is commonplace in large-scale distributed systems, where there is a dependence on loosely coupled messaging and messages can potentially be sent from different sources to the same component. Performance evaluation of such systems is essential.

We understand queued communication as depicted in Fig. 1. A number of calls enter a queue per unit time. We write $X_i \geq 0$ for the random variable denoting the number of calls entering at unit time $i = 1, 2, \ldots$. Each call is numbered and served in some order. As soon as the server finishes servicing a call, it immediately starts to serve the next call (if there are calls remaining in the queue) and the served call leaves the system. If repeated processing is necessary, a call joins the queue right from the beginning. The server is called *idle* when there are no calls remaining in the queue.

There is a range of different *models* of queued communication, each with well understood QoS benefits and disadvantages. Factors effecting the values of QoS characteristics for a particular model include the distribution of incoming requests, the servicing discipline (for example, randomly selected, incoming order or some priority discipline) and serving rate distribution.

For queued systems, a central QoS characteristic is stability. A system is defined to be *stable* when each queued call is served. It is *unstable* if there is a possibility that calls will not be served.

We compute stability via the notion of system utilization, defined as follows.

We assume each X_i is independent and equally distributed and with an average $\lambda = E[X_i]$, defining the average number of calls joining the queue per unit time. We assume that all calls have an average serving time $b > 0$ – this is the average unit time to process a single request. We consequently define the average service rate, the average number of requests that are served per unit time, to be $\mu = 1/b$.

Definition 2.1 (System utilization) *The utilization of a system, ρ is defined*

$$\rho = \lambda/\mu = \lambda b$$

where λ is average number of calls joining the queue per unit time and b is the average serving time. The system is stable *if, and only if, $\rho < 1$ and* unstable *if, and only if, $\rho > 1$. In case when $\rho = 1$, the system is stable only when $X_i = X_j$ for all i, j.*

Thus, if system works stable we have a finite queue length.

Given utilization, it is possible to compute a number of quality of service characteristics that can help in evaluating the efficiency of a M/1/1 system.

Theorem 2.1 (QoS Characteristics) *The following statements are true of a M/1/1 queued system:*

- *The probability of n requests being in a queue is*

  $$P(n) = \rho^n P(0)$$

  where the probability of 0 calls in the system is $P(0) = 1 - \rho$.

- *The average number of requests in the system is*

  $$E[N] = \sum_{n=0}^{\infty} nP(n) = \rho/1 - \rho$$

- *The average waiting time $E[W_i]$ for a request is*

  $$E[W_i] = bE[N]$$

The proof can be found in, for example, [8].

It follows from the theorem that the total average time for a call spent in the system is $E[W] = E[W_i] + b$.

Forecasting queue QoS

Given a system in which there are no trends in the number of calls made to a server or in the service time for a call, then the definition of ρ above provides the best means of predicting stability and QoS dependent characteristics.

While such situations are common, there are many contexts where trends in queue usage occur. For instance, if we are running a Google-like web service search engine, then, depending on various factors that influence the popularity of the service (our business plan, advertising strategies, the novelty and use of the service itself, etc), it is possible that there will be a genuine trend in the number of calls over time. In such a case, using an overall average to determine ρ will not provide the best immediate prediction of stability. It would be preferable to factor out earlier measures of calls (when the service was unpopular) and to emphasize the newer values, to predict if the system is likely to become unstable soon.

Some of the following results were developed by Duzbayev and Poernomo in [7].

ARMA forecasting strategies

ARMA was developed to treat trends in a more sophisticated way than simple averages.

One of the simplest strategies for time series prediction based on trends is to take an average of recent values of the time series, ignoring earlier values. This is the *simple moving average* technique. Here, the average is computed as

$$SMA(X_n, r) = \sum_{i=n-r+1}^{n} \frac{X_i}{r}$$

where r is cycle length and n is the total number of observations. This is essentially an arithmetic mean, but over a shorter cycle length than the total number of observations.

ARMA extends the simple moving average and is widely used in financial domains for forecasting time series. The equation for ARMA is

$$y[k] = \sum_{q=0}^{m} x[k-q]\beta_q - \sum_{p=1}^{n} y[k-p]\alpha_p \tag{1}$$

where $x[i]$ is a time series of input values and $y[j]$ is a time series of forecasted values of x, β_i and α_j are weights for the moving average and autoregressive components and m and n define how much of the time series the prediction method should consider. Given a particular time series $x[i]$ we can use least squares regression to determine optimal values of these coefficients. There is a way of obtaining the confidence interval for an ARMA forecast – see [7] for details.

So, the ARMA prediction for the k-th arrival rate $\hat{X}_n$ will be

$$\hat{X}_k = \sum_{q=0}^{m} X_{k-q}\beta_q - \sum_{p=1}^{n} \hat{X}_{k-p}\alpha_p \tag{2}$$

Prediction of utilization

Given the ARMA prediction of call arrivals, it is possible to forecast M/M/1 QoS characteristics by substitution of predicted values of X into the formula for utilization. Confidence intervals are calculated similarly.

We define the predicted utilization to be

$$\hat{\rho}_k = \hat{X}_k b$$

We can determine the confidence interval for predicted utilisation in terms of predicted arrival rate's error, using the fact that the error of $\hat{\rho}_k$ will be $\hat{X}_n b - X_n b = R_n^X b$:

$$\left[\rho_{n,k}^{min}, \rho_{n,k}^{max}\right] = \hat{\rho}_{n,k} \mp \frac{\xi}{\sqrt{k}} \sqrt{Var\left\{\sum_{m=1}^{k} R_{n,m}^X b\right\}}$$

Utilization prediction is useful in two cases:

- When there is a genuine trend towards instability. This is a serious problem for a queued system and pre-emptive notification can be very useful if an adaptation solution exists. For example, if a webservice has a predicted instability, administration could refuse any more requests until the queue normalizes.
- When there is a "local" trend towards instability. A time series might have a globally stable utilization, but with locally unstable segments. That is, a queued system might be able to respond to all requests eventually, but at certain times, might have an unacceptably high number of requests compared to service time. This situation can also benefit from pre-emptive notification to inform an adaptation strategy.

The previous confidence intervals for a prediction are helpful for determining the certainty we have of a current predicted trend in utilization.

The $n + k$ predicted probability of i calls being in the system queue will be

$$\hat{P}_{n,k}(i) = (\hat{\rho}_{n,k})^i (1 - \hat{\rho}_{n,k})$$

and the average number of requests in the system queue

$$\hat{E}[N_{n,k}] = \hat{\rho}_{n,k}/(1 - \hat{\rho}_{n,k})$$

with confidence interval

$$\left[E_{n,k}^{min}, E_{n,k}^{max}\right] = \hat{E}[N_{n,k}] \mp \frac{\xi}{\sqrt{k}} \sqrt{Var\left\{\sum_{m=1}^{k} \frac{R_{n,m}^X}{\left(\mu - \mu\rho_{n,m} - R_{n,m}^X\right)(1 - \rho_{n,m})}\right\}}$$

where the minimum values of the interval are at least limited by 0.

Example

To illustrate our predictive methods, we describe a simulated B2B web service based system. A queued server *WSDistributor* is a computer component distributor selling computer parts. There are 30 Client web services $WSAssembler_1, \ldots, WSAssembler_{30}$ that act as communication points to businesses that use the distributor for purchasing components which they then assemble into computers. Clients could make one of the following three types of call: makeOrder, cancelOrder or makeQuery. For the sake of simplicity, we do not differentiate between the different calls, so that the arrivals per time unit X may consist of any number of these method invocations. The architecture of the system is shown in Fig. 2.

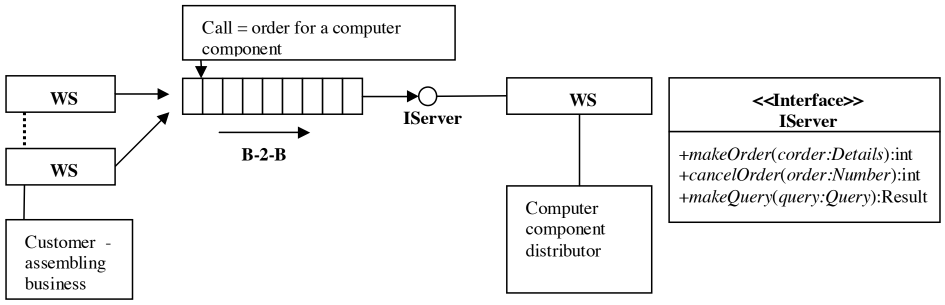
Call = order for a computer component
WS
WS
Customer - assembling business
B-2-B
IServer
WS
Computer component distributor
<<Interface>> IServer
+makeOrder(corder:Details):int
+cancelOrder(order:Number):int
+makeQuery(query:Query):Result

Figure 2: Our B2B example.

In order to obtain an initial working dataset of arrivals per time unit, we implemented the distributor webservice as an ASP.NET webservice, running on a Xeon 1,7GHz server running Windows Server 2003 and IIS 6. We ran the Microsoft Web Application Stress tool on a Pentium M laptop to simulate various demand profiles, both random and noisy trends.

Fig. 3(a) shows an example simulation of incoming requests per minute for a noisy trend. Fig. 3(b) shows an overall average calculation up to observation time i (the most horizontal graph), against the 1-step-ahead exponentially smoothened prediction of calls at i. Confidence intervals for the prediction are represented by the faded lines.

In this example, it is clear that the probable time to process a call will not always be best given by the overall average. For example, the client WS's will make many more requests of the distributor during business hours, and probably no requests after business hours. In this case, the overall average is of little use in defining, for instance, an adaptation strategy involving server replication and load balancing. The exponentially smoothened version does a good job at filtering out noise and effectively identifies peak trends at the (181,201) and (301,341) intervals, at other points remaining very close to the overall average.

A more sensitive smoothened average could be obtained by changing the coefficients of the ARMA equations: this would be useful if there are more subtle trends present in the time series. We should choose coefficients to avoid an overly

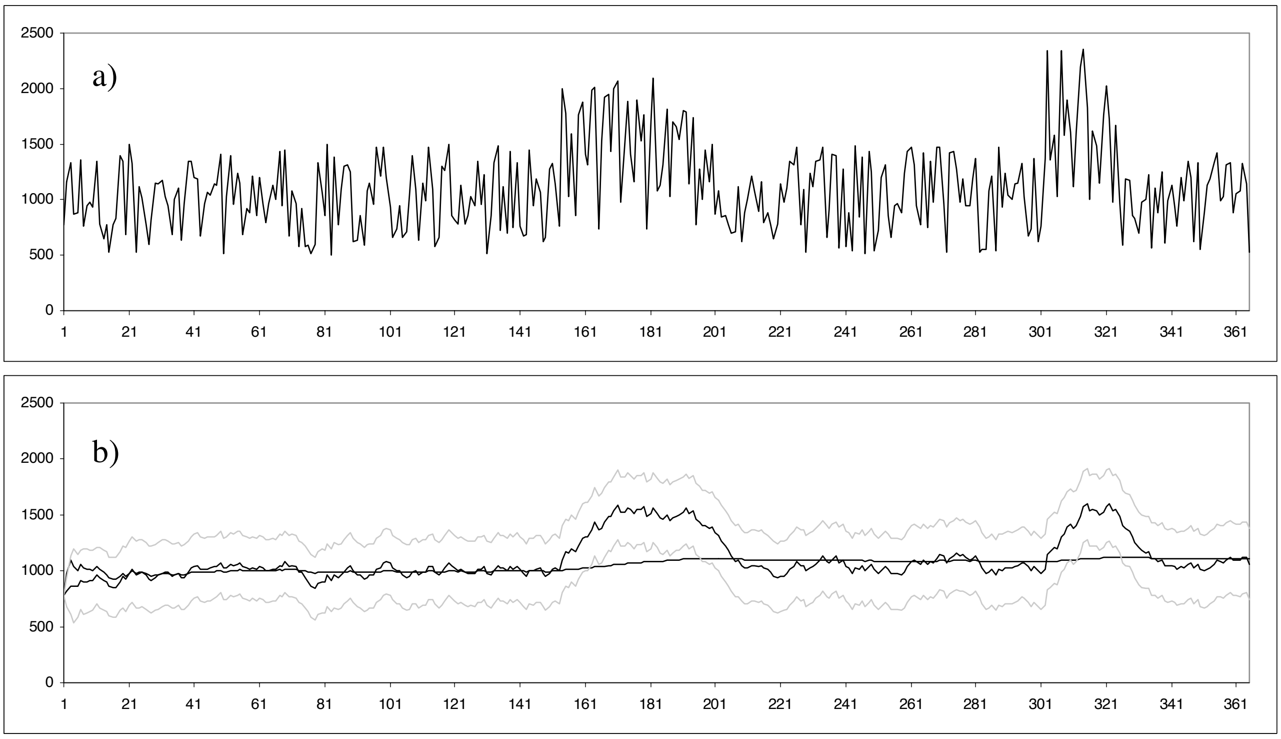

Figure 3: (a) Observations of incoming stream. (b) Three-step prediction of load rate using exponential smoothing of incoming stream with 95% confidential interval, plotted against the arithmetical mean. The mean is the most constant of the graphs. (Horizontal axes denote time and vertical axes denote incoming stream load.)

sensitive average that identifies false trends. However, from the perspective of adaptation methods (describe in the next section), we need not be overly cautious: it is generally better to adapt a system to a few false trend predictions than have no adaptation in the face of genuine performance deterioration.

The QoS characteristics can be calculated with confidence intervals following the discussion of the previous section. An 18 point plot of predicted utilization is given in Fig. 4(a), together with a 95% confidence interval. Calculation of the utilization by means of the overall average will yield a value of around 0.275 at any point in time. In contrast, our predicted utilization reaches about 0.42 during the second peak. The overall utilization never reaches such a high level, and so overall stability is certain. However, while stability is maintained overall, it is more useful to understand where "local" instability can be found – sections where predicted utilization goes above 1. At such regions of the time series, while all calls may *eventually* be served, there is the potential for backlog which, if trends continue, would result in global instability. Observe that the 95% confidence interval for the utilization forcast is remains reasonably close to the forcast. The exception to this is at earlier points, when there are significant random fluctuations in the number of requests received, resulting in a greater error of the prediction. This illustrates the importance of computing confidence intervals, particularly when our source data is noisy and there is uncertainty about whether a trend exists or not.

An 18 point plot of predicted number of requests to be served in the queue is given in Fig. 4(b), together a 95% with confidence interval. The predicted number of calls in the queue never reaches 1: that is, we never predict there to be an unanswered request in the queue at any time. However, the predicted number of requests peaks between (150,190) interval. This also suggests a potential trend towards a backlog. If the predicted trend increased to an unacceptable level, there might be cause to adapt the system to pre-emptively eliminate too many calls waiting in the queue.

Such adaptations are enabled through our control system, now described.

3 CONTROL SYSTEM MODEL

We now outline how classical control theory can be applied to develop a controller for changing a queue service rate pre-emptively in response to trends in utilization toward instability.

Specifically, we treat pre-emptive control of service rate in order to maintain a desired level of utilization. If the predicted utilization diverges above or below the desired level, we require our controller to apply a positive or negative *gain* to offset the divergence.

The idea is as follows. At any point in time, the utilization for the service can be computed in terms of the current average number of calls and average service rate. A predicted utilization model can also be determined, following the ARMA

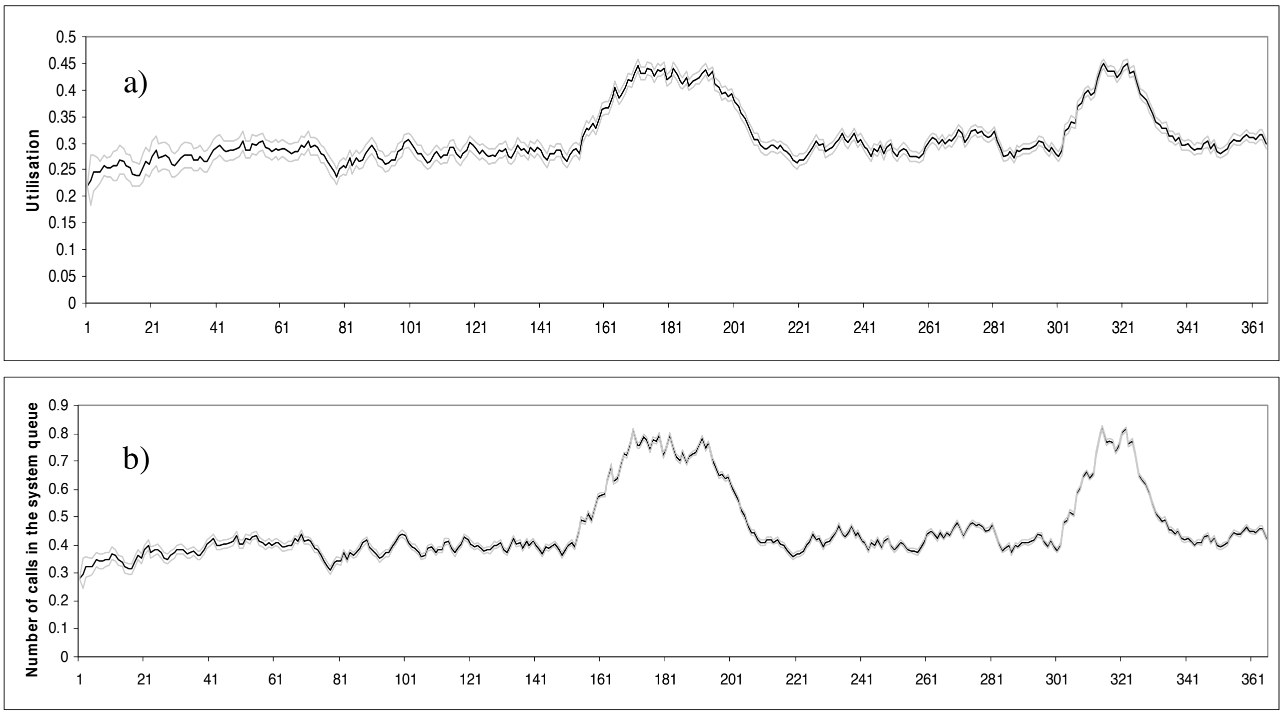

Figure 4: (a) Observations of incoming stream. (b) Three-step prediction of load rate using exponential smoothing of incoming stream with 95% confidential interval, plotted against the arithmetical mean. The mean is the most constant of the graphs. (Horizontal axes denote time and vertical axes denote incoming stream load.)

technique of the previous section, to give a picture of how we expect the utilization to continue, given known trends. Consider a constraint that the utilization should not exceed a certain amount. The *control error* of the service is the difference between the predicted utilization and a desired utilization. The controller is constructed to manipulate the average service rate in such a way that this error is minimized. The controller is characterised by a function over the error and a constant *gain* parameter. Each adaptation is associated with a cost, so the controller must also ensure that cost is minimal. We construct a closed-loop transfer function model for the system. This is a complex valued function. Root locus analysis is a technique whereby the poles of the transfer function are identified in order to determine the optimal gain for the controller function.

Control systems

Control systems to consist of a series of interconnected 'plants'. The term 'component' is often used in place of 'plant' by control engineers, but to avoid confusion, we shall only use the term 'component' in the software engineering sense. However, a plant does have some similarities to a software component, that we expound upon in the next section. A plant is considered as a functional module, taking an input signal and returning an output signal.

Transfer functions

The relationship between input and output relationship for a plant is often characterised by a transfer function. In the case of continuous relationships, where input and output are related via time-invariant, differential equations, the transfer function is given as the Laplace transform of the output over the Laplace transform of the input. The analogous situation for discrete relationships uses a Z-transform instead of a Laplace transform:

$$\text{Transfer function} = G(z) = \frac{\mathcal{Z}\{output\}}{\mathcal{Z}\{input\}}$$

The Z-transform $\mathcal{Z}$ is essentially a Fourier transform generalised over the complex plane [12]. Given a signal $x[n]$, the Z-transform is defined as

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{n=0}^{\infty} x[n] z^{-n}$$

where z is some complex number $z = Ae^{j\phi}$.

Transfer functions are used because they allow us to represent system dynamics by algebraic complex equations. If the highest power of z in the denominator of the transfer function is equal to n, the plant is called nth-order.

The transfer function for a plant enables us to understand its output for various forms of input. Transfer functions can be experimentally derived by studying how a plant reacts to a range of sample input values.

Automatic control systems and block diagrams

A block diagram of a system represents how signals flow from one plant to another. Each plant is represented by a square block, with the name of its transfer function drawn inside. Arrows connecting plants define the way in which the output of one plant is passed on as the input for another plant. Two signals can be added or subtracted from one another at what is called a *summing point*. A signal can split and be fed concurrently into several other blocks or summing points – the point of such a split is called a *branch point*.

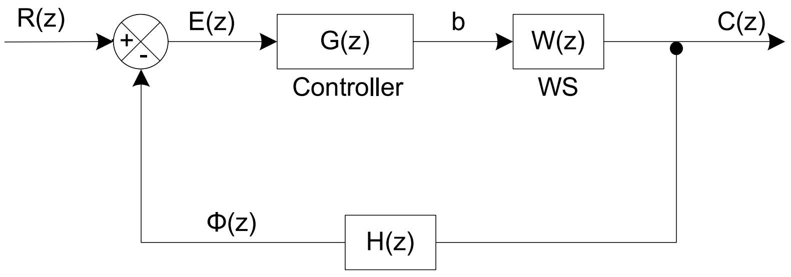

Figure 5: Control flow for a single component architecture.

A simple *automatic control system* is a system with the block diagram of Fig. 5. It consists of three plants with transfer functions $G(z)$, $W(z)$ and $H(z)$, with input and output signals being sent as depicted by the arrows. The output signal of $W(z)$ branches out of the system and also is fed into a summation point (depicted by the circle), where it is subtracted from a reference input R (summation of the two signals would be depicted by two positive symbols in the circle).

The entire system forms a plant, with an input R and an output C. It can therefore be associated with a transfer function of the form

$$\frac{C(z)}{R(z)} = \frac{G(z)W(z)}{1 + G(z)W(z)H(z)} \tag{3}$$

This is called *the closed-loop transfer function* for the system. The term

$$G(z)W(z)H(z)$$

relates the signal $\phi(z)$ to the error signal $E(z)$, and is called the open-loop transfer function.

Control system for a queued service

Our intention is to define a control system of the form given in Fig. 5, where

- The reference input R is the desired level of utilization to be maintained for the service.
- The service is modelled by the plant W. We are not concerned with modelling the actual functionality of the service here. Rather, we are only interested in its current utilization. Therefore, in the control system, it is modelled as a function that takes in a controlled service rate b from the controller G and outputs a current utilization rate $c = Xb$, where X is the current number of calls joining the queue. The transfer function for the service is

$$G(s) = \frac{\mathcal{Z}\{output\}}{\mathcal{Z}\{input\}} = \frac{X(z)b(z)}{b(z)} = X(z)$$

  Thus, the transfer function is parametrised over the model for the service's arrival process. For example, if we consider a service with a $M/M/1$ queue, we assume the service is characterized by a Poisson arrival process and a FIFO queue ordering discipline. In this case, the transfer function for G will be the Z-transform of a Poisson distribution for arrivals X.

- The current utilization is fed into an prediction plant H that forecasts the likely future utilization of the service, using an ARMA function of the form (1), with coefficients optimised according to the service's queuing model. There is a well-known result in discrete control theory that relates ARMA to a Z-transform, which tells us the Z-transform for (1) is

$$H(z) = \frac{\Phi(z)}{C(z)} = \frac{\sum\limits_{q=0}^{m} z^{-q}\beta_q}{1 + \sum\limits_{p=1}^{n} z^{-p}\alpha_p}$$

Control laws

Our control system, following Fig. 5, should also include a controller plant. We then identify the controller plant defined by a function over the error signal, taken as the difference between the desired utilization and the predicted utilization

$$e = R - \phi$$

The controller plant G takes this error as input and will return a service time b that will attempt to offset any trends away from the desired utilization. This service time is taken as the input for the plant that models the queued service, W.

Control theory offers a range of possible equations that can define the controller's actions. Our approach works equally well with any of these strategies. For the purposes of illustration, we employ the integral control action strategy, in which the controller output value, the controlled service time, $b(t)$ is changed at a rate proportional to the error signal $e(t)$,

$$\frac{db(t)}{dt} = K_i e(t)$$

where K_i is a constant parameter, called the *gain* of the controller. Because we consider only discrete time, this becomes

$$b(t) = K_i \sum_{t=0}^{t} e(t)$$

The Z-transform for an integral controller is

$$G = \frac{K_i}{z}$$

Note that the strategy is a simplification: in practice, the control strategy will be implemented by discrete actions (for instance, increasing the service time by diverting calls to replicated servers) offering generally coarser shifts in b. The relation between the controller plant *model* and its implementation is addressed in the next section.

Clearly a higher value of K_i will lead to a faster correction to predicted divergence from a desired utilization. However, at some point, higher values of K_i will "overshoot", leading to a large predicted divergence in the opposite direction that will then require another large correction. When this occurs, the value of K_i results in feedback through the control system and oscillations in control actions. In terms of implementation, this is undesirable as oscillations will effect performance, as each control action will be associated with a cost.

So, we require the highest value of K_i that does *not* overshoot in this fashion. This can be determined through root locus analysis of the closed-loop transfer function for the control system.

The root locus method involves finding the roots of the characteristic equation for the closed-loop transfer function are plotted for all values of the gain parameter K_i. The characteristic equation for the closed-loop system is obtained by setting the denominator of (3) to zero. This occurs when the open loop transfer function GWH is equal to -1,

$$G(z)W(z)H(z) = -1$$

The method enables the analyist to determine the effects variations on K_i will have on the location of closed-loop poles. While theoretically quite complex, the solution for systems of our form can be easily determined automatically by MATLAB.

The *damping* ratio for a root locus plot determines the speed at which we would like the controller to converge to a desired value after a predicted divergence, assuming no further external perturbations to the utilization arising from changes to the arrival rate X. A lower damping ratio is desirable, but should be balanced with as high a value of K_i as possible. A given damping ratio r can be used to determine the value of K_i from the root locus plot: the value is found from the point on the plot that makes an angle $cos^{-1}r$ with the negative real axis. Because the ratio should be between 0 and 1, with a ratio above 1 leading to oscillations, all non-oscillating values of K_i are to be found in the upper left hand side of the complex plane for the the root locus plot.

4 MODEL DRIVEN DEVELOPMENT OF CONTROLLED ARCHITECTURES

The method of the previous section developed a controller function that determines how the service rate should be changed in order to avoid undesirable trends in utilization. This function is a model of how control should work for a particular queued service. We now sketch our approach to controller generation. We first develop control systems for each constrained component of our extended version of the UML2 superstructure. Then we apply a so called Y model-transformation, mapping UML components and their associated control systems to .NET based implementations.

We have implemented a beta version of our translations using the INRIA Triskell group's Kermeta transformation language. In this language, model transformations are defined as meta-operations of M2 metaclasses, whose input types are Platform Independent metaclasses and whose output types are Platform Specific metaclasses. This has the advantage of providing a unifying MOF-style framework for understanding both metamodels and model transformations.

We omit the full description of the transformations and instead describe their behaviour informally. As a motivating example for our transformations, we define a simple application of model transformation, involving a constraint over utilization imposed over method calls to a component.

Three models are given given in Fig. 6. We use the UML2 superstructure extended with QoS constraints, as defined in [3]. Our transformations map this to a control system model, that defines an adaptation strategy.

5 EXAMPLE

To illustrate our method for control, we continue with the webservice based example of section 2.

We then ran least squares regression over the dataset to obtain an optimal set

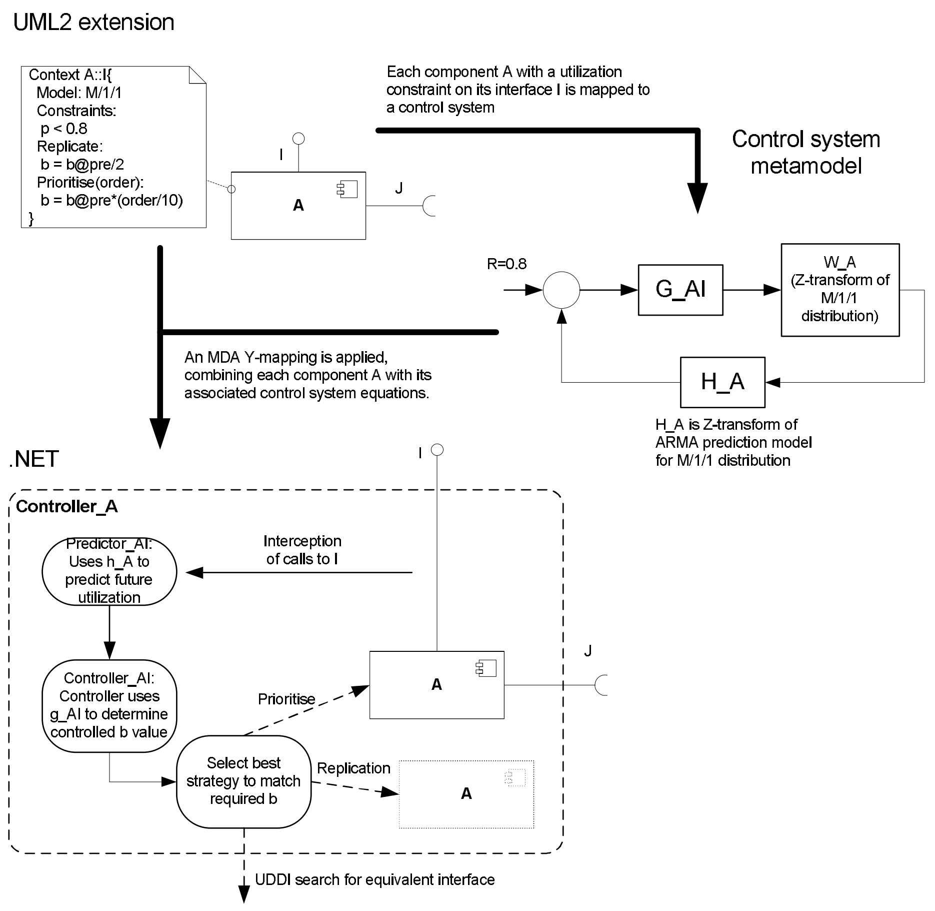
UML2 extension
Context A::I{
Model: M/1/1
Constraints:
p < 0.8
Replicate:
b = b@pre/2
Prioritise(order):
b = b@pre*(order/10)
}
I
J
A
Each component A with a utilization constraint on its interface I is mapped to a control system
Control system metamodel
R=0.8
G_AI
W_A (Z-transform of M/1/1 distribution)
H_A
H_A is Z-transform of ARMA prediction model for M/1/1 distribution
An MDA Y-mapping is applied, combining each component A with its associated control system equations.
.NET
Controller_A
Predictor_AI: Uses h_A to predict future utilization
Interception of calls to I
Controller_AI: Controller uses g_AI to determine controlled b value
Prioritise
Select best strategy to match required b
Replication
A
UDDI search for equivalent interface

Figure 6: Model driven development of the controlled system.

of 10 α_i and β_j parameters for an ARMA model of the form (1) with $n = m = 9$.

The model transformations of the previous section generate a context for the server, mapping this ARMA model to prediction pre-call code in the context. We assume each call type has the same processing time b.

The architecture of the system assumes an $M/M/1$ queuing model, with a Poisson arrival distribution. This information is used in the UML to controller transformation, generating a control system model with the following description:

$$\phi(n) = ARMA(c(n))$$

$$H(z) = \frac{\Phi(z)}{C(z)} = \frac{\sum_{q=0}^{9} z^{-q}\beta_q}{1 + \sum_{p=1}^{9} z^{-p}\alpha_p}$$

$$GWH = \left(\frac{K}{s}\right) HX = \left(\frac{K}{s}\right) H \left(\frac{2.55s - 2.023}{s + 1.3}\right)$$

so that GWH is

$$\frac{2K\left(\beta_0 z^{10}+(\beta_1-\beta_0)\,z^9+(\beta_2-\beta_1)\,z^8+\ldots+(\beta_9-\beta_8)\,z-\beta_9\right)}{z^{11}+(\alpha_1+1)\,z^{10}+(\alpha_2+\alpha_1)\,z^9+(\alpha_3+\alpha_2)\,z^8+\ldots+(\alpha_9+\alpha_8)\,z^2+\alpha_9 z}$$

We then apply root-locus analysis over this equation to determine the optimal value of K. We plot the roots of the characteristic equation against all values of K. This can be done easily in MATLAB, producing Fig. 7. The poles are denoted by an x and the zeros by a o. We can use this to determine the value of the gain that will make the damping ratio of the dominant closed-loop poles as prescribed. For example, valid, non-oscillating values for K_i are .1 with damping ratio .2, .5 with damping ratio .193 and 1 with damping ratio .63.

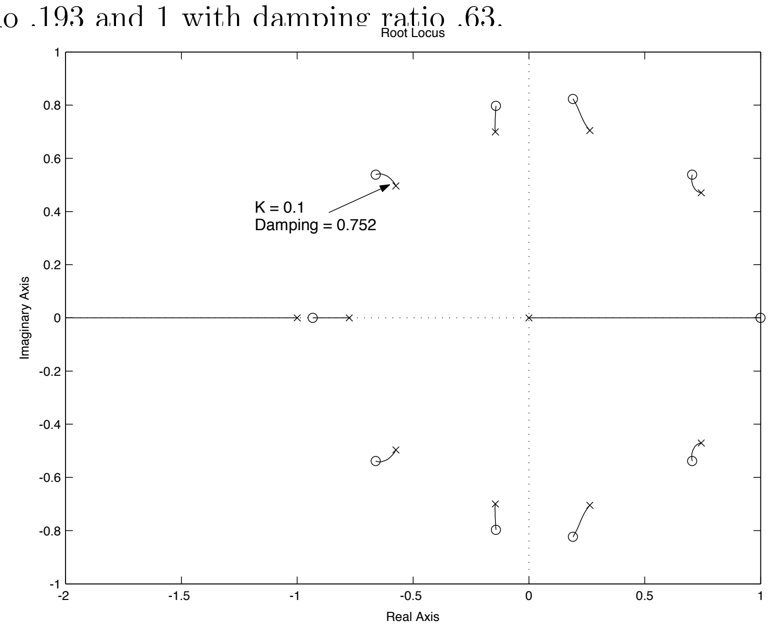

Figure 7: Root loci for our example.

Fig. 8 shows controlled utilization with valid, non-oscillating gain values (0.1 and 1) against uncontrolled utilization. As can be seen, higher gain results in faster response to trends away from the desired utilization level of 0.7. Because there is always random arrival rates, the controlled utilization will never be exactly at the desired level, but it maintains a satisfactory level of stability. Most notably, when the uncontrolled utilization moves into instability between time units 144-199 and 298-331, both graphs show the controlled utilization maintains a much better level of stability. Fig. 9 shows controlled utilization for oscillating gain values with a damping ratio of greater than 1. Clearly the controlled utilization here "overshoots" the correction to deviation from required utilization. In particular, the controller leads to undesirable oscillations between a near zero utilization (corresponding to, for instance, a very high number of replicated servers) and very unstable utilization (corresponding to a single overloaded server) at the points where uncontrolled utilization exhibits prolonged instability. This makes the controlled utilization worse than retaining ordinary utilization.

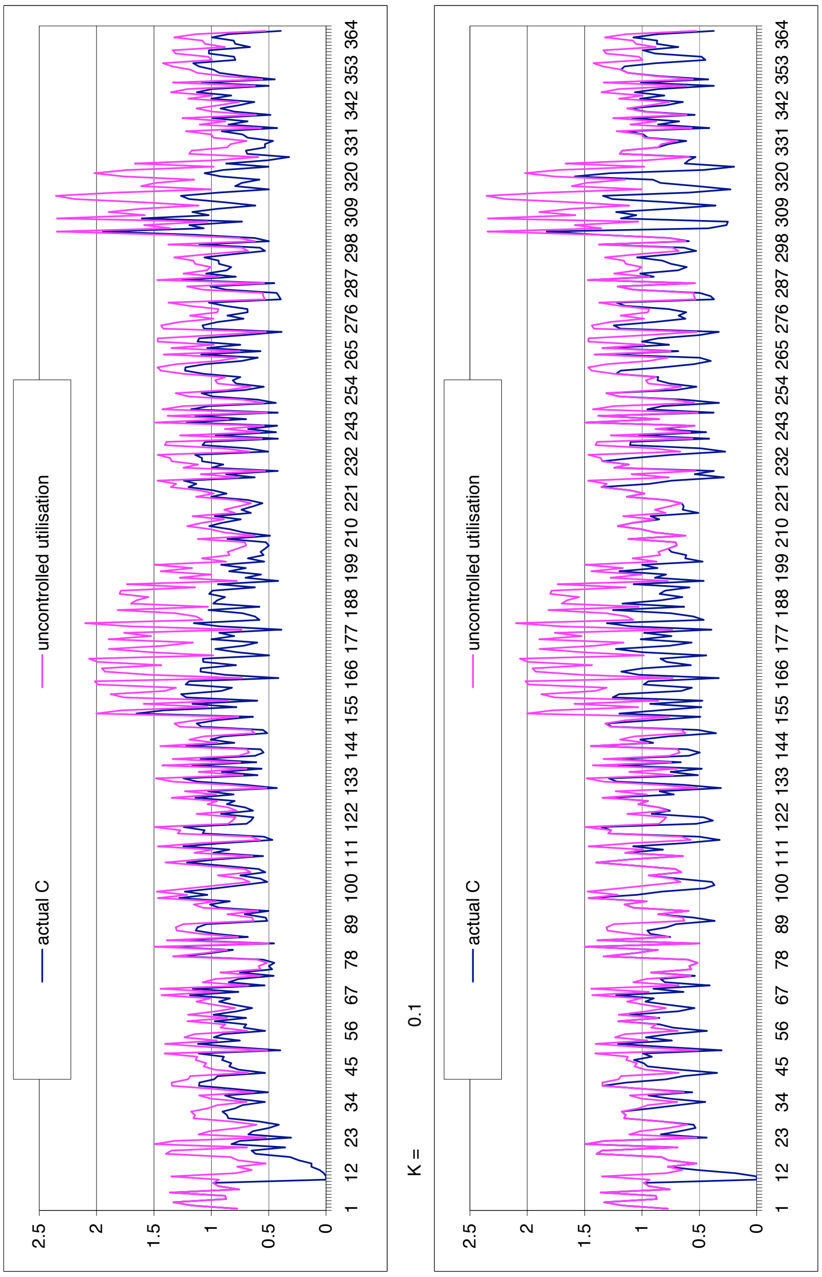

Figure 8: Two plots of controlled utilization with valid, non-oscillating gain values against uncontrolled utilization.)

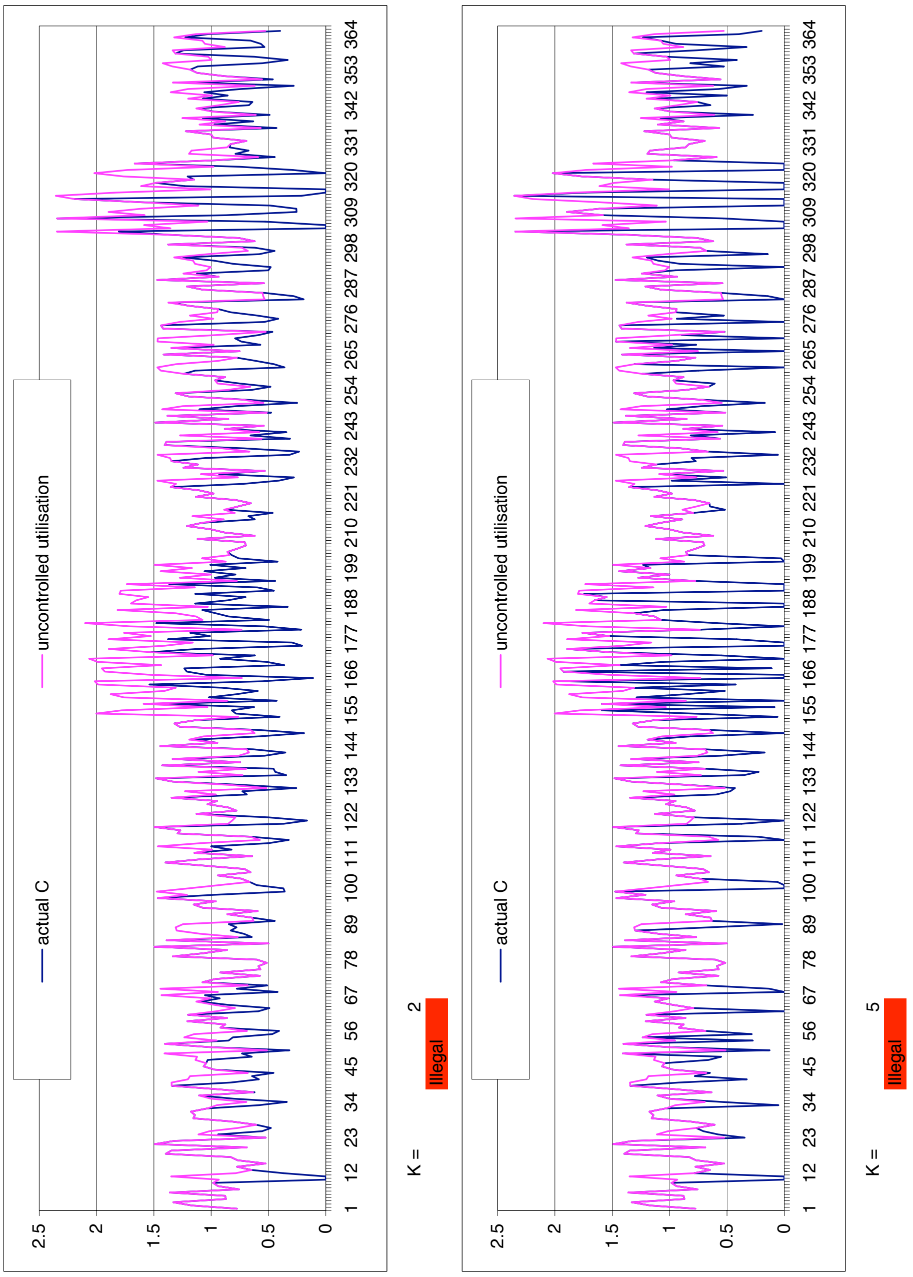

Figure 9: Two plots of controlled utilization with invalid gain values against uncontrolled utilization.)

In the implementation, the service rate is changed by means of several adaptation strategies. 1) The server can be replicated n times, with yielding a new processing time equal to bn. 2) There are equivalent versions of the server available to access via UDDI, each with a variable serving time. If calls are diverted equally to one of these servers with service time b', then the new overall processing time is $b + b'$. These strategies are mapped to server selection code that forms the final part of the pre-call in the context generated by the transformation. Given a desired service level b, the selection code chooses the best strategy to meet the target. Provided the service rate change strategies are sufficiently fine grained enough, the selector can follow the controller's requests closely.

6 RELATED WORK AND CONCLUSIONS

There are many systems that permit dynamic adaptation of architectures based on real time QoS information [2, 1, 18, 4, 11, 16]. The work of [18] is most similar to ours, as they use UDDI and QoS information to assemble web service architectures of optimal performance. Adaptation has also been proven to be useful in a range of other contexts. For example, [1] defines a language of QoS policies for grid services that are enforced by means of adaptation mechanisms. A different approach to QoS adaptation is considered in [17] for the case of embedded systems. These systems do not involve control theoretic notions or forecasting of values as part of their adaptation strategies. The ARMA methods need not only be applied to compute QoS queue characteristics. These strategies have the potential to be combined with such (non-queued) QoS-based runtime adaptation technologies.

Similar forecasting methods are used by Dinda for host load prediction [6] of CORBA based systems and the performance prediction methods of [2] and [8]. The difference with that works is that we adapt ARMA methods to queued models, instead of load time estimation models. The intention behind our system is analogous to that of the Running Time Advisor of [6], but applied to widely distributed webservice component architecture.

Further work on stability for more complicated queuing strategies is given in [9]. These results could be adapted to our context, to enable us to predict stability for systems involving multiple servers.[10] presents the most completed overview of different stochastic methods to examine queued system to stability. There are about ten methods. Each of them represents particular interest of investigation, study and especially applying. Most of these methods have been contrived to be applied in different parts of science, but never was applied to predict behaviour of complicated computer systems.

We intend to improve the trustworthiness of our model-driven approach next. While the mathematics behind the adaptive approach is sound, there remains the problem of a semantic gap between the platform independent specification of adaptive constraints and the platform specific control strategies. There is no formal guarantee that the latter strategies are accurately generated to satisfy the former specifications – we simply trust that the transformation writer has done his/her job in providing an accurate generation. This problem holds for any MDA context, not only our own. However, because the transformations are between models that have a formalizable semantics, it should be possible to apply the methods of [14] and [15] to obtain provably correct generation strategies.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their helpful commments, and also the reviewers and audiences for previous presentations of some of this material at the QoSA and Aquserm events. The authors are especially greatful to Iman Poernomo for his guidance and supervision on this topic.

REFERENCES

[1] Rashid Al-Ali, Abdelhakim Hafid, Omer Rana, and David Walker. An approach for quality of service adaptation in service-oriented grids. *Concurrency and Computation: Practice and Experience*, 16(5):401–412.

[2] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-based performance prediction in software development: A survey. *IEEE Transactions On Software Engineering*, 30(5):295–310, MAY 2004.

[3] Kenneth Chan and Iman Poernomo. Qos-aware model driven architecture through the UML and CIM. *Information Systems Frontiers*, 9(2-3):209–224, 2007.

[4] Luiz Marcio Cysneiros and Julio Cesar Sampaio do Prado Leite. Nonfunctional requirements: From elicitation to conceptual models. *IEEE Transactions On Software Engineering*, 30(5):328–350, MAY 2004.

[5] Brian D.Bunday. *An introduction to queueing theory.* New York, Halsted Press, 1996.

[6] Peter A. Dinda. Online prediction of the running time of tasks. *Joint International Conference on Measurement and Modeling of Computer Systems*, pages 336 – 337, MAY 2001.

[7] Nurzhan Duzbayev and Iman Poernomo. Runtime prediction of queued behaviour. In *Quality of Software Architectures and Software Quality, Second International Conference on the Quality of Software Architectures, QoSA 2006*, LNCS. Springer, 2006.

[8] Paul J. Fortier and Howard E.Michel. *Computer Systems Perfomance Evaluation and Prediction.* Digital Press, 2003.

[9] S. Foss and N. Chernova. On stability of a partially accessible multi-station queue with state-dependent routing. *Queueing Systems*, 1(29):5573, 1998.

[10] S. Foss and T.Konstantopoulos. An overview of some stochastic stability methods. *Journal of the Operations Research Society of Japan*, 47(4):275–303, 2003.

[11] George T. Heineman, Joseph P. Loyall, , and Richard E. Schantz. Component technology and qos management. *International Symposium on Component-based Software Engineering (CBSE7), Edinburgh, Scotland*, May 24-25 2004.

[12] Eliahu Ibraham Jury. *Theory and Application of the z-Transform Method*. John Wiley and Sons, 1964.

[13] L. Kleinrock. *Queueing Systems*, volume 1. New York, J. Wiley, 1975.

[14] Iman Poernomo. A type theoretic framework for formal metamodelling. In *Architecting Systems with Trustworthy Components*, volume 3938 of *LNCS*, pages 262–298. Springer, 2006.

[15] Iman Poernomo, John Crossley, and Martin Wirsing. *Adapting Proofs-as-Programs: The Curry-Howard Protocol*. Monographs in Computer Science. Springer, 2005.

[16] Iman Poernomo, Heinz Schmidt, and Jane Jayaputera. Verification and prediction of timed probabilistic properties over the distributed management taskforce common information model. *International Journal of Cooperative Information Systems*, 15(4):633–658, December 2006.

[17] Praveen K. Sharma, Joseph P. Loyall, George T. Heineman, Richard E. Schantz, Richard Shapiro, and Gary Duzan. Component-based dynamic qos adaptations in distributed real-time and embedded systems. *International Symposium on Distributed Objects and Applications (DOA) , Agia Napa, Cyprus*, pages 1208–1224, October 25-29 2004.

[18] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Transactions On Software Engineering*, 30(5):311–327, MAY 2004.

ABOUT THE AUTHORS

Assel Akzhalova is a Ph.D. student at the Predictable Assembly Laboratory, a subgroup of the Department of Computer Science, King's College London. She holds a Candidate Ph.D. degree in control systems from the Al-Farabi Kazakh National University where she is an associate professor.

Assel Altayeva holds a Master's degree in mathematics from Novosibirsk State University, specializing in domain theory. Her main research interests lie in the application of domain theory to provide a formal foundation for optimal control.

Nurzhan Duzbayev is a Ph.D. student at the Predictable Assembly Laboratory, King's College London. He holds a Masters degree in control systems from the Al-Farabi Kazakh National University.