Performance Sensitive Self-Adaptive Service-Oriented Software using Hidden Markov Models *

Diego Perez-Palacin
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza
Zaragoza, Spain
diegop@unizar.es

José Merseguer
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza
Zaragoza, Spain
jmerse@unizar.es

ABSTRACT

Service Oriented Architecture (SOA) is a paradigm where applications are built on services offered by third party providers. Behavior of providers evolves and makes a challenge the performance prediction of SOA applications. A proper decision about when a provider should be substituted can dramatically improve the performance of the application. We propose hidden Markov models (HMM) to help service integrators to foretell the current state of third-parties. The paper leverages different algorithms that change providers based on predictions about their states. We also integrate these algorithms and HMMs in an architectural solution to coordinate them with other challenges in the SOA world.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*performance measures*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*

General Terms

Design, Performance

Keywords

Open-world software, service oriented architecture, hidden Markov models, software architectures

1. INTRODUCTION

Service Oriented Architectures (SOA) provide abstraction mechanisms to ease building complex, heterogeneous and distributed software systems. A pillar of these architectures is the concept of *service*, which is a software entity that allows to execute functionalities in a loosely coupled manner and whose interface is well-described. As in [3] and [2], we call *abstract service* to the offered functionality, and *concrete service* to the particular service the provider exposes. A service can be offered by several providers and they are differentiated by their QoS. Moreover, although the QoS of providers were similar in the long term for the same service, in a given moment providers are exhibiting different QoS; e.g., their performance can differ due to the current workload.

Concrete services can be invoked as part of a complex service oriented system which acts as a *service integrator*. A service integrator can follow a workflow of requests to services provided by third-parties. Working with third-parties adds new concerns since services are deployed and maintained without control of our organization, which implies that their QoS variations are unpredictable. Furthermore, new services can be developed or existing ones can be retired from the market. *Open-world* software [1] is a paradigm that suitably considers these new concerns. In open-world, systems are aware of both, their own behavior and environment, and can react to changes by adapting their behavior. This kind of systems is called self-adaptive or self-managed [11, 5, 4]. Due to the QoS uncertainties in an open-world, self-adaptive systems have become an important research topic. SOA eases adaptation tasks because of its intrinsic properties, such as service discovery or dynamic binding.

The approach in this work deals with the tasks of deciding *when* and *how* adaptation actions should be executed to maintain the system working with a suitable performance. Here, adaptation actions carry out a change in the service provider that will be requested to execute the required abstract service. The approach relies on some knowledge of the performance of service providers. Concretely it assumes that the number of different "performance states" for a given provider is finite, so it supports providers acting with varying performance. Moreover, it is also considered the mean sojourn time that providers spend in each performance state and the probability of moving from one state to another. Since services are provided by third-parties, service integrators cannot assume knowledge of the actual performance of the third-parties. On the contrary, integrators should collect data during runtime to predict this information. Later, integrators will use such prediction to decide whether adapt the request of the system to providers with higher performance. Our approach uses Hidden Markov Models (HMMs) [10] to foretell the performance states of providers based on monitored information related to their response time. The approach herein presented continues the work in [9], where we started to study the generation of performance-aware adaptation strategies for software systems.

Section 2 describes the type of HMMs we use and how they are useful for our purposes; Sections 3 and 4 develop the approach. Section 5 concludes the paper with the related and future work.

*The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no 224498 and from the Spanish Plan Nacional de I+D+i 2008-2011 under grant no DPI2010-20413.

ICPE'11, March 14–16, 2011, Karlsruhe, Germany.

2. A FORMAL MODEL FOR SOA PROVIDERS

2.1 Hidden Markov Models

A Hidden Markov Model (HMM) [10] is a double stochastic process, where one of the processes is observable and unobservable the other. We use this model to predict the state of the unobservable process by means of the observable one. An HMM is characterized by the following [10]:

1. The number of states in the model, N. These states are hidden, and they use to represent the meaning of what is intended to be predicted. Individual states are denoted as $S = \{s_1, s_2, ..., s_N\}$, among them the actual state at time t is denoted as q_t.

2. A state transition probability distribution matrix $A = \{a_{ij}\}$, $1 \leq i, j \leq N$. Transition probabilities are assumed normalized, i.e., $\forall i \sum_{j=1}^{N} a_{ij} = 1$.

3. An initial state distribution $\pi \in (\mathbb{R}^+ \cup 0)^N$, where $\pi(i) = P[q_1 = s_i]$.

The former three characteristics conform to a discrete time Markov chain (DTMC). The state of the DTMC at each instant of time will be the unobservable process.

4. Number of distinct observation symbols per state, M. They are denoted as $V = \{v_1, v_2, ..., v_M\}$.

5. The observation symbol probability distribution in state i, $B = \{b_i(k)\}$, where $b_i(k) = P[v_k \; at \; t | \; q_t = s_i]$.

Continuous observation density HMMs.

Since our observations will be measured times, which are continuous values, the number of observed symbols $M \rightarrow \infty$. Moreover, such measures are expected to follow an exponential distribution with parameter λ_i. Therefore, we consider the HMM to be a single continuous observation, then items 4 and 5 change to: probability distribution function of observation O in state i is $b_i(O) = \lambda_i e^{-\lambda_i O}$. [1]

Continuous time HMMs.

Advancing descriptions in next subsection, our approach models the non-observable behavior of third-party providers. Such behavior is defined by several states, mean sojourn time in each state, and transition probabilities among states. So, it will be needed to model duration of states in the HMM. Unfortunately, citing [10], *perhaps the major weakness of conventional HMMs is the modeling of state duration*. To overcome this weakness, we use the modeling approach in [14], and we will work with continuous time Markov chains (CTMC). As a result, items 1,2 and 3 will describe a CTMC. Items 1 and 3 do not change their meaning, but transition probabilities are converted into transition rates taking into account both the mean sojourn time in each state (Soj_i) and the probability to change from state s_i to state s_j ($p_{s_i s_j}$) where ($\sum_{s_j} p_{s_i s_j} = 1 \wedge p_{s_i s_i} = 0$). Consequently, item 2 is redefined: There is a state transition rate matrix $R = \{r_{ij}\}, 1 \leq i, j \leq N$, where $r_{ii} = \frac{-1}{Soj_i} \wedge \forall i \neq j, r_{ij} = \frac{p_{s_i s_j}}{Soj_{s_i}}$

[1] $b_i(O)$ can be any finite mixture of log-concave or elliptically symmetric densities $\mathfrak{N}$ [10], $b_i(O) = \sum_{m=1}^{M} c_{im} \mathfrak{N}(O, \mu_{im}, U_{im})$ where c_{im} are mixture coefficients, μ_{im} are means and U_{im} are covariance matrices. In our case, the mixture has only one component, which is the exponential probability distribution function.

From now on, we call CT-HMM to an HMM with continuous observation density, and its state change behavior is given by a CTMC. Figure 1 depicts an example of CT-HMM.

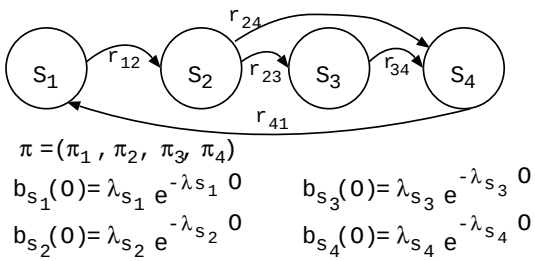

Figure 1: CT-HMM example

2.2 HMM representation of SOA providers

Let us consider the case of a client with an *abstract* service that requests for an item of news, and there exist several providers of news that offer the *concrete* service. The information the client can manage, about the behavior of the providers, is the one it can acquire monitoring their performance. For example, Figure 2 shows the monitored response time of a provider[2]. We can observe that: 1) during the night, the provider shows a low response time, which is around 100 time units (tu), maybe since few clients (human or service integrators) are consulting news; 2) during the weekend, less than 500tu are needed to generate the response; 3) during daylight of working days the response time increases up to 1100tu, maybe because the workload to request news increases; 4) in such daylight, there can also be peak zones where the time to receive a response suddenly reaches maximums. Peak zones can be predictable or unpredictable. For example, a predictable one is at the beginning of the day (around 9:00 hours), when lots of people may want to read the news; while an unpredictable one may occur when an important event happens (e.g., a crime report). From the information in Figure 2, the client could group the behavior of the provider in four states: *daylight*, *daylight-peak*, *night*, *weekend*. See that another provider of the same news service can show different response times (better in certain moments and worse in others). The variation can be due to the workload the provider supports in each moment, which indeed is an unknown information from the client view. Moreover, the long-term behavior could even be the same (same states and mean expected response time), but they are showing different response time in a given moment just because they belong to different time-zones.

The client should address each service call to the provider that shows best performance at the moment the request is delivered. Then, the client has to predict the future expected response time of the providers based on the response times it has already monitored. Besides, providers are not monitored strictly periodically but when their services are requested. This fact can make prediction methods that are exclusively based on monitored response times lose accuracy. Since we pursue accurate prediction methods, we consider two concepts to carry out providers performance prediction: monitored response times and the time instant they are measured. CT-HMMs provide mechanisms to represent both concepts. Then, we propose them for the client to represent the behavior of a provider:

[2] This figure has actually been acquired from [7], and it depicts real workloads for one of the webmail providers of the University of Zaragoza during the week of 20-26 june 2010.

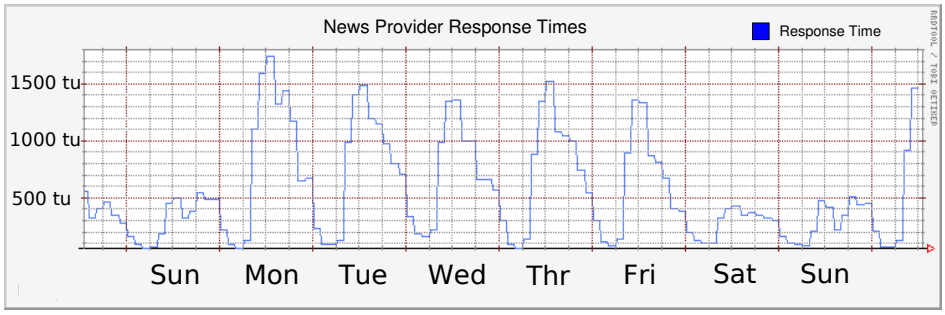

Figure 2: Response times of a service provider

- A continuous time Markov chain (CTMC) where each state represents a state of the provider (*night, daylight,...*) and the client knows their mean sojourn times s_s (Soj_{s_s}) and the probability to change from a source state to a target state ($p_{s_s s_t}$ and $\sum_{s_t} p_{s_s s_t} = 1$).

- Probability of the value of observations in each state. In this case, observations are the response times monitored from service calls. For us, the expected response time follows an exponential distribution in each state. Therefore, the probabilities of observations in state $s_i \in \{night, daylight, ...\}$ are $b_{s_i}(O) = \lambda_{s_i} e^{-\lambda_{s_i} O}$, where λ_{s_i} is the inverse of the mean expected response time in state s_i and O is the observed time by means of monitoring the provider.

- An initial state distribution π_0. Since no knowledge about the state of the provider is known when the service integrator starts its execution, we assume the initial state distribution to be the steady-state solution of the CTMC $\pi_0 = \pi_{steady}$.

Note that in this model, states are hidden but transition rates among states are known, as well as the expected mean response time in each state. CT-HMM manages two time parameters when receives each observation k: the monitored response time (O_k) and the time instant when such observation has been received (t_k). That is, the service integrator stores the absolute time in which each response has been received, being $t_0 = 0$ the instant where the integrator was switched on.

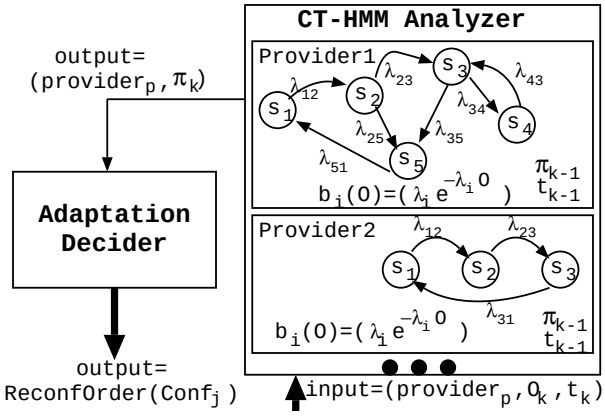

Figure 3: Modules for a change of configuration

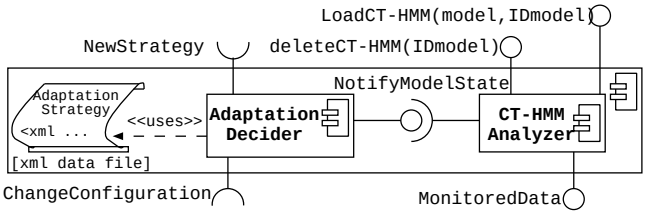

Figure 4: Abstract view of Figure 3.

3. STATE PREDICTION AND CONFIGURATION ADAPTATION

As discussed in previous section, the behavior of providers can be formally represented and the client, or service integrator, can use measured response times to foresee in which state a provider should be currently executing. Besides, service integrators need abilities to change the system configuration, i.e. to select for the current request the provider that has been predicted to be in the state with best response time.

3.1 Prediction of the state of a provider

The CT-HMM proposed in Section 2.2 will be useful to predict for a provider the probability distribution of its states. Concretely, using the CT-HMM of provider p, we can predict the state probability (π_k^p) when observation O_k^p is received at time t_k^p considering: the calculated state probability of the previous observation (π_{k-1}^p), the observed response time O_k^p and the amount of time elapsed, $t_k^p - t_{k-1}^p$, since the previous service call to p:

$$\pi_k^p(i) = CalcTransientProb(\pi_{k-1}^p, s_i, t_k^p - t_{k-1}^p) \cdot b_i^p(O_k) \cdot c \quad (1)$$

being $b_i^p(O_k) = \lambda_i e^{-\lambda_i O_k}$, i.e., the probability of provider p to receive an observation with value O_k being in state i, and being c a constant to normalize the vector to $\sum_i \pi_k^p(i) = 1$. As previously mentioned, before the first observation (O_1^p), the state probabilities correspond to the steady-state distribution ($\pi_0^p = \pi_{steady}^p$).

To operate with the CT-HMM, which considers CTMCs, *CalcTransientProb* uses CTMC transient analysis equations to calculate the probability of being in state s_i after $t_k^p - t_{k-1}^p$ time units, being the π_{k-1}^p state probability distribution at time t_{k-1}^p.

Figure 3 (right hand side) depicts a supposed software module, we call *CT-HMM Analyzer*, aimed at predicting for a provider the probability distribution of its states. The module stores for each provider the corresponding CT-HMM. When the system advertises

that a service call has finished, this module receives as input information, the name of the provider ($provider_p$), the monitored response time (O_k), and the current time (t_k); then it computes and stores π^p_k. When the system requests for the current π^p_k, the expected probability distribution of the states of a provider, this module computes it using: π^p_{k-1} (that calculated when the last observation of p was received) and t^p_{k-1} (the moment when the observation was received).

3.2 Adaptations based on state predictions

The probability distribution of the states of the providers, π_k, enables service integrators to select providers offering best response time. Figure 3 (left-hand side) depicts a supposed software module, we call *Adaptation Decider*, aiming at deciding system configuration changes. We mean by "configuration change" the replacement of a service provider by another one, indeed offering best response time.

Being the decisions of the *Adaptation Decider* based on predictions, there can happen fails or hits. Fails occur when: 1) the provider has actually changed its state but the prediction does not advise it, we call it "false negative"; 2) the provider has not changed its state but the prediction erroneously advises a change, we call it "false positive". Likewise, a decision hit happens when: 1) no change is advised when it was not needed (called "no-adaptation hit"); 2) or a configuration change is advised when needed (called "adaptation hit"). From the point of view of a self-adaptive software system "false negatives" are lost opportunities to improve system performance, see that a non-adaptive system will always miss such opportunities. However, "false positives" can make the self-adaptive system work worse than a non-adaptive one.

The *Adaptation Decider* calculates the system configuration that is expected to show the lowest mean response time for the system workflow execution. Algorithm 1 describes such calculation: for each possible configuration (line 4) calculates the weighted mean of the mean response time for each state combination of a provider (lines 6-10); the weight of each term is the probability for providers to be in the state expected in that state combination (line 8). Finally, it is selected the configuration that offers the lowest weighted mean response time.

Algorithm 1 has some combinatorial executions (calculation of mrt_i and $Conf_i$) that make it not practicable. The remainder of the subsection discusses three techniques that improve it, we discuss the improvement they achieve and the kind of prediction error they incur.

Most probable state: This technique will change system configuration taking into account for each provider its current most probable state. It will pick the state with lowest response time and consequently selects the provider this state belongs to.

This technique will incur in a large amount of false positives. See for example Figure 5: say the workflow consists of only one service call to $S1$. There are two providers (P_{11} and P_{12}), so the system can execute in two configurations ($Conf1$ and $Conf2$). Each provider can execute in 3 states, s_1, s_2 and s_3, with different mean response times as shown in Figure 5. Initially, the state probability distributions could be: $\pi^{P_{11}}(1) = 0.3, \pi^{P_{11}}(2) = 0.37, \pi^{P_{11}}(3) = 0.33$ and $\pi^{P_{12}}(1) = 0.15, \pi^{P_{12}}(2) = 0.25, \pi^{P_{12}}(3) = 0.6$.

Let us assume the system in $Conf_1$, since the most probable state for P_{11} is s_2, the expected mean response time is 50 tu. Now, let us assume that the *CT-HMM analyzer* calculates a new state distribution for P_{11}: $\pi^{P_{11}}(1) = 0.2, \pi^{P_{11}}(2) = 0.39, \pi^{P_{11}}(3) = 0.41$. So, now the most probable state for P_{11} is s_3 and the system mean response time is 140 tu (see Figure 5). In addition, the most probable state for P_{12} is s_3, whose expected mean response time is 110tu. Since $110 < 140$, the decision will be to change from $Conf_1$ to $Conf_2$. This decision has a very high probability to be a false positive, because if all state probability distributions were had been taken into account, the expected mean response time in $Conf_1$ would have been less than in $Conf_2$, and no reconfiguration would have been proposed (then being a no-adaptation hit). Indeed, applying Algorithm 1, the result would have been: $(0.2 \cdot 10 + 0.39 \cdot 50 + 0.41 \cdot 140) < (0.15 \cdot 20 + 0.25 \cdot 80 + 0.6 \cdot 110)$ which indicates that it is better to remain in $Conf_1$.

This technique is faster than Algorithm 1 since it only looks for one state for each provider (the most probable) and executes a comparison between pre-calculated mean response times. On the other hand, it needs to have pre-calculated and stored the expected mean response times for each provider configuration, which can be costly for large service based systems.

Most probable state with "sureness": This technique still considers the most probable state for each provider i (mps_i), but it also takes into account its probability and the expected improvement in the system response time. The technique calculates a "sureness" value (*Sr*), based on the response time of the system considering source and target configurations ($Conf_s$ and $Conf_t$), as $Sr = \frac{MRT(Conf_t(mps_i))}{MRT(Conf_s(mps_j))}$. Moreover, it calculates a probability $PgoodPred = \pi_i(mps_i) \cdot \pi_j(mps_j)$. The system will change configuration only if $PgoodPred \geq Sr$. Note that when $PgoodPred$ is high -almost one-, the system will reconfigure even when the performance in $Conf_t$ is not very much higher than in $Conf_s$.

This technique executes as fast as the previous one since it also only needs to look for the most probable state probabilities. However, this technique avoids some adaptations that would most likely incur in a false positive. For example, in Figure 5, $Sr = \frac{110}{140} = 0.7857$ and $PgoodPred = 0.41 \cdot 0.6 = 0.246$, then $PgoodPred \ngeq Sr$ and the system would not incur in a false positive adaptation, as it happened in the previous one. The technique also avoids false

Algorithm 1 Algorithm of the *Adaptation Decider*

Require: $AbstractServices(AS), ConcreteProviders(CP)$, $ProvidersStateDistributions\ (\pi_k)$
Ensure: Conf, the configuration with the lowest expected MRT.
1: **set** $Conf_i$ {Configuration, selection of a CP for each AS}
2: **set** $StateComb_{ij}$ {Possible combination of providers states in a Configuration $Conf_i$}
3: **set** $mrt_i = 0.0$ {weighted mean system response time in configuration $Conf_i$ being its providers state distribution π}
4: **for all** $Conf_i \in (AS, CP)$ **do**
5: $mrt_i = 0.0$
6: **for all** $StateComb_{ij} \in Conf_i$ **do**
7: $mrt_{ij} \leftarrow CalculateMRT(StateComb_{ij})$
8: $probState_j \leftarrow CalculateProbability(StateComb_{ij}, \pi_k)$
9: $mrt_i \leftarrow mrt_i + mrt_{ij} \cdot probState_j$
10: **end for**
11: **end for**
12: **return** $Conf_i \mid \forall mrt_{i'}, mrt_i \leq mrt_{i'}$

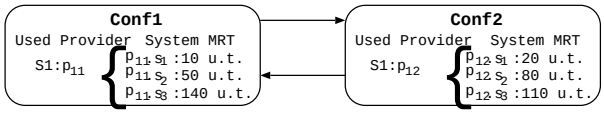

Figure 5: Example of two system configurations

positives that are due to “not as much sure of the most probable state probability as to reconfigure”. However, since it only takes into account the probability of the most probable state, it still incurs in false positives related to “in which states are the probabilities that are not in the most probable state of $Conf_s$ providers”.

Fail compensation: This technique not only takes into account the most probable state, but the whole state distribution. It reduces the complexity of inner loop in Algorithm 1 because (see lines 6-9) it does not calculate mrt_s for each possible combination of states $StateComb_{sj}$ in a source configuration $Conf_s$, however it considers a pre-calculated mean. Concretely, this mean value is pre-calculated for each state of a provider $p_i s_j$ in a configuration, and it represents the mean response time of the system when p_i is in state s_j and considers the steady state distribution for the rest of the providers in $Conf_s$. Therefore, this value represents the mean of the mean response times $mmrt$. Following this technique, the number of loops is $N_{p_i} \cdot N_{p_k}$ instead of $\prod_{p_i \in providers} N_{p_i}$ where N_{p_i} is the number of states of provider p_i.

The technique will cause false positives due to the use of steady state distributions to pre-calculate $mmrt$, whereas in the complete loop in Algorithm 1 the actual state probabilities distributions are considered. To mitigate them, it does not reconfigure just when the expected response time in $Conf_t$ is lower than in $Conf_s$, it also considers the “expected profit" when the decision is a hit or the “performance loss" when the decision is a fail (false positive or negative). Then the adaptation is carried out when $ProfitWhenHit > LossWhenWrong$. $ProfitWhenHit$ is calculated as:

$$\sum_{s_j \in p_i} (\pi_{p_i}(s_j) \cdot \sum_{s_l \in p_k} \pi_{p_k}(s_l) \cdot coeff_{prof}(p_i s_j, p_k s_l))$$

where $coeff_{prof}(p_i s_j, p_k s_l)$ is the function

$$coeff_{prof}(p_i s_j, p_k s_l) = \begin{cases} \frac{mmrt(p_i s_j)}{mmrt(p_k s_l)} & \text{if} \quad \frac{mmrt(p_i s_j)}{mmrt(p_k s_l)} > 1 \\ 0 & otherwise \end{cases}$$

To calculate $LossWhenWrong$ it is also used the previous formula but changing the coefficient for $coeff_{loss}(p_i s_j, p_k s_l)$ where

$$coeff_{loss}(p_i s_j, p_k s_l) = \begin{cases} \frac{mmrt(p_k s_l)}{mmrt(p_i s_j)} & \text{if} \quad \frac{mmrt(p_k s_l)}{mmrt(p_i s_j)} > 1 \\ 0 & otherwise \end{cases}$$

4. ADAPTIVE CONFIGURATIONS: AN ARCHITECTURAL SOLUTION

Kramer and Magee [8] proposed a three-layer reference architecture for self-adaptive systems. In [9], we adapted it to the open-world context, see Figure 6, with the purpose of doing adaptations (configuration changes) targeted to improve the performance of the system. Now, it is our intention to fit proposals in Sections 2 and 3 into this architecture. Subsection 4.1 discusses how to carry it out. In the following we briefly recall the architecture, see [9] for details:

Component Control: This layer senses and reports to its upper layer the *status* of the world where the system executes. It receives adaptation orders to change the current system configuration, i.e., the current service providers.

Change management: When the lower layer reports here the current world *status*, this layer can answer with an order to change the system configuration to a new one which works better regarding both functional and non-functional requirements. To quickly achieve its purpose, it has stored a precomputed set of *plans* or *strategies* to achieve the system goal or *mission*, and it consults them to select a choice. The choice can be either a change in the configuration or to remain in the same one. This layer also stores temporal logs with the information of this type of messages because it also performs periodic studies about providers behavior. This is so since a provider can change its implementation and deployment unpredictably, which will affect its performance. Therefore, this layer tracks the performance of each provider in the long term (e.g., weekly), and periodically checks whether the expected behavior still holds or this knowledge must be updated. In the latter case, it reports the provider performance information to its upper layer. Eventually, it will receive the answer, from the upper layer, with both an appropriate model that represents the current provider performance behavior, and an update of the adaptation strategy that takes into account the new behavior of the provider.

Goal management: This layer studies how the open-world behaves to plan future decisions of change. It manages the system mission producing strategies that takes into account the current world and system workflow. Strategies are produced when the mission changes as well as when the change management layer requests them. Adaptation strategies are not only aware of system functional requirements, but also of performance requirements.

Strategies are created on demand, then, performance requirements can change during runtime to new ones that were not taken into account during design time, and this layer can still produce an ad hoc adaptation strategy for such new requirements.

To generate a strategy, first, the layer uses discovered information about providers as well as measured information from the world that produces knowledge about providers performance behavior. This knowledge is used to analyze the system workflow and to generate a set of feasible system configurations and the suitable changes between them, which indeed is what we are calling a strategy. In each configuration, it also adds information about the expected response times to help its lower layer deciding when to change between configurations (i.e., each configuration indicates a threshold for leaving it). The layer also supports strategy updates, which means, changes in the expected performance values of a configuration or in the structure of the configuration. This can happen for example, when it realizes that a provider has changed its behavior or when a new provider is discovered.

4.1 Integrating the approach in the architecture

Benefits of integrating our proposals into this architecture are clear: we are approaching to a complete reference architecture for open-world software, that can meet performance requirements while manages uncertainties in the environment through a formal model.

Figure 4 appears now embedded within the shadow part of Figure 6, which clearly describes how the new proposal fits in the 3-layer architecture. The *Adaptation Decider* tasks are indeed part of the *Adaptation Manager*, concretely those related to change the system configuration when receives a message notifying providers performance. The *CT-HMM analyzer* matches with the *Provider Performance Predictor*.

Regarding the *Adaptation Manager*, firstly, generates input values of the *Providers Performance Predictor*. In Figure 3, they are `provider`$_p$, `O`$_k$, and `t`$_k$, which now respectively match with information about *Who*, *responseTime* and *When* in Figure 6. Configuration changes are decided using this information, then updating the internal model that stores which one is the main provider for each service and producing a `ReconfOrder` which matches with the *ChangeConfiguration* message of the component control layer.

Regarding the integration of the *CT-HMM analyzer* in the architecture some issues need to be clarified, since it was presented in Section 3 without taking into consideration some challenges in open world. Then, it now should offer additional interfaces: *loadCT-*

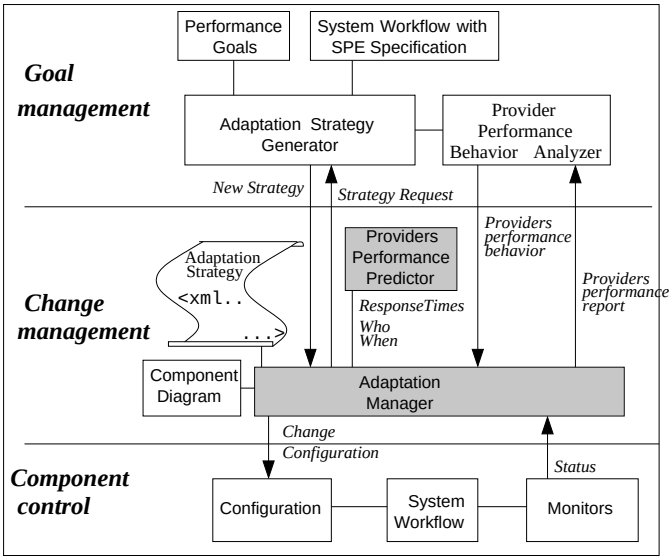

Figure 6: 3-layer architecture adapted to open-world

HMM(model,IDmodel) and *deleteCT-HMM(IDmodel)*. *Adaptation Manager* will use these interfaces when:

- the upper layer *provider performance behavior analyzer* produces a provider performance behavior that was not included in the *Providers Performance Predictor* (*loadCT-HMM*).
- a service provider vanishes off the world (*deleteCT-HMM*).
- knowledge about a service provider behavior is out of date and the *provider performance behavior analyzer* offers an updated model (*deleteCT-HMM* followed by *loadCT-HMM*).

5. RELATED WORK AND CONCLUSION

In [6] authors evaluate selection strategies of providers. Comparison between strategies is based on the mean response time the clients achieve. Although our work shares motivation regarding to reach best workflow performance, we take different assumptions. Firstly, we do not rely on user agreements or collaborations to reach a global knowledge about providers performance behavior and their changes, but we assume independent adaptive clients that only concern about their best performance in a selfish way. Secondly, we assume that our requests do not affect the workload of the providers and hence neither their performance.

In [13] HMMs have been used, in the provider side of self-adaptive software systems, to predict requests based on the monitored history of the behavior of the clients.

Other approaches have been recently proposed for software adaptation, such as [12], which proposes a 3-layered architecture for model-driven adaptation. We share the concept of dynamically generated adaptation plans. Indeed, our previous work in [9] deals with the work of the uppermost layer which is referred to the adaptation plan generation for performance-aware open-world systems. Another key difference is that in our proposal, actions to follow the plans are based on probabilities, since they necessarily come from predictions about the uncertainties in the open-world.

In this work we have presented an approach, based on HMMs, to predict the performance of SOA providers in the open-world. We have also fitted the approach in a 3-layer architecture that is recognized for self-adaptive software systems.

We have to integrate, from the registered information, periodic updates about the knowledge of the providers behavior. Indeed, in the hidden Markov models theory, this is one of the typical studied problems, which means to find out the most probable parameters of an HMM from an observation sequence.

Acknowledgments

Authors would like to thank Javier Campos for fruitful discussions regarding hidden Markov models.

6. REFERENCES

[1] L. Baresi, E. D. Nitto, and C. Ghezzi. Toward open-world software: Issue and challenges. *Computer*, 39(10):36–43, 2006.

[2] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. A framework for QoS-aware binding and re-binding of composite web services. *J. Syst. Softw.*, 81(10):1754–1769, 2008.

[3] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola. QoS-driven runtime adaptation of service oriented architectures. In *ESEC/FSE '09*, pages 131–140, New York, NY, USA, 2009. ACM.

[4] B. H. Cheng et al. Software engineering for self-adaptive systems. chapter Software Engineering for Self-Adaptive Systems: A Research Roadmap, pages 1–26. Springer-Verlag, Berlin, Heidelberg, 2009.

[5] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engg.*, 15(3-4):313–341, 2008.

[6] C. Ghezzi, A. Motta, V. P. L. Manna, and G. Tamburrelli. QoS driven dynamic binding in-the-many. In G. T. Heineman, J. Kofron, and F. Plasil, editors, *QoSA*, volume 6093 of *LNCS*, pages 68–83. Springer, 2010.

[7] Mail service monitor of Universidad de Zaragoza. `https://webmail.unizar.es/mail_monitor.php`.

[8] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *FOSE '07*, pages 259–268, Washington, DC, USA, 2007. IEEE Computer Society.

[9] D. Perez-Palacin, J. Merseguer, and S. Bernardi. Performance aware open-world software in a 3-layer architecture. In *WOSP/SIPEW '10*, pages 49–56, New York, NY, USA, 2010. ACM.

[10] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

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

[12] H. Tajalli, J. Garcia, G. Edwards, and N. Medvidovic. PLASMA: a plan-based layered architecture for software model-driven adaptation. In *ASE '10*, pages 467–476, New York, NY, USA, 2010. ACM.

[13] H. Wang and J. Ying. Toward runtime self-adaptation method in software-intensive systems based on hidden Markov model. *Computer Software and Applications Conference, Annual International*, 2:601–606, 2007.

[14] W. Wei, B. Wang, and D. Towsley. Continuous-time hidden Markov models for network performance evaluation. *Perform. Eval.*, 49(1-4):129–146, 2002.