Mercury: Multi-Agent Adaptive Service Selection Based on Non-Functional Attributes

Michal Jakob
BT Research
jakob@labe.felk.cvut.cz

Alex Healing
BT Research
alex.healing@bt.com

Fabrice Saffre
BT Research
fabrice.saffre@bt.com

Abstract

Service selection in an SOA is primarily based on matching functional requirements to those advertised by the services. The focus of this paper is on selecting, from those functionally capable services, the best service in terms of its non-functional attributes. We propose an adaptive multi-agent framework, Mercury, which involves collaborative modelling of the service landscape based on consumer experience of service quality. The framework introduces several mechanisms to enable the system to be adaptive, and thus effective in highly dynamic scenarios. In particular, different strategies of information exchange between selector agents are investigated with the aim of maximising the speed of adaptation while minimising unfavourable competition between consumers. Combined with adaptive exploitation-exploration control, this allows the selection mechanism to improve the QoS delivered in the presence of strong demand/supply fluctuations and possible service overloading, thus improving the overall operation of an SOA.

1. Introduction

The development and adoption of Service Oriented Architectures (SOAs) [3] has been in part motivated by the promise of enabling a quicker response to dynamic business environments. By wrapping up software components as services they can be used as part of larger processes in a loosely-coupled manner thus allowing disparate information resources to be combined at runtime into larger, more accessible capabilities which users can make use of. Through understanding of the service landscape the service composition process can be, and indeed in many cases may need to be [6], made autonomous. Autonomy in the case of service selection requires the system to know which services in particular are best for a particular user request. This decision is based on both knowing which services are capable of fulfilling the request as well as the quality of the fulfillment which may need to meet some user-specified requirements. There has been a substantial amount of recent work relating to the Semantic Web and Semantic Web Services whereby richer semantics are applied to service descriptions in order for an autonomic service selection mechanism to more easily discern which services are able to process a request based on their functional attributes advertised. There has been less focus, however, on involving non-functional attributes of services for selection decision-making.

We propose an autonomic framework for adaptive service selection, Mercury, focussed on learning through experience which services are best suited for each task based on how well the services perform with relation to their non-functional attributes. It is intended to address the problem of service selection at the level beyond what the functional description language employed by an SOA is capable of. As such, the Mercury framework is to be utilised alongside a functional service selection mechanism to dynamically select the best or near-best service according to the Quality-of-Service (QoS) measured. The system is designed to be adaptive and, so, effective in scenarios where there is contention between services, impacting on the QoS delivered, and when there is a high churn-rate of services. By decentralising the problem of service-selection into a multi-agent system, Mercury draws on knowledge about service providers distributed throughout the network and shared between collaborative selector agents. The focus of the contribution is on the learning algorithm of the selector agents and specifically how their exploration/exploitation strategy is adapted and how experiences are shared between agents.

The paper is structured as follows: Section 2 further discusses the motivation behind incorporating non-functional attributes into service selection and presents some recent efforts to formalise non-functional qualities. Section 3 defines the main components of the framework giving an overview of the architecture. In Section 4, the selection model is defined and the algorithms for updating the model are presented. Section 5 presents validation of the framework components through simulation results. Section 6 summarises the framework and gives an overview of planned future work.

2. Motivation & Related Work

The framework proposed is intended to provide an effective means for dealing with QoS modelling in highly dynamic scenarios. As such the system needs to be adaptive in the case of service providers becoming unavailable and new ones available at any time.
pre-specified compositions of services do not suffice. The framework is intended to be both highly scalable and robust and achieves this through its decentralised approach. Potential application areas include mobile ad-hoc networks (MANETs) and pervasive computing where such a bottom-up, self-organising approach may be particularly attractive. In contrast, the majority of existing systems are designed for static environments where static sets of services can be relied upon.

The majority of related work which addresses the problem of service selection based on QoS by formalising the QoS requirement space. This is often achieved by defining a QoS ontology [5, 10, 11, 12] which is used to specify the qualities that constitute QoS. This is then used for a consumer of a service to specify strict requirements as well as advertising certain quality capabilities from the provider side. Mercury addresses the need for more work dealing with ranking services based on their QoS without explicit QoS requirements as suggested in [9].

The notion of providers advertising their own quality capability, however, brings about the question of trust, which is dealt with by the related works through trust and reputation modelling, in particular [5]. Trust can be used to build relationships between consumers and providers explicitly based on reputation; however the alternative approach taken by Mercury is to build up relationships implicitly based on agent learning dynamics and their interactions with other agents, the details of which make up the main contribution of this work.

A small number of works acknowledge that defining quality requirements and capabilities using precise terms is not always suitable and that instead fuzzy terms of quality may be adopted [2, 11]. It is envisaged that the Mercury framework adopts this approach; however, this is not discussed in this paper and is left for future work (Section 6).

In [12], a multidimensional QoS model similar to that of Mercury is presented, although Mercury goes further to propose a mechanism for which this model can be populated in a collaborative fashion using a multi-agent system.

Much of the work surveyed adopts a decentralised (P2P) architecture combining agents to perform collective modelling. Of particular relevance are [1] and [5]: in the former, a reputation model is built based on peer votes for quality; whilst in the latter, quality ratings are shared via rendezvous nodes in the network. Sonnek et al. [8] has developed and evaluated a task allocation mechanism based on statistical modelling of provider reliability. In contrast to our approach, they use a central reputation server, and they do not consider competition between the clients of the allocation mechanism.

We have previously conducted work using a more theoretical approach whereby relationships with service providers are established based on past experience and simple rules which cause emergent self-organisation of peers [7]. Mercury builds on this work by adding task and context-aware capabilities in the internal model of service selection and by introducing the notion of designated selector agents which may collaborate in order to further improve their selection behaviours.

The main contribution of the framework is described in Section 3.2 and is primarily novel in the way in which multi-agent gossiping strategies aid adaptivity of the system, resulting in a higher level of quality-of-service attainable.

3. Basic Principles

3.1. Definitions

The framework is designed for an SOA and as such assumes a network of interconnected devices, each capable of hosting a number of processes. The processes may adopt at least one of two roles: service provider or consumer. Service providers offer capabilities that other devices (consumers) can access and use.

Mercury-based service-selection takes place on the consumer side and assumes that for every device where there is a service consumer, a selector agent is hosted. The selector agents are used to select services on behalf of consumers residing on the same device (see Figure 1). Selector agents are interconnected forming a Peer-to-Peer (P2P) overlay network which they use for collaboration purposes.

A task is a request for a specific type of processing with details of functional requirements necessary in order for it to be processed, e.g. a task might consist of a requirement for a service to have an Imaging capability but may also contain finer-grained information constraining the task to a certain type of image, of a certain area, and of a certain quality.
Services are software components with well-defined interfaces and can be invoked with a particular task that they can perform. The input to the framework is a task and, using the algorithm detailed in the following section, it selects a service to pass the task to.

On completion of the task, a result is returned to the selector agent and to the user and the Quality-of-Service is derived. Quality-of-Service is an aggregate value of a services performance based on how well it fulfilled a given task.

Mercury may be positioned within an existing SOA and is particularly suited for those which adopt a P2P approach such as [4]. Mercury is not dependent on either the format of the service advertisement or how invocation is carried out. It is instead assumed that some service discovery mechanism (e.g. UDDI based on WSDL service descriptors) is available in the SOA in order to gain a list of functionally capable service providers for a particular task. This functional discovery is based on those attributes that the service providers advertise in their description. The Mercury selector agents then use the list of capable services as a basis for further finer-grained, non-functional selection. This is achieved by aggregating QoS data for each of the providers through the consumer’s experience of them and ranking them accordingly.

The QoS data of providers is stored in a model local to each selector agent and is parameterised by the task, as well as the context. Context is defined as the set of attributes which are external to the task requirements but nevertheless may influence the performance of providers (e.g. performing differently at different times of day). Each service selector therefore builds up a model of how suited each provider is at fulfilling each particular task in each context.

3.2. Contribution

The main contribution is the design of an efficient distributed service selection model and (collaborative) algorithms for its construction and real-time adaptation. Specifically, a decision function is employed to ensure that the probability of exploration is linked to the relative improvement expected when exploration is pursued over exploitation. Here, the term exploration means selecting a provider for which there is none or very limited information about their QoS available, while the term exploitation refers to selecting a provider for which substantial performance data is known, and where the selection algorithm can therefore make a reliable estimate of the QoS the provider is likely to deliver.

An adaptive momentum mechanism for updating the model has been developed so that the incorporation of new data into the model is dependent on the amount and recency of the information already stored. The methods used allow a system of multiple agents to be adaptive to changes in the service environment improving the overall QoS of the system, and may be made more effective through introducing collaborative strategies.

Two collaborative gossiping strategies have been investigated which vary in the degree to which the agents share information. The first strategy involves only partial sharing of information and allows selector agents to gain a better estimation of the distribution of QoS attainable in the network on which the exploration-exploitation control is based. The second collaborative strategy involves sharing detailed information about providers between selectors to speed up learning through exploration. The agents may, however, choose to be selective with the information about providers which they share with others so as not to create unfavourable competition on a subset of service providers, and hence undermine their own performance.

4. Formal Description of Framework

4.1. Problem Space and Preliminaries

The previous section mentioned that the selection model for each selector agent is parameterised by both the task and the context. Let us extend this view to the overall system level whereby the combination of the individual selector agent models can be seen as a distributed model of the global problem space which is defined by the task and the context, but also by the consumer as properties of the consumer itself influence the resultant QoS of a provider. For example, for a given common task in the same context, two consumers residing at different points in the network might have different experiences of the same provider in terms of the resultant QoS they receive for their task. The problem space $P$ is defined formally as:

$$ P = T \times CT \times CS $$

where $T$ is the set of possible tasks, $CT$ the set of possible contexts and $CS$ the set of consumers. Any point in the problem space can then be interpreted as a specific instance of a problem for which a service should be selected.

Clustering of problems within the problem space is key to making the selection model tractable and requires a problem similarity function:

$$ \Delta : P \times P \rightarrow R_+^* $$

where $P$ is the problem space and $R_+^*$ is the set of non-negative real numbers specifying how similar the two problems are. For any two particular problems $p_1$ and $p_2 \in P$, $\Delta(p_1, p_2)$ denotes their similarity, conceptually the inverse of the distance between the two points in the problem space.

The similarity function takes into account how similar each of the constituents of the problems are, and so assumes that the relationship between different tasks,
contexts and consumers is defined beforehand. The similarity between different problems may, however, change on-the-fly depending on which attributes a consumer considers more or less important at any given time.

The selection function $\Omega$ is responsible for making a mapping from the problem space $P$ to the set of services $S$:

$$\Omega: P \rightarrow S \tag{3}$$

Given a problem, i.e. the triple of task, context and consumer attributes, the selection function returns a service which is expected to deliver highest QoS when processing the task.

In practice, however, because the selector agents’ in our framework each have their own local model of the service landscape, which is specific to the subset of consumers for which they broker, they need not include a set of consumers as an input to the selection function as this is implicit. Instead each selector agent has its own consumer-specific selection function $\Omega_{cs}$, where $CS$ is the subset of consumers for which the agent brokers:

$$\Omega_{cs}: T \times CT \rightarrow S \tag{4}$$

By distributing the model of the problem space based on the consumer-specific attributes, the problem space is effectively partitioned into local problem spaces, with each selector agent having responsibility for a single partition.

### 4.2. Selector Agent Model

Let us now look at an example of how a selector agent views its local problem space and the basics of how it builds a selection model.

Each local problem space is parameterised by the attributes of the task and context that impact on the perceived QoS of providers for the consumers that the selector brokers (attributes which don’t affect the QoS need not be modelled). Let us take a simple example with just one task attribute and one context attribute. In this case the local problem space can be visualised as the area defined by the limits of two dimensions – one for each attribute (Figure 2).

Upon selecting a service, a selector agent aggregates the resultant QoS data and stores it in its selection model. This entails recording details of both the task for which the service was used and the context in which it was used in order to associate the QoS data with a particular location in the local problem space. The model, however, doesn’t reference experience based on the exact location in the local problem space but instead clusters similar problems into larger containers called quality registers. The decision whether or not to cluster QoS data relating to a certain problem with already existing data is based on whether or not data of sufficiently similar problems already exists. As discussed in Section 4.1, the problem similarity function (Eq. 2) can be used to discern how similar two problems are and clustering takes place if the level of similarity falls within a certain similarity threshold.

The selection model thus consists of a set of quality registers which are created if the model so far doesn’t contain any data of sufficiently similar problems. The registers are defined by their master problem which is the problem which brought about that particular register’s creation. They store the QoS data for any services which have been selected in the past for use with the problems falling in the space surrounding the master problem, the register’s catchment area. This area is thus defined as that which encompasses problems which are sufficiently similar with relation to the similarity threshold. In the two-dimensional example as in Figure 2, the catchment area of a quality register might be visualised as a neighbourhood surrounding the coordinate of master problem in the local problem space.

Each quality register holds a list of quality records for services that have been selected and invoked in the past on problems from within a register’s catchment area. A quality record contains:

- **Service id** – references the services provider to which the quality record belongs.
- **Expected QoS** – estimate value of QoS when the service is used to process a task from the register’s catchment area.
- **Weight** – specifies how significant the record is, based on the number and recency of invocations of the service referenced.

The records in each register are ranked according to their associated expected QoS. The computation of both the expected QoS and the weighting of each record is described in Section 4.3.

![Figure 2. Local problem space, a QoS Register and problem similarity measure S between two problems: A and B. Problem A corresponds to a task requiring an image with 1cm resolution issued in cloudy weather conditions and shares a similarity S with Problem B. The catchment area of the register is shown in grey.](image-url)
4.3. Selection Algorithm

As mentioned in Section 3.1, an SOA which implements the Mercury framework is assumed to contain some service discovery mechanism capable of returning a list of services functionally compatible with the task, i.e. those that are able to process the task with relation to their functional attributes. This is used as the basis from which Mercury selector agents can more finely derive which particular service to invoke.

The algorithmic schema of the selection process is depicted in Figure 4. When inputted with a problem to select a suitable service to act upon from the resultant list from service discovery, a selector agent needs to decide whether to invoke the best service for which there is already QoS data (exploitation) or to invoke one for which there is either no prior data or one which is thought not to be best (exploration). The decision is probabilistic and is influenced by two main factors:

- The similarity of the current problem to any current quality register’s master problem.
- The contents of the nearest sufficiently similar quality register including the score of the top record and the estimated exploration QoS.

The first factor involves a selector consulting its selection model to determine whether there is a quality register whose master problem is sufficiently similar to the given problem (as specified by the similarity threshold). In the case that no sufficiently similar problem has been experienced in the past then a new quality register is created and exploration ensues with a service being invoked randomly from the list of functionally compatible services. In the case that there is quality data relating to sufficiently similar problems from past experience, the selector agent uses the quality register whose master problem is closest to that of the current problem to calculate the adaptive exploration probability.

Each selector agent maintains a value of estimated exploration QoS by averaging the resultant QoS from past explorations. The exploration probability, used to decide between the exploration and exploitation strategy, is calculated using the difference between the register’s top score and the estimated exploration QoS, i.e., the difference between the mean QoS expected when exploitation is pursued vs. the one expected when exploration is pursued. In doing so, adaptive exploration probability encourages exploration when it is likely to lead to better a QoS than exploitation and vice versa.

Specifically, the exploration probability is calculated as follows. We first define expected relative (QoS) improvement as:

\[ \delta_{rel} = \frac{q_{\text{explore}} - s_{\text{top}}}{s_{\text{top}}} \]  \hspace{1cm} (5)

where \( s_{\text{top}} \) is the QoS score of the top service in the register, and \( q_{\text{explore}} \) the estimated exploration QoS. Exploration probability is then calculated as:

\[ P_{\text{explore}} = \frac{1}{1 + e^{-\beta \delta_{rel}}} \]  \hspace{1cm} (6)

where \( \beta \in (0, +\infty) \) is the exploration sensitivity. When the expected improvement is positive, exploration probability is higher than 0.5 in order to encourage exploration, and vice versa. Exploration Exploitation probability is then simply:

\[ P_{\text{exploit}} = 1 - P_{\text{explore}} \]  \hspace{1cm} (7)

If the exploitation strategy is chosen, the request is forwarded to the top service from the registry and no further reasoning is done by the selector agent. In the case that the exploration strategy is chosen, however, a candidate service for exploration is chosen randomly from all services compliant with the task using the directed exploration distribution function.

The goal of directed exploration is to primarily explore services for which limited or no experience exists. A priority is assigned to each provider so that the selector is most likely to explore those with the least amount of prior experience. The priority \( r_i \) of each candidate is defined as:

\[ r_i = \frac{1}{(1 + w_i)^\gamma} \]  \hspace{1cm} (8)

where \( w_i \) is the weight of the register’s record corresponding to service \( i \), and \( \gamma \in [0, +\infty) \) is the exploration novelty preference. The weight is zero if the service does not have a corresponding record in the register. The exploration probability distribution function is then derived from the distribution of candidate priorities by normalizing it to the sum of one (see Figure 3 for an example of the distribution function).

Figure 3. Exploration candidate distribution function for a scenario involving 20 services and \( \gamma = 1.5 \). For services with IDs 1 to 10 have there are quality records in the register; the weights of these records are 1,\,...,10 respectively. Services 11,\,...,20 do not have records in the register and consequently have the highest probability of being selected for exploration.
4.4. Model Update

Upon selection, the task is dispatched to the service provider and QoS is evaluated. This evaluation may take place autonomically by an agent on the device where the consumer resides or by the consumer themselves (necessary if QoS depends on qualitative measures). A task process record is constructed from the resultant QoS value, the reference to the provider selected to process the task and whether an exploration or an exploitation strategy was employed. The task process record is used to update the selection model. Model updates can take place in two ways: model adaptation or model augmentation.

Model adaptation takes place if the registry relating to the task already contains a quality record for the selected service. The weight of the record is updated as

$$w_{t+1} = w_t + 1$$  \hspace{1cm} (9)$$

where $w_t$ is the existing weight. The expected QoS of the record is updated as

$$q_{t+1} = (1 - \mu) q_t + \mu q$$  \hspace{1cm} (10)$$

where $q_t$ is the existing QoS value for the service, $q$ the value received in the cycle and $\mu$ is the adaptive update momentum calculated as the inverse of record’s weight, i.e.

$$\mu = \frac{1}{w}$$  \hspace{1cm} (11)$$

The higher the momentum, the more difficult it is to modify the existing QoS value. The adaptive update momentum makes updates to the model more significant for quality records of services which have been invoked few times and/or have not been updated in some time and therefore have a higher change of being inaccurate. It addresses two needs that arise with the adaptive selection mechanism, and that cannot be addressed using a fixed momentum update:

1. The selection mechanism needs different update speeds at different times. Low momentum and high update speed is required in the initial, explorative stages of a system’s operation, when new information should have strong impact on existing quality records. Later, however, high momentum is preferable as it maintains the stability of the acquired service selection function. The use of a fixed momentum may result in either slow convergence during the exploration phase (due to momentum being too high) or can lead to oscillations in the exploitation phase (due to momentum being too low).

2. The amount of experience aggregated for each provider is different, and consequently each record needs a different update momentum.

The adaptive momentum mechanism is very important if provider overloading is possible as it allows the selection function to converge to a stable configuration. This is because the selector that uses a particular provider most, has the highest weight for the associated record, and consequently the highest momentum. Let us imagine that when another selector attempts to use the provider it effectively overloads the provider, the (temporarily) low QoS received by both providers has much higher impact on the record held by the “intruding” selector, hence discouraging it from using the provider in the near future. On the contrary, for the “longer-affiliated” selector, the temporary loss of QoS will only marginally affect the record, allowing it to retain its preference.

In order to maintain the ability of the selection system to adapt to changes in the system, the weight of a quality record is gradually decreased if the record is not updated with recent information. The exponential decay law is used in the current implementation of the system. At each time step:

$$w_{t+1} = \alpha \cdot w_t$$  \hspace{1cm} (12)$$

where $\alpha \in (0,1)$ is the quality record forgetting rate.

Model augmentation occurs when the quality register does not contain any relevant quality information pertaining to the problem and service selected and hence needs to be augmented with a new QoS record. Two sub-cases are possible depending on the similarity of the nearest existing register to the problem handled. If the similarity of the nearest register is within the similarity threshold, i.e., if the problem falls in the register’s catchment area, the new QoS record is added to the register. The expected QoS value is set to the QoS of the task processing record and the weight to the initial value of 1. If, however, there is no sufficiently similar existing register then a new register is created at the position of the current problem and the QoS record is

![Service selection algorithm diagram](image-url)
added to it. This process and the difference between both cases above are illustrated in Figure 5.

![Figure 5. Update of the selection model. The problem A is within the catchment area of existing QoS register R1, and the new QoS record obtained when processing the problem is therefore added there. Problem B, on the other hand, does not fall within a catchment area of any existing register and a new register R3 is therefore created to store the record.](image)

4.5. Collaboration between Selector Agents

Gossiping, i.e. sharing experience about providers, is an important element of Mercury. Although not critical for the operation of the selection mechanism, gossiping enables faster convergence and consequently results in a higher average QoS, particularly in situations when the availability of services or their performance varies.

Selectors share experience of providers by exchanging **gossip records** which are derived from the task process records (Section 4.4) created each time a service is selected. We distinguish between two types of gossip records:

- **Full gossip record** containing a full task processing record.
- **Anonymous gossip record** containing a processing record without the provider field.

In the simulated implementation of Mercury described in Section 5, three different levels of gossiping are employed:

- No gossiping – no information is exchanged between selectors.
- Anonymous gossiping – only the information about resultant QoS is communicated (i.e. the provider field of the gossip record is omitted). This allows selectors to get a better understanding of the distribution of QoS.
- Full gossiping – allows the selectors not only to understand the overall distribution of QoS in the network but also to find out what the expected QoS of specific services are.

In our implementation, gossiping takes place each time a service is selected and after the task has been processed, however, delayed strategies whereby data to be shared between agents can be saved and sent in bulk is a trivial alteration.

Anonymous gossiping involves stripping the **Service ID** from the task process record, created from each service invocation, to create the gossip record and broadcasting this **anonymous** record to other selector agents.

Full gossiping copies the entire task process record to a gossip record, allowing the other selectors to gain information about a service provider as if they had experienced it themselves. This significantly speeds up the exploration phase, i.e. amount of exploration per selector is reduced quicker than if there was no gossiping, increasing the likelihood that a good selection is consistently made. Through increased collective exploration, redundant exploration effort is avoided by focussing further exploration on the providers for which limited or no information is available. It also allows providers with low QoS to be collectively identified and avoided so that tasks are less likely to be poorly processed.

Secretive full gossiping works much the same way as full gossiping, sharing all data fields in the task process record, with the exception that an anonymous gossip record is created for the provider which is deemed best, i.e. after the model is updated, it is the provider at the top of the ranked list of quality records. This can be useful when there can be contention of providers between consumers. By keeping the most ‘exploitable’ provider secret from other selector agents, the overloading problem is alleviated and results in a more diverse selection function across different providers.

The way in which a gossip record is processed when received by a selector depends on:

- whether it is full or anonymous
- whether or not it is the result of exploration on the part of the gossiper (an **exploration gossip record**)

Gossip records from exploration (both full and anonymous) are used to update selectors’ estimates of the average exploration QoS. Full gossip records (from both exploration and exploitation) are used to update selectors’ quality registers.

Each selector maintains a time-limited queue of all exploration gossip records it has received in the past. It then calculates its estimate $\hat{q}_{\text{explore}}$ of system average exploration QoS as the average of the QoS values in the gossip records stored on its exploration gossip queue.

In addition to the gossip queue, each register also maintains a time-limited queue of processing records concerning tasks it handled in the past. Using this queue,
the selector can calculate a subjective estimate $\hat{q}^{subj}_{\text{explore}}$ of average exploration QoS based solely on its own experience.

The aggregate estimate of exploration QoS (used in deciding between exploration and exploitation strategy, Section 4.3), is then calculated as

$$q^{\text{explore}} = \delta \hat{q}^{subj}_{\text{explore}} + (1 - \delta)q^{\text{explore}}$$

(13)

where $\delta \in [0,1]$ is the exploration subjectivity coefficient, reflecting how much the average exploration QoS depends on either personal experience or that gained from other selector agents. In the current implementation, we use $\delta = 0.5$, i.e. both the selector’s individual experience and the experience of other selectors are considered equally important.

Full gossips are equivalent in their information content to task processing records obtained by selectors themselves. They are consequently used to update a selector’s register in exactly the same way as described in Section 4.4 for updates based on task processing records.

The whole task processing cycle, combining the steps described above, is depicted in Figure 6.

\[\text{Figure 6. Task processing cycle diagram.}\]

**5. Experimental Analysis**

In order to quantitatively compare the main features of the Mercury framework a simulation environment was developed which can be populated with $n$ providers of a single service and $m$ service selector agents. We abstract away from the notion of consumers in this case and assume that both the task and context parameters of the problem stay constant.

We were particularly interested in investigating the effectiveness of the system in the case where QoS of a particular service degrades depending on how many simultaneous connections there are to it at any one time. In this sense there is competition for resources and in order to reach an optimal configuration of service selection, it is necessary for the selector agents to both be able to form relationships with certain providers whilst remaining adaptive to changes. In the simulation, the environment is dynamic in the sense that resultant QoS is non-deterministic from an individual selector’s point of view due to competition and the distribution of QoS capability can be parameterised.

For all of our experiments the simulation was set up with the following parameters:

- 30 Providers
- 5 Selector agents
- Exploration sensitivity (Eq. 6) = 2
- Exploration subjectivity (Eq. 13) = 0.8

The QoS capability distribution was set to uniformly increase such that the 1st provider had the minimum capability and the 30th provider had the maximum (zero and one, respectively). At each time step in the simulation, each selector agent chooses a provider to be invoked and receives the measure of QoS from the provider as a result. The internal selection model is built up through subsequent time steps and at the end of each time step, each selector agent may gossip with all other selector agents, depending on their gossiping strategy. The results are averaged over 10 runs.

The first set of experiments was used to compare the different selector agent collaboration strategies on the resultant system (global) QoS attained (Figure 7). It is clear that gossiping enables the QoS to be increased faster and rather unsurprisingly full gossiping produces the fastest rate of QoS increase through the initial stages. The full gossiping approach would be highly effective if at some point the service landscape were to change dramatically. With little or no provider churn, though, full gossiping actually results in a lower QoS than if there was no communication. This demonstrates how, by sharing information about the ‘best’ provider with other agents results in unfavourable competition whereby relationships between a selector $S_i$ and a particular provider $P$ becomes infected by another selector $S_j$ which has gained information about $P$ from $S_i$ and so believes that such a relationship is best for it too. In this case, the global QoS actually decreases. As discussed in Section 4.5, secretive gossiping aims to counteract this effect by not sharing ‘best’ providers between selector agents. Indeed, Figure 7 indicates that the resultant QoS is highest when using the secretive gossiping strategy. A slight lag compared to the full gossiping curve can be seen and this represents the trade-off of not sharing with other agents the top provider.

The anonymous gossiping strategy clearly also proved to be very good but elicits slower convergence which demonstrates that there is a case for sharing direct references to providers such as in the full and secretive strategies. Nevertheless, its effectiveness highlights the
importance of collaborating to improve the data on which
the exploration/exploitation decision is based.

![Figure 7. Effect of different selector agent
 collaboration strategies on resultant system QoS.](image)

Communication methods comparison

<table>
<thead>
<tr>
<th>QoS</th>
<th>Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>no communication</td>
<td>1 3 5 7 9 11 13 15 17 19 21 23</td>
</tr>
<tr>
<td>anonymous gossiping</td>
<td>1 3 5 7 9 11 13 15 17 19 21 23</td>
</tr>
<tr>
<td>full gossiping</td>
<td>1 3 5 7 9 11 13 15 17 19 21 23</td>
</tr>
<tr>
<td>secretive gossiping</td>
<td>1 3 5 7 9 11 13 15 17 19 21 23</td>
</tr>
</tbody>
</table>

![Figure 8. Comparison of fixed vs. Mercury adaptive
eploration probability mechanisms.](image)

Exploration-Exploitation control logic

<table>
<thead>
<tr>
<th>QoS</th>
<th>Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed (0.2)</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20</td>
</tr>
<tr>
<td>fixed (0.5)</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20</td>
</tr>
<tr>
<td>adaptive</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20</td>
</tr>
</tbody>
</table>

6. Conclusions and Future Work

The Mercury framework is a concrete illustration of
how emergent properties can be leveraged to improve
global system behaviour in Service-Oriented
Architectures. The combination of local decision-making
(exploration/exploitation strategy) with diffusion of QoS
information (gossiping) allows a population of selectors
with variable needs to collectively identify and converge
toward a configuration that meets the requirements of a
majority of participants. Moreover, this distributed
problem-solving is largely implicit: the establishment of
preferential relationships between selectors and providers
incorporates any bias associated with initial conditions
and/or the influence of the early history of the system. For
instance, in the case that there is competition between two
or more selectors for a contended resource, the progressive
gain of momentum will ensure that random fluctuations
are amplified to the point where only an adequate subset
of all competing selectors keep their affiliation with the
service. By forcing the ‘losers’ to identify an alternative
provider, this process usually leads to improved global
QoS, without any need for central planning or explicit
negotiations between selectors.

Furthermore, since QoS is constantly re-evaluated,
Mercury is capable of detecting and adapting to changing
circumstances, whether they affect the service consumer
(e.g. new requirements), the provider (e.g. change of
context) or the relationship between them (e.g. bandwidth
shortage). This effectively means that when the existing
web of selector-provider relationships is no longer
adequate, the system can self-organise into a new stable
state reflecting the changing conditions. Depending on the
severity of the perturbation and/or on the presence of
strong coupling (e.g. intense competition for services),
the process can lead to a cascading reorganisation or, on the
contrary, be confined to a small region of the system.
Most critically though, this global plasticity is achieved
without any modification to the selector agents’
behavioural repertoire (there is no explicit ‘emergency
response mode’). So, by any practical definition, the
spontaneous adjustment to changing conditions is an
emergent property of the whole system, mediated
exclusively by local decision-making.

Future work will look at minimising overhead from the
gossiping mechanism, as well as addressing the possibility
that QoS may be consumer-relative. The notion of fuzzy
requirement matching, whereby consumers can assign
priorities to certain non-functional attributes, will also be
formalised.
7. References


[2] C.- L. Huang, C.- C Lo, Y. Li, K.- M Chao, J.- Y Chung and Y Huang, ‘Service Discovery through Multi-Agent Consensus’


