# Optimizing Decisions in Web Services Orchestrations

Ajay Kattepur, Albert Benveniste, Claude Jard

▶ **To cite this version:**

Ajay Kattepur, Albert Benveniste, Claude Jard. Optimizing Decisions in Web Services Orchestrations. 9th International Conference on Service-Oriented Computing, Dec 2011, Paphos, Cyprus. 10.1007/978-3-642-25535-9_6 . hal-00650313

**HAL Id: hal-00650313**
**https://inria.hal.science/hal-00650313**

Submitted on 9 Dec 2011

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Optimizing Decisions in Web Services Orchestrations

Ajay Kattepur[1], Albert Benveniste[1] and Claude Jard[2]

[1]IRISA/INRIA, Campus Universitaire de Beaulieu, Rennes, France.
[2]ENS Cachan, IRISA, Université Européenne de Bretagne, Bruz, France.

**Abstract.** Web services orchestrations conventionally employ exhaustive comparison of runtime quality of service (QoS) metrics for decision making. The ability to incorporate more complex mathematical packages are needed, especially in case of workflows for resource allocation and queuing systems. By modeling such optimization routines as service calls within orchestration specifications, techniques such as linear programming can be conveniently invoked by non-specialist workflow designers. Leveraging on previously developed QoS theory, we propose the use of a high-level flexible query procedure for embedding optimizations in languages such as Orc. The *Optima* site provides an extension to the sorting and pruning operations currently employed in Orc. Further, the lack of an objective technique for consolidating QoS metrics is a problem in identifying suitable cost functions. We employ the *analytical hierarchy process (AHP)* to generate a total ordering of QoS metrics across various domains. With constructs for ensuring *consistency* over subjective judgements, the AHP provides a suitable technique for producing objective cost functions. Using the *Dell Supply Chain* example, we demonstrate the feasibility of decision making through optimization routines, specially when the control flow is QoS dependent.

**Keywords:** Web Services, QoS, Optimization, Orc, AHP.

## 1 Introduction

A *composite* web service is an application whose implementation calls other self-contained *atomic* services. A composite web service *orchestration* specifies the interaction, management and coordination between these atomic services. Such a composite service can take decisions to invoke or pass parameters to atomic services depending on returned data and quality of service (QoS) metrics. Traditional orchestrations make use of simple comparisons of returned values from atomic services for decision making purposes. While such comparisons are plausible in small orchestrations, involved operations such as multi-criteria decisions from a directory of hundreds of distributed services would require optimizations strategies. With QoS metrics modeled as random variables [1], the use of probabilistic contracts for service level agreements (SLAs) [2] becomes mandatory. Optimizing these random variables for decision making is a natural extension of the probabilistic nature of both composition as well as contracts.

As switching between technologies while developing workflows is detrimental, integration of optimization techniques as part of the specifications of a service orchestration or choreography is required. We show that optimization of QoS metrics can be formulated within concurrent programming languages like Orc [11]. Employing specialized *sites* that perform optimization routines, alternatives to conventional sorting and searching techniques may be incorporated within workflow specifications.

As the designers of such workflows are assumed to be non-specialists in optimization modeling, we propose techniques for formulating complex queries through simple user judgements / constraints. This will relieve the dependency on domain-specific and involved concepts such as queuing and process management theory in order to generate realistic cost functions. Weighing parameters effectively is done by employing the analytic hierarchy process (AHP) [4]. It provides a simple approach for retaining consistency of subjective evaluations of QoS metrics across different domains.

To prevent deadlock in an orchestration where there are intricate links between parameters, it is essential that optimal settings are employed. This is demonstrated in the QoS dependent choreography of the *Dell* supply chain example [9]. By modeling this choreography as a *linear programming* problem, we demonstrate the efficacy of our technique to ensure contractual obligations with shared resources. Due to the tractable nature of AHP, cost functions can be generated to set suitable resupply batch sizes for varying demand rates. This exemplifies clearly a situation where the control flow is dependent on optimal setting of parameters.

The paper is organized as follows: Section 2 presents background material required for understanding the rest of the paper. This includes optimization models, Orc language for orchestrations, the analytic hierarchy process and QoS aspects of web services. The methodology proposed in this paper is outlined in Section 3 with emphasis on formulating optimizations in web services. Section 4 elucidates the Dell logistics example as an optimization of QoS metrics. Extending this notion to general orchestration problems, in Section 5, we formulate a general site that provides such optimization routines in the Orc context. Results for optimization runs of both examples are presented in Section 6. This is followed by related work and conclusions in Sections 7 and 8, respectively.

## 2 Fundamentals

### 2.1 Optimization models

Optimization problems may be formulated as [6]:

$$\begin{aligned} &\mathbf{min} \quad f_0(a,x) \\ &s.t. \quad f_i(a,x) \leq 0, \quad i = 1,\ldots,m \end{aligned} \tag{1}$$

where $f_0$ is the objective function, $f_i$ are the set of constraint functions dependent on the input vector $x = (x_1, x_2, \ldots, x_N)^T$ and model parameters $a =$

$(a_1, a_2, \ldots, a_M)^T$. This can be solved in a variety of linear, non-linear, stochastic and exhaustive search techniques. Approximate bounds to reduce stochastic uncertainty can also be used. This can lead to three categories of minimization problems.

– Minimization of primary expected costs subject to secondary cost constraints.

$$\begin{aligned} &\textbf{min} \;\; F_0(a,x) \\ &s.t. \;\; F_i(a,x) \leq F_i^{max}, \;\; i = 1, \ldots, m \end{aligned} \tag{2}$$

where $F_0(a,x)$ is the primary goal, $F_i(a,x)$ are secondary constraints with worst-case bounds represented by $F_i^{max}$.

– Minimization of the cost function with positive weights $k_0, k_1, \ldots, k_m$.

$$\textbf{min} \;\; \sum_{i=0}^{m} k_i F_i(a,x) \tag{3}$$

– Minimization of the maximum weighted expected costs.

$$\textbf{min} \;\; \max_{0 \leq i \leq m} k_i F_i(a,x) \tag{4}$$

Such formulations of cost functions with constraints can be applied to a variety of decisions within the web services framework.

### 2.2 QoS in Web Services

Available literature on industry standards in QoS [3] provide a family of QoS metrics that are needed to specify SLAs. These can be subsumed into the following four general QoS observations [1]:

1. $\delta \in \mathbb{R}_+$ is the service latency. When represented as a distribution, this can subsume other metrics such as availability and reliability of the service.
2. $\$ \in \mathbb{R}_+$ is the per invocation service cost.
3. $\zeta \in \mathbb{D}_\zeta$ is the output data quality. This can represent other metrics such as data security level and non-repudiation of private data over a scale of values.
4. $\lambda \in \mathbb{R}_+$ is the inter-query interval, equivalent to considering the query rate for a service. Performance of the service will depend on negotiations with the amount of queries that can be made in a given time interval.

Along with QoS, the web service performs its task and returns some functional data $\rho \in \mathbb{D}_\rho$ as the output. The tuple of *(Data value, QoS value)* is used for the decision process within orchestrations. The implementation of Orc allows such typing to be specified for input and output parameters, which can be extended to QoS typing for orchestrations.

[1] Aspects such as scalability, interoperability and robustness are not dealt with as they are specific to the supplier side operation (not necessarily part of SLAs).

For comparing metrics with differing scales and units of measurement, a normalization or scaling technique is needed. As developed in [5] [16], the normalization of QoS values $\mathbf{q}_i$ in a domain $\mathbb{D}_Q$ can be performed using a scaling function, prior to optimization. The scaling function $S(\mathbf{q}_i)$ in eq. (5) ensures that the range of QoS values falls within $[0, 1]$ for equivalent comparison. Essentially, this prevents larger scale values in domains (eg. latency) nullifying optimal selection in smaller valued domains (eg. boolean valued availability).

$$S(\mathbf{q}_i) = \frac{\mathbf{q}_i - \mathbf{q}_{min}}{\mathbf{q}_{max} - \mathbf{q}_{min}} \tag{5}$$

where $\mathbf{q}_{min}$ and $\mathbf{q}_{max}$ are the minima and maxima of the (available) distributions of these QoS domains. A generic range of values for metrics such as data quality or service invocation costs may be reduced to a comparable scales via this method. An example of scaling measured values is shown in Table 1. The measured values are scaled to the range $[0, 1]$ with the scaling invariant to changes in measurement units of, for instance, the response time $\delta$.

| **Metric** | **Measurement** $\mathbf{q}_i$ | **Scaled Value** $S(\mathbf{q}_i)$ |
|---|---|---|
| $\delta(hours)$ | (0.017, 0.001, 0.0095, 0.01) | (1, 0, 0.53125, 0.5625) |
| $\delta(seconds)$ | (61.2, 3.6, 34.2, 36) | (1, 0, 0.53125, 0.5625) |
| $\$(Euros)$ | (9.5, 3.4, 6.8, 12) | (0.7093, 0, 0.3953, 1) |
| $\zeta([1, 10])$ | (6, 1, 3, 8) | (0.7143, 0, 0.2857, 1) |

**Table 1.** Scaling QoS metrics across domains to the range [0,1].

### 2.3 Analytic Hierarchy Process

Multiple dimensions in web services' QoS are only partially ordered, with comparisons between domains not possible. In order to use optimization routines, a total ordering of these domains is mandatory. To reconcile this, the analytic hierarchy process (AHP) can be used. Introduced by [4], AHP can be used to objectify subjective evaluations of multi-criteria decisions, which essentially develops tradeoffs between domains. In order to briefly explain the AHP, we make use of an example.

Consider the pairwise assignment of relative ranks for QoS metrics as defined by a user. It is a matrix that defines the relative change between dependent QoS metrics $\delta$, \$, $\zeta$, $\lambda$ and $\rho$. For simplicity, all parameters are classified as the same hierarchical level with values assigned using the relative comparison shown in Fig. 1. This in turn will produce a matrix $\mathbf{W} = (\mathbf{w}_{ij})$ as shown in eq. 6 with the subjective pairwise comparison of criterion.

$$\mathbf{W} = \begin{array}{c} \\ \delta \\ \$ \\ \zeta \\ \lambda \\ \rho \end{array} \begin{array}{c} \begin{array}{ccccc} \delta & \$ & \zeta & \lambda & \rho \end{array} \\ \begin{pmatrix} 1 & 1 & 5 & 3 & 5 \\ 1 & 1 & 5 & 3 & 5 \\ 1/5 & 1/5 & 1 & 1 & 2 \\ 1/3 & 1/3 & 1 & 1 & 3 \\ 1/5 & 1/5 & 1/2 & 1/3 & 1 \end{pmatrix} \end{array} \tag{6}$$

| The Fundamental Scale for Pairwise Comparisons | | |
|---|---|---|
| **Intensity of Importance** | **Definition** | **Explanation** |
| 1 | Equal importance | Two elements contribute equally to the objective |
| 3 | Moderate importance | Experience and judgment slightly favor one element over another |
| 5 | Strong importance | Experience and judgment strongly favor one element over another |
| 7 | Very strong importance | One element is favored very strongly over another; its dominance is demonstrated in practice |
| 9 | Extreme importance | The evidence favoring one element over another is of the highest possible order of affirmation |
| Intensities of 2, 4, 6, and 8 can be used to express intermediate values. Intensities 1.1, 1.2, 1.3, etc. can be used for elements that are very close in importance. | | |

**Fig. 1.** Comparison Scale for AHP [4].

The principal eigenvector of the *positive reciprocal matrix* $\mathbf{W}$ provides the relative rankings of the parameters. As the principal diagonal of the matrix $\mathbf{W}$ consists of real values, the principal eigenvector (and corresponding highest eigenvalue) are also real valued.

**Theorem 1** ***Perron Frobenius Theorem****: For a given positive matrix* $\mathbf{W}$*, the only positive vector* $v$ *and only positive constant* $c$ *that satisfy* $\mathbf{W}v = cv$*, is a vector* $v$ *that is a positive multiple of the principle eigenvector of* $\mathbf{W}$ *and the only such* $c$ *is the principal eigenvalue of* $\mathbf{W}$*.*

This eigenvector may be normalized to provide the *priority vector* for the QoS metrics. This will generate a weighted cost function for minimization, which is superior to cost function weights obtained by least squares [8]. For the example above, the linear cost function after generating the normalized weight vector is shown in eq. (7) with scaling of values done previously according to eq. (5).

$$\mathbf{Z} = 0.3625\delta + 0.3625\$ + 0.0935\zeta + 0.1237\lambda + 0.0579\rho \tag{7}$$

A unique feature of the AHP is its ability to estimate consistency in the subjective evaluation of criteria.

**Definition 1** *A* $n \times n$ *positive reciprocal matrix* $\mathbf{W} = (\mathbf{w}_{ij})$ *is a* ***Consistent Matrix****, if the highest eigenvalue* $c_{max}$ *equals* $n$*. This is equivalent to* $\mathbf{w}_{ij} = v_i/v_j$*, where the eigenvector* $v$ *corresponds to eigenvalue* $c_{max}$*. Since small changes in* $\mathbf{w}_{ij}$ *imply changes in* $c_{max}$*, the deviation from* $n$ *is a deviation from consistency given by* $(c_{max} - n)/(n - 1)$ *which is called the* ***consistency index (CI)****.*

This technique evaluates the perturbation in the highest eigenvalue due to changes in subjective evaluation of metrics in $\mathbf{W}$. The values of the consistency index are used to generate a consistency ratio (CR), that is used to determine the consistency of the comparison. The consistency ratio must be $\leq 0.1$, indicating deviations from subjective evaluations are less than an order of magnitude [4]. For the example above, the highest eigenvalue has the value 5.122, producing a CI = 0.0280 and a CR = 0.0252, which is within the specified limits. Techniques outlined in [8] provide steps and tools to improve consistency in the weight matrix.

## 3 Methodology

The following steps are used to solve optimization problems in web services:

1. **Scaling Inputs**: Obtain the pair of QoS domains and vector of values $(\mathbb{D}_Q, \mathbf{q})$ required for evaluation of the orchestration. For each domain $\mathbb{D}_Q$, scale the values $\mathbf{q}$ to the range $[0, 1]$ as specified in eq. (5).
2. **Consistent Judgements**: Extract the comparative judgement matrix $\mathbf{W} = (\mathbf{w}_{ij})$ from the user. From this, obtain the maximum eigenvalue $c_{max}$ and the corresponding normalized eigenvector $\upsilon$. If this judgement matrix is not *consistent*, examine the judgment for an entry $\mathbf{w}_{ij}$ for which $\mathbf{w}_{ij}\upsilon_j/\upsilon_i$ is the largest, and see if this entry can reasonably be made smaller. Such a change of $\mathbf{w}_{ij}$ also produces a new comparison matrix with a smaller eigenvalue, resulting in a possibly consistent matrix [8]. This process may be performed either manually or automatically through iterative perturbations of $\mathbf{W}$ until consistency is achieved. Once a consistent matrix is obtained, the objective function $\mathbf{Z}$ to be minimized with linear weights $\upsilon$ and $(\mathbb{D}_Q, \mathbf{q})$ values may be generated.
3. **Constraints**: The scaled optimization constraints $\mathbf{C}$ in the form $(\mathbb{D}_Q, \preceq, K_Q)$, where $\mathbb{D}_Q$ is a QoS domain, $\preceq$ is a specified partial order and $K_Q$ is the threshold value (constant or distribution quantiles), may also be set by the user.
4. **Optimization**: With a selected constraint satisfying solver with inputs $(\mathbf{Z}, \mathbf{C})$, optimization is performed. If constraints $\sum_{i=1}^{N} x_i = 1, x_i \in \{0, 1\}$ for model variables $x$ exists in $\mathbf{C}$, it implies an integer programming problem (eg. selecting a single site). In the absence of such a constraint, the solver employs a conventional linear programming approach (eg. finding an optimal setting from a continuous distribution).

The only inputs required from the user are the judgement matrix and constraints over QoS domains. This methodology is intended to enhance previous theory [7] with optimization routines to compare returned QoS token values.

## 4 Formulating Optimization Problems

In this section, we investigate the Dell supply chain, a choreography of *Dell Plant* and *Supply* orchestrations with a shared *Revolver* resource. This exemplifies the optimization of setting inventory levels to ensure efficient control flow and preventing contractual deviations.

In the *Dell* supply chain [9], QoS metrics are functional in nature, with slight changes in optimal settings sending the supply chain to a dead state. The *Dell* application is a system that processes orders from customers interacting with the Dell webstore. According to [9], this consists of the following prominent entities:

– *Dell Plant* - Receives the orders from the Dell webstore and is responsible for the assembly of the components. For this they interact with the *Revolvers* to procure the required items.

- *Revolvers* - Warehouses belonging to Dell which are stocked by the suppliers. Though Dell owns the revolvers, the inventory is owned and managed by the *Suppliers* to meet the demands of the *Dell Plant*.
- *Suppliers* - They produce the components that are sent to the revolvers at Dell. Periodic polling of the *Revolvers* ensures estimates of inventory levels and their decrements.

The interaction between the *Dell Plant*, *Revolvers* and the *Suppliers* may be summarized in Fig. 2. The requests made by the plant for certain items will be favorably replied to if the revolvers have enough stock. This stocking of the revolvers is done independently by the suppliers. The suppliers periodically poll (withdraw inventory levels) from the revolvers to estimate the stock level. In such a case, a contract can be made on the levels of stock that must be maintained in the revolver. The customer side agreement limits the throughput rate. The supplier side agreement ensures constant refueling of inventory levels, which in turn ensures that the delay time for the customer is minimized. Thus, it represents a *choreography* comprising two plant-side and supplier-side orchestrations interacting via the revolver as a shared resource.

The critical aspect in the Dell choreography is efficient management of revolver levels. As discussed in [9], for the efficient working of the supply chain, the interaction between the Dell Plant and the Supply-side workflows should be taken into account. This will involve optimizing critical QoS metrics listed in Table 2. They are also presented informally in Fig. 2.

| | |
|---|---|
| $t$ | Unit of time with $t \in 1, 2, ...T$ hours |
| $\lambda_t$ | Number of queries per unit time that the plant requests the revolver |
| $\delta_{cust}$ | Waiting time for the plant |
| $\mu_t$ | Stock level for an item in the revolver at time $t$ |
| $\mu_c$ | Critical stock levels of the item in the revolver |
| $\mu_{max}$ | Maximum stock level allowed in the revolver |
| $\rho$ | Inventory polling period of the supplier |
| $\beta$ | Size of the refueling batch from the supplier |
| $\delta_{sup}$ | Delay period for refueling the revolver |
| $\upsilon_{\mu_c}, ...\upsilon_\beta$ | Normalized eigenvector from the consistent AHP matrix |

**Table 2.** QoS Metrics for the Dell Supply Chain.

For the plant-side behavior, the demand $\lambda_t$ reduces the current revolver level ($\mu_t = \mu_{t-1} - \lambda_t$). Constant polling at a rate $\rho$ ensures the re-fueling of revolver inventory within a supply delay $\delta_{sup}$. When the value of the revolver token drops below a critical level $\mu_c$, the supplier begins the process of refueling the inventory. The refueling batch size $\beta$ is governed by the maximal capacity of the revolver $\mu_{max}$. Optimal setting of these parameters minimizes the customer waiting time $\delta_{cust}$. If the supplier does not refuel on time, the choreography sets into deadlock with the plant waiting for (possible) restocking. A deadlock occurs

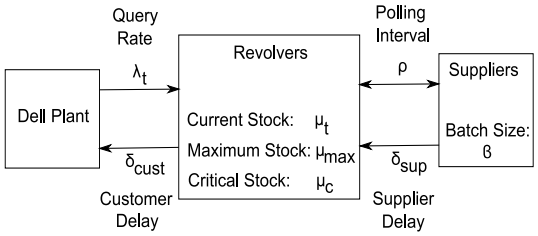


**Fig. 2.** QoS interactions in the Dell supply chain.

when a choreography reaches a state that (1) is not final and (2) can not be left without violating the message ordering of the choreography.

Considering estimated distributions of customer demand and refueling delays, effective settings for supplies may be set. Using AHP weights, the optimization procedure is given as a linear programming problem (without any integer constraints). More classical logistics cost functions [10] can also be applied to similar problems.

$$\textbf{minimize } \mathbf{Z} = \upsilon_{\mu_c}\mu_c + \upsilon_{\mu_{max}}\mu_{max} + \upsilon_{\beta}\beta \tag{8}$$

Subject to the following user-specified constraints:

$$0 \leq \mu_c \leq \mu_{max} \tag{9}$$

$$0 \leq \beta \leq \mu_{max} \tag{10}$$

$$\mu_{max} - \mu_c + (\lambda_t \times \delta_{sup}) = \beta \tag{11}$$

$$0 \leq \mu_{max} \leq K \times \lambda_t \tag{12}$$

Constraints in eqs. (9) and (10) limit the revolver critical level $\mu_c$ and the supplier batch size $\beta$ to be less than the maximal revolver capacity $\mu_{max}$. The constraint in eq. (11) essentially controls the revolver batch size, dependent on the critical / maximum level in the revolver and the plant query rate $\lambda_t$. Estimates of $\lambda_t$ are provided to the supplier during the polling period through measured decrements in the revolver levels. The supplied batch $\beta$ also incorporates the decrement in inventory since the critical level was detected, and the delay in restocking $\delta_{sup}$. Finally, the constraint in eq. (12) prevents overstocking of items in the revolver by limiting the capacity to be proportional to the demand. Optimal setting of these parameters is tested by the constant demand for products $\lambda_t$ which must be delived while minimizing the customer delay $\delta_{cust}$. Essentially, these constraints ensure the revolver level does not fall to zero, which would mean rejection or long delays in orders (deadlock in the choreography).

## 5 Optimization Routines in Orc

While the previous sections demonstrate the utility of optimization techniques when applied to decisions in workflows, it is imperative to provide a convenient

technique to embed such mathematical packages within orchestrations. Extending Orc [11] with a suitable interface will enable smooth integration of optimization libraries for the utility of workflow designers. In this section, we provide a high-level specification of optimizing QoS metrics within Orc.

Orc [11] serves as a simple yet powerful concurrent programming language to describe web services orchestrations. The fundamental declaration used in the Orc language is a *site*. The type of a *site* is itself treated like a service - it is passed the types of its arguments, and responds with a return type for those arguments. An Orc *expression* represents an execution and may call external services to publish some number of values (possibly zero).

Orc has the following combinators that are used on various examples as seen in [11]. The *Parallel* combinator $X|Y$, where $X$ and $Y$ are Orc expressions, runs by executing $X$ and $Y$ concurrently; returns from $X$ and $Y$ are interleaved. Whenever $X$ or $Y$ communicates with a service or publishes a value, $X|Y$ does so as well. The execution of the *Sequential* combinator $X >t> Y$ starts by executing $X$. Sequential operators may also be written compactly as $X \gg Y$. Values published by copies of $Y$ are published by the whole expression, but the values published by $X$ are not published by the whole expression; they are consumed by the variable binding. If there is no response from either of the sites, the expression does not terminate. The *Pruning* combinator, written $X <t< Y$, allows us to block a computation waiting for a result, or terminate a computation. The execution of $X <t< Y$ starts by executing $X$ and $Y$ in parallel. Whenever $X$ publishes a value, that value is published by the entire execution. When $Y$ publishes its first value, that value is bound to $t$ in $X$, with the execution of $Y$ immediately terminated. The *Otherwise* combinator, written $X; Y$ has the following execution. First, $X$ is executed. If $X$ completes, and has not published any values, then $Y$ executes. If $X$ did publish one or more values, then $Y$ is ignored. The publications of $X; Y$ are those of $X$ if $X$ publishes, or those of $Y$ otherwise.

Consider the following two Orc expressions - one of which chooses the fastest responding service; another produces the lowest costing service value:

```
def minLatencySite() = s <s< (Site_1 |...| Site_N)
def minCostSite() = (Site_1,...,Site_N) >(c_1,...c_N)> minimum([c_1,...c_N])
```

Combining these expressions in Orc can currently be done with priorities, that is, choosing a site with lower cost over one with lower latency, or vice versa. This can be detrimental in typical situations involving more than one QoS metric. Finding an optimal service that provides a "middle path" solution from various domains can be beneficial. Such an expression in Orc with weights `w`:

```
def optimalSite() = (Site_1,...Site_N) >((d_1,c_1),...,(d_N,c_N))>
minimum([ w*d_1 + (1-w)*c_1 ,..., w*d_N + (1-w)*c_N ])
```

A drawback of the above formulation is that exhaustive comparison of metrics are still used. In order to overcome this, the selection of services can be

formulated as an optimization problem. Such a formulation is useful in a variety of orchestrations where the control flow is dependent on optimal resolution of competition between services. A point to note here is that the fastest service cannot be given priority as the orchestration waits for responses from all services (until timeout).

In [7], the "best" operator provides a general function for comparison of a variety of metrics. We propose an extension of this to satisfy more complex queries, when "enumerate and evaluate" is both ineffective and slow. Moreover, there are no standard sets of QoS parameters that are declared in general for all orchestrations - which draws the need for a framework for totally ordered metrics.

### 5.1 QOrc: Upgrading Orc for QoS management

A proposal is making use of a QoS enhanced orchestration declaration called *QOrc*. Every invoked service responds with not only the desired output data but also with a set of QoS values. So, selection of a service can entail complex queries dependent on a variety of parameters for optimization. Consider a *site* `Optima` that may be invoked during an orchestration run. This site has input tuple `(QoS, AHPWeight, Constraint, Routine)` where `QoS` is the set of QoS domains with a list of corresponding values, `AHPWeight` is a set of (normalized) weights dependent on AHP criterion, `Constraint` are the (normalized) constraint functions and `Routine` is the optimization protocol to be employed. The user can specify the routine to be either binary integer or linear programming depending on the problem. A typical implementation in Orc is:

```
type Latency = Number
type Cost = Number

val l = Buffer()
val c = Buffer()
val QoS = ((Latency,l),  (Cost,c))
val AHPWeight = (0.3,0.7)
val Constraint = ((Latency,(<),0.5), (Cost,(<),0.8))
val Routine = ''binary integer''
```

For example, the following orchestration describes optimal selection from three generic services, while using the `Optima` site.

```
(Site1(), Site2(), Site3()) >((l1,c1), (l2,c2), (l3,c3))>
Optima(((Latency,[l1,l2,l3]), (Cost,[c1,c2,c3])), (0.3,0.7),
((Latency,(<),0.5), (Cost,(<),0.8)), ''binary integer'')
```

A library of optimization routines available as services allow complex decision making in orchestrations, even to non-specialized users of such tools. As described in the COIN-OR (COmputational INfrastructure for Operations Research) project [12] [13], a host of solvers and APIs are provided for integrating optimization. A variety of input formats such as AMPL (A Modeling Language

for Mathematical Programming), MPS (Mathematical Programming System) and GAMS (General Algebraic Modeling System) may be used to specify the problems.

### 5.2 Interfacing QOrc to Optimization Services

We use the example of the LP file format used for the open-source *lpsolve*[2] solver to demonstrate the compatibility of an input from Orc. The input syntax of the LP format uses an *Objective Function* with associated *Constraints* and variable *Declarations*. With the inputs provided from the Orc `Optima` site, the optimization problem can be conveniently formulated to the binary integer problem. Formulation of linear or more complex quadratic problems can follow this procedure to conceal intricacies of mathematical packages from non-specialist users. The transformation of these inputs, through an interface, into a LP optimization routine is represented below:

- Generate variables `x1, x2 ... xN`, where `N` equals the number of participating services. These are the variables that will be the valued as 1 or 0 during optimization and represent the selection / rejection of a particular `SiteN()`.
- The `AHPWeight` values `(w1, w2)`, and corresponding `QoS` values `[l1,...,lN], [c1,...,cN]` are combined with the variables to generate a linearly weighted cost function `(w1 l1 + w2 c1)x1 +...+ (w1 lN + w2 cN)xN`.
- The `Constraint` values provide the specified domains, partial orders and corresponding thresholds `(K1, K2)`, which are transformed into (`l1 x1 +...+ lN xN <= K1`; `c1 x1 +... + cN xN <= K2`).
- As the `Routine` "binary integer" is set, values `x1, x2,..., xN` are further constrained to be binary valued. A further constraint automatically specified is the `x1 + x2 +...+ xN = 1`, restricting only a single site is selected by the optimization procedure.

The results of such a transformation produces a LP format of the problem, that can be solved by the lpsolve optimization solver. Due to the elegant nature of Orc, this is equivalent to calling another (possibly external) *Site* with input `Optima` format and output LP format.

```
/* Objective function */
min: (0.3 l1 + 0.7 c1) x1 + (0.3 l2 + 0.7 c2) x2 + (0.3 l3 + 0.7 c3) x3;
/* Variable bounds */
l1 x1 + l2 x2 + l3 x3 <= 0.5;
c1 x1 + c2 x2 + c3 x3 <= 0.8;
x1 + x2 + x3 = 1;
bin x1, x2, x3;
```

In the current stage of implementation calculation of normalized AHP weight vector is be performed using MATLAB. The optimization of QoS values generated from distribution fitting of actual web services' readings is also done

[2] http://lpsolve.sourceforge.net/5.5/

through MATLAB routines. This can be enhanced in future with direct calls to optimization packages (local or external) from within Orc as described. This would prevent switching between technologies while developing workflows in Orc and associated management of QoS dependent decisions.

## 6 Optimal Decision Results

The results of the optimization procedure are described in this section. Rather than concentrating on optimization aspects (primal-dual feasibility, relative error, number of iterations, etc.), we focus on the implications of using optimization as a tool in orchestrations.

The AHP weight matrix shown in Table 3 is used for the optimization of the problem described in Section 4 using the `linprog` function in MATLAB. These are the judgement criteria that can be fixed by the user / service orchestrator as the inputs to the optimization solver. The polling period $\rho$ is set to a constant of 1 hour to limit model parameters. By setting the customer demand and

| | $\mathbf{w}_{\mu_c}$ | $\mathbf{w}_{\mu_{max}}$ | $\mathbf{w}_{\beta}$ | **Normalized Vector** $\upsilon$ |
|---|---|---|---|---|
| $\mathbf{w}_{\mu_c}$ | 1 | 1/3 | 1/5 | 0.1047 |
| $\mathbf{w}_{\mu_{max}}$ | 3 | 1 | 1/3 | 0.2583 |
| $\mathbf{w}_{\beta}$ | 5 | 3 | 1 | 0.6370 |
| $c_{max} = 3.0385$, CI = 0.0193, CR = 0.0370 | | | | |

**Table 3.** Parameters for Dell supply chain optimization.

supplier delay distributions, the optimization produces the distributions of the refuel batch size, critical and maximum stock levels as shown in Fig. 3. These settings, when applied to the Dell system provides the system performance as shown in Fig. 4. The cumulative distribution of the revolver inventory remains stochastically above the critical distribution for 10000 runs, being refueled periodically by the supplier. As a consequence, the revolver stock level $\mu_t$ does not drop to zero throughout the simulation period. This demonstrates that the optimization formulation through AHP is robust to changes in inputs of demand and delay distributions. Though scaling as in eq. (5) has been employed, the normalized values are omitted from figures (to demonstrate realistic outputs).

As further seen in one particular setting of the Dell example in Fig. 5, the linear programming method converges within a few iterations to the optimal value. This is true for well formulated linear programming problems with optimal outputs produced (relative errors of the order of $\leq 10^{-6}$) for most input settings.

Parameters for optimal evaluations such as relative error, maximum number of iterations and so on can be set conveniently with most generic optimization solvers. Such a precise setting of parameters are needed for orchestrations like the Dell supply chain, to prevent unwarranted delay in production and supply of parts (choreography deadlock). This example highlights the crucial use of optimization and associated packages for managing QoS in complex workflows.

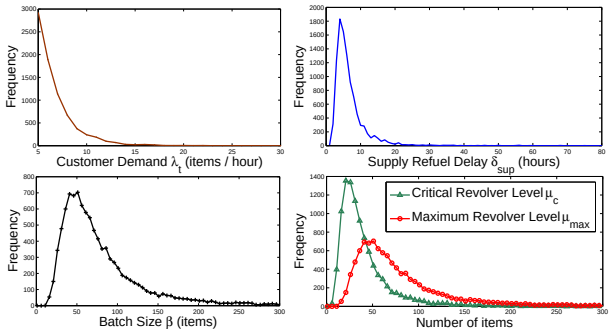


**Fig. 3.** Optimal setting of parameters in the Dell Supply Chain.

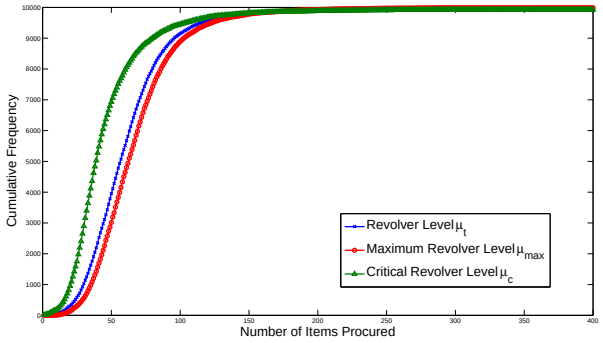


**Fig. 4.** Distributions of the inventory levels in the Dell system.

## 7 Related Work

Analysis of QoS in web services orchestrations has received considerable attention. In [1], Hwang et al. use QoS parameters as random variables for composition. Rosario et al. [2] provide a framework for probabilistic contracts modeling QoS parameters as random variables. Instead of using fixed hard bound values for parameters such as response time, the authors proposed a soft contract monitoring approach to model the QoS bounds. This is further developed with a theory for QoS modeling within the Orc framework in [7]. We extend the "best" operator from this theory to accommodate alternatives to exhaustive search.

Though there are many techniques available for optimizing functions [6] routines needed to incorporate them into orchestrations is still a developing area. In the paper by Alrifai and Risse [14] the use of mixed integer programming is proposed to find the optimal decomposition of global QoS constraints into local constraints. Optimal QoS compositions make use of genetic programming in Canfora et al. [15] and linear programming in Zeng et al. [16]. The use of a reputation guided selection and feedback dependent policy for web services is outlined in [5]. In [17], the optimization of dynamic service compositions are

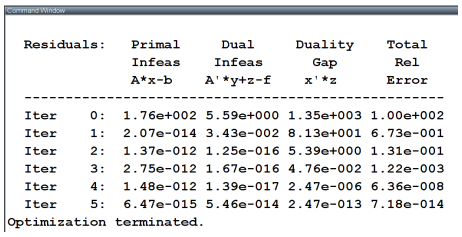


**Fig. 5.** Optimization output for a single setting of the Dell example in MATLAB.

modeled as a multidimension-multichoice knapsack problem (MMKP). MMKP of medium sizes can be solved by most commercial integer-linear programming solvers, as employed in this paper. A framework for specifying optimizations within Orc workflows would aid in deploying real-world applications. This can then be combined with a host of optimization solvers [12] [13] applied to most QoS dependent decisions in service orchestrations.

In this paper, we extend the concepts of optimizing cost function defined via AHP to complex queries in workflows. Extending such a framework to orchestrations can provide more complex queries to be incorporated with flexibility in comparing domains. The Dell optimization example from [9] provide realistic case studies within the web service framework where optimal QoS values affect functioning of the orchestration.

Analytical hierarchy process developed by Saaty [4] has been shown to be applied to diverse fields including manufacturing, logistics, finance and management. Work by Ho [18] reviews the combination of AHP to mathematical models including linear programming, integer linear programming, mixed integer linear programming, and goal programming. An application of AHP for automated negotiation of SLAs are studied in [19]. In [20], another multi-criteria decision making approach (PROMETHEE) is used to extend the decision making for exhaustive comparison of web services' QoS.

## 8 Conclusion

With increasing need for decision making capabilities in services orchestrations, the use of mathematical packages like optimization should be employed for leveraging QoS dependent choices. Embedding optimization routines as part of orchestration specifying languages like Orc provides the capability to use these tools for runtime decision making in a variety of workflows. A simple extension of user defined criterion and constraints is proposed to specify such optimization problems for non-specialist workflow designers. By applying the AHP, we show that a consistent minimizing cost function can be developed for total ordering QoS metrics. Demonstrating this methodology for the Dell supply chain example, it is shown to be effective in solving realistic problems in resource allocation and logistics. Such techniques are required to estimate optimal decisions on runtime, dependent on variations in associated QoS parameters.

## References


1. S. Y. Hwang, H. Wang, J. Tang, and J. Srivastava, "A probabilistic approach to modeling and estimating the QoS of web-services-based workflows," *Elsevier Information Sciences*, vol. 177, pp. 5484–5503, 2007.
2. S. Rosario, A. Benveniste, S. Haar, and C. Jard, "Probabilistic QoS and Soft Contracts for Transaction-Based Web Services Orchestrations," *IEEE Trans. on Services Computing*, vol. 1, no. 4, pp. 187 – 200, 2008.
3. W3c, "QoS for Web Services: Requirements and Possible Approaches," *W3C Working Group Note*, Nov. 2003.
4. T. L. Saaty, "How to make a decision: The analytic hierarchy process," *European J. of Operational Research*, vol. 48, no. 1, pp. 9 – 26, 1990.
5. N. Limam and R. Boutaba, "Assessing Software Service Quality and Trustworthiness at Selection Time," *IEEE Trans. on Software Engineering*, vol. 36, no. 4, pp. 559-574, 2010.
6. J. Nocedal and S. J. Wright, "Numerical Optimization," *Springer Series in Operational Research*, 2nd ed., 2006.
7. S. Rosario, A. Benveniste, and C. Jard, "A Theory of QoS for Web Service Orchestrations," *HAL INRIA Research Report*, 2009.
8. T. L. Saaty, "Decision-making with the AHP: Why is the principal eigenvector necessary," *Elsevier European J. of Operational Research*, vol. 145, pp. 85-91, 2003.
9. R. Kapunscinski, R. Q. Zhang, P. Carbonneau, R. Moore, and B. Reeves, "Inventory Decisions in Dells Supply Chain," *Interfaces*, vol. 34, no. 3, pp. 191-205, 2004.
10. R. L. Rardin, "Optimization in Operations Research," *Prentice Hall*, 1998.
11. J. Misra and W. R. Cook, "Computation Orchestration: A Basis for Wide-area Computing," *Springer J. of Software and Systems Modeling*, vol. 6, no. 1, pp. 83 – 110, Mar. 2007.
12. R. Fourer, J. Ma, and K. Martin, "Optimization Services: A Framework for Distributed Optimization," *COIN-OR*, 2008.
13. R. Fourer and J. Goux, "Optimization as an Internet Resource," *Interfaces*, vol. 31, no. 2, pp. 130–150, 2001.
14. M. Alrifai and T. Risse, "Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition," *Intl. World Wide Web Conf.*, Spain, 2009.
15. G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for QoS-aware service composition based on genetic algorithms," *Conf. on Genetic and evolutionary computation*, USA, pp. 1069–1075, 2005.
16. L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for Web services composition," *IEEE Trans. on Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.
17. T. Yu, Y. Zhang, and K. Lin, "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints," *ACM Trans. on the Web*, vol. 1, no. 1, 2007.
18. W. Ho, "Integrated analytic hierarchy process and its applications A literature review," *European J. of Operational Research*, vol. 186, no. 1, pp. 211-228, 2008.
19. C. Cappiello, M. Comuzzi, and P. Plebani, "On Automated Generation of Web Service Level Agreements," *Intl. Conf. on Advanced Info. Sys. Engineering*, LNCS 4495, pp. 264-278, 2007.
20. Y. Seo, H. Jeong, and Y. Song, "Best Web Service Selection Based on the Decision Making Between QoS Criteria of Service," *Intl. Conf. on Embedded Soft. and Sys.*, LNCS 3820, pp. 408-419, 2005.