QoS Analysis in Heterogeneous Choreography Interactions

Ajay Kattepur, Nikolaos Georgantas, Valérie Issarny

QoS Analysis in Heterogeneous Choreography Interactions

Ajay Kattepur, Nikolaos Georgantas & Valérie Issarny*

Equipe ARLES, Inria Paris-Rocquencourt, France.
email: firstname.lastname@inria.fr

Abstract. With an increasing number of services and devices interacting in a decentralized manner, *choreographies* are an active area of investigation. The heterogeneous nature of interacting systems leads to choreographies that may not only include conventional services, but also sensor-actuator networks, databases and service feeds. Their middleware behavior within choreographies is captured through abstract interaction paradigms such as *client-service*, *publish-subscribe* and *tuple space*. In this paper, we study these heterogeneous interaction paradigms, connected through an *eXtensible Service Bus* proposed in the *CHOReOS* project. As the functioning of such choreographies is dependent on the Quality of Service (QoS) performance of participating entities, an intricate analysis of interaction paradigms and their effect on QoS metrics is needed. We study the composition of QoS metrics in heterogeneous choreographies, and the subsequent tradeoffs. This produces interesting insights such as selection of a particular system and its middleware during design time or end-to-end QoS expectation/guarantees during runtime. Non-parametric hypothesis tests are applied to systems, where QoS dependent services may be replaced at runtime to prevent deterioration in performance.

Keywords: Heterogeneous Choreographies, Quality of Service, Interaction Paradigms, Middleware Connectors.

1 Introduction

Choreographies, unlike centrally controlled orchestrations, involve a decentralized service composition framework where only the participants' functionality and associated message passing are described [1]. Service Oriented Architectures (SOA) allow choreography components to interact via standard interfaces, with the enterprise service bus (ESB) [3] providing a common middleware protocol to convey the messaging interactions. However, these are principally based on the *client-service* interaction paradigm, as, for instance, with RESTful services [20].

Heterogeneous choreographies that involve conventional services, sensor networks and data feeds, such as those seen in the *Internet of Things* [12], require additional paradigms to ensure interoperability. Heterogeneous applications are handled at the middleware level with varied interactions, data structures and communication protocols made interoperable. In particular, platforms such as

* This work has been partially supported by the European Union's 7th Framework Programme FP7/ 2007-2013 under grant agreement number 257178 (project CHOReOS, `http://www.choreos.eu`).

REST [20] based *client-service* interactions, *publish-subscribe* based Java Messaging Service [19] or JavaSpaces [9] for *tuple space* are made interoperable through *middleware protocol converters*.

Studies that characterize the Quality of Service (QoS) in web services orchestrations have conventionally been done at the application level; heterogeneous choreographies consisting of services and sensor networks require further QoS analysis of paradigms, unless they can rely on QoS aware middleware [18]. An intricate analysis of QoS at the abstract level of interaction paradigms, in addition to the application level, would enable analysis of heterogeneous choreographies.

In this paper, we use the *eXtensible Service Bus (XSB)* proposed by the *CHOReOS* project [4,10] in order to deal with the heterogeneous aspects of choreographies. The common protocol of XSB preserves the interaction paradigms of the individual components, while still allowing interoperability. It supports interoperability among the three aforementioned, widely used, middleware paradigms: *client-service*, *publish-subscribe* and *tuple space*.

We enhance the middleware paradigms that are employed inside heterogeneous choreographies with QoS composition frameworks. While our previous work [15] studies the effect of choreography topology on the QoS, by fine-grained analysis of message interactions, we evaluate the performance of choreography participants in relation to heterogeneous paradigms. This methodology enables: 1) Design-time selection of interaction paradigms to match required functional and QoS goals, and 2) Runtime analysis of composed choreographies to prevent deterioration of end-to-end QoS of participants. Interesting facets include the use of non-parametric Kolmogorov-Smirnov hypothesis testing to replace an interaction paradigm with another, when abstracted with a particular QoS metric.

The rest of the paper is organized as follows. An overview of heterogeneous interaction paradigms and XSB is provided in Section 2. The QoS domains, metrics and algebra for composition are studied in Section 3. The methodology of measuring and propagating QoS increments across various domains are analyzed in Section 4. The results of our analysis through experiments are presented in Section 5, which includes an analysis of tradeoffs and interaction substitution. This is followed by related work and conclusions in Sections 6 and 7, respectively.

2 Interconnecting Heterogeneous Interaction Paradigms

In this section, we briefly describe the three interaction paradigms that may be abstractly applied within our QoS analysis framework. The functional semantics of these paradigms are abstracted into a set of corresponding *middleware connectors* proposed by the *CHOReOS* project. After that, we provide an overview of the *eXtensible Service Bus (XSB) connector*, which ensures interoperability across these connectors and the represented paradigms.

2.1 Interaction Paradigm Connectors

In order to let choreographies include services (typically, client-service interactions), service feeds (publish-subscribe), and sensor-actuator networks (shared tuple spaces), it is a requirement to allow heterogeneity. We briefly review in

Interaction	Primitives	Arguments
Client-Service	send	destination, operation, message
	receive_sync	↑source, ↑operation, ↑message, timeout
	receive_async	source, operation, ↑callback(source, operation, message), ↑handle
	end_receive_async	handle
	invoke_sync	destination, operation, in_msg, ↑out_msg, timeout
	invoke_async	destination, operation, in_msg, ↑callback(out_msg), ↑handle
Publish-Subscribe	publish	broker, filter, event, lease
	subscribe	broker, ↑filter, ↑handle
	get_next	handle, ↑event, timeout
	listen	handle, ↑callback(event)
	end_listen	handle
	unsubscribe	handle
Tuple Space	out	tspace, extent, template, tuple, lease
	take	tspace, extent, template, policy, ↑tuple, timeout
	read	tspace, extent, template, policy, ↑tuple, timeout
	register	tspace, extent, template, ↑callback(), ↑handle
	unregister	handle

Table 1. APIs of Interaction Paradigms.

the following the relevant interaction paradigms [10], and discuss the Application Programming Interfaces (APIs) of corresponding connectors [4]. These APIs are depicted in Table 1. Their primitives and arguments are provided, with ↑ representing an `out` or `in-out` return argument of a primitive.

- *Client-Service (CS)* - Commonly used paradigm for web services. The client may send a message with `send`; the receiving service blocks further execution until synchronization or `timeout`, with a `receive_sync`. Alternatively, asynchronous reception can be set up with `receive_async`; then, a `callback` is triggered by the middleware when a message arrives. The two-way request-response invocation procedure is further captured by the `invoke_sync` and `invoke_async` primitives. CS represents tight *space coupling*, with the client and service having knowledge of each other. There is also tight *time coupling*, with service availability being crucial for successful message passing.
- *Publish-Subscribe (PS)* - Commonly used paradigm for content broadcasting/feeds. Multiple peers interact using an intermediate *broker* service. Publishers `publish` events that may be forwarded to peers via the broker until a `lease` period. Filtering (`filter`) of messages may be done with respect to subscriptions (`subscribe`) of the peers. Besides synchronous reception (`get_next` with `timeout`) of an event, asynchronous reception of multiple events is procured via `listen` and `callback`. PS allows *space decoupling*, as the peers need not know each other. Additionally, *time decoupling* is possible, with the disconnected peers receiving updates synchronously or asynchronously when reconnected to the broker.
- *Tuple Space (TS)* - Commonly used for shared data with multiple read/write users. Peers interact with a *data space*, with participants having write (`out`), `read` and data removal (`take`) access. The peers can retrieve data whose value matches a `tuple` pattern (`template`), either synchronously with a `timeout` or asynchronously via `register` and `callback`. A peer may connect to the space at any time and procure data before the `lease` period. TS enables both space and time decoupling between interacting peers.

2.2 eXtensible Service Bus (XSB)

CHOReOS [4] uses the following abstractions to deal with large scale choreographies that connect heterogeneous participants:

- *Components*: The heterogeneity of services and devices encountered are modeled as *service interface abstractions*, which represent groups of alternative services that provide similar functionalities through varied interfaces.
- *Connectors*: This definition relates to the introduction of a new multi-paradigm *eXtensible Service Bus (XSB) connector*, which allows components to interoperate even if they are based on heterogeneous interaction paradigms. XSB extends the conventional ESB system integration paradigm [10].
- *Coordination Delegates*: These are used to enforce the realizability of choreographies despite the autonomy of the composed services. They ensure that the choreography specifications, such as message ordering integrity, are adhered to by the participating entities.

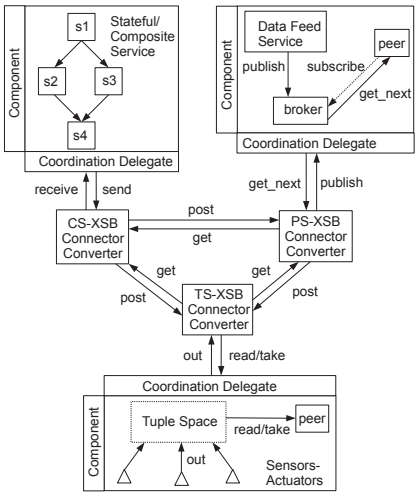

Fig. 1. *CHOReOS* Choreography Model.

The generic representation of the *CHOReOS* model is shown in Fig. 1, with services/devices represented by *components* abstracting their functional behavior. At the choreography level, the *coordination delegates* wrap such services and adapt their roles to message passing specifications. The *XSB connector* ensures interoperability across a host of middleware protocols (following the CS, PS, TS paradigms). Note that there are multiple types of components that may participate: *Atomic/Composite Services* (CS requests-responses); *Data Feed Services* (they publish PS events, which are then passed to subscribers); *Sensor Actuator Networks* (TS-based interaction).

The semantics of the XSB connector is elicited as the *greatest common denominator* of the semantics of the CS, PS and TS connectors. As the latter semantics are incompatible in certain aspects, some enforcement by the applications employing the heterogeneous connectors may be necessary. For example,

Primitives	Arguments
post	scope, data
get_sync	↑scope, ↑data, timeout
get_async	scope, ↑callback(scope, data), ↑handle
end_get_async	handle
post_get_sync	scope, in_data, ↑out_data, timeout
post_get_async	scope, in_data, ↑callback(out_data), ↑handle

Table 2. XSB connector API.

the two-way, time-coupled, CS interaction has no equivalent in the PS and TS paradigms. In this case, the PS and TS applications interacting with a CS application will have to complement the semantics of their underlying connectors with the lacking behavior (e.g., ensure that a PS peer receiving a published event will respond by publishing a correlated event that will be received by the initial publishing peer). Hence, the XSB connector can abstractly represent any of the three CS, PS and TS connectors. The XSB API is depicted in Table 2. It employs primitives such as `post` and `get` to abstract CS (`send, receive`), PS (`publish, get_next`), and TS (`out, take/read`) interactions. The `data` element can represent a CS `message`, PS `event` or TS `tuple`. The `scope` argument is used to unify space coupling (addressing mechanisms) across CS, PS and TS. Two-way interaction is enabled with the `post-get` primitive. Additionally, XSB is the *common bus protocol* employed for the interconnection of CS, PS and TS systems, as seen in Fig. 1. Finally, XSB represents also the *end-to-end interaction protocol* among such interconnected systems.

3 Modeling Quality of Service

While conventional middleware connectors focus on heterogeneity and functional interoperability, choreographies include additional non-functional metrics during design/runtime. This specifically involves analysis of constraints on QoS performance of individual participants (at the application/middleware level) and their side-effects on choreography enaction. QoS metrics being probabilistic and multi-dimensional, accurate analysis of increments and composition rules are crucial. In this section, the QoS domains that are of interest for interactions and the corresponding algebra for their composition are analyzed.

3.1 QoS Domains

We review the basic domains of QoS that require analysis for heterogeneous choreography interactions. We identify three basic domains [6]:

- δ: *Timeliness* incorporates aspects such as latency and reliable message delivery. For the case of client-service interactions, timeliness concerns one-way message or two-way request-response latency. In case of publish-subscribe, the latency between publication to a broker and subsequent coupled or decoupled delivery to peers is examined. With tuple space, the latency between writing to a tuple and coupled or decoupled access to the data is analyzed.
- $\mathcal{S}$: *Security/Data Integrity* incorporates the trustworthiness of the interaction paradigms with respect to the confidentiality of information. This especially holds in publish-subscribe systems, where there is an intermediate broker, or

QoS Metric	$\mathbb{D}$	$\leq$	$\oplus$	$\bigwedge$	$\bigvee$
δ: Timeliness	$\mathbb{R}_+$	$<$	$+$	min	max
$\mathcal{S}$: Security/Data Integrity	finite set $\mathbf{Q}$	$>$	$\bigvee$	max	min
λ: Resource Efficiency	finite set $\mathbf{Q}$	$<$	$+$	min	max

Table 3. Basic classes of QoS domains

tuple spaces, where there is an intermediate space and multiple peers have access to the same data. For example, a peer in the tuple space may remove and modify the data before they are procured by the other peers.

- λ: *Resource Efficiency* incorporates multiple aspects, such as efficiency in bandwidth usage and protocol message passing. In the case of publish-subscribe, for instance, the additional resources needed for subscription messages are to be included. Generally, this may be traded off with timeliness: greater bandwidth usage/active sessions help in timely delivery of messages. Analysis of these tradeoffs will help understand the pros and cons of a particular interaction paradigm.

3.2 QoS Algebra

In order to aggregate metrics available from heterogeneous interactions, we make use of an algebraic framework as introduced in [21]. This can handle random variables drawn from a distribution associated with a lattice domain. A *QoS metric* q is a tuple:

$$q = (\mathbb{D}, \leq, \oplus, \bigwedge, \bigvee) \tag{1}$$

1. $(\mathbb{D}, \leq)$ is a QoS domain with a corresponding partially ordered set of QoS values.
2. $\oplus : \mathbb{D} \times \mathbb{D} \rightarrow \mathbb{D}$ defines how QoS gets incremented by each new *action* or *operation*, like sending a message or receiving an event. It satisfies the following conditions:
   - $\oplus$ possesses a *neutral element* q_0 satisfying $\forall q \in \mathbb{D} \Rightarrow q \oplus q_0 = q_0 \oplus q = q$.
   - $\oplus$ is *monotonic*: $q_1 \leq q_1'$ and $q_2 \leq q_2'$ imply $(q_1 \oplus q_2) \leq (q_1' \oplus q_2')$.
3. $(\bigwedge, \bigvee)$ represent the lower and upper lattice, meaning that any $q \subseteq \mathbb{D}$ has a unique greatest lower, least upper bound $(\bigwedge_q, \bigvee_q)$. When taking the *best* QoS with respect to the ordering $\leq$, we take the lowest QoS value, with $\bigwedge$. When synchronizing (for instance, with fork-joins), the operator $\bigvee$ amounts to taking the *worst* QoS as per the ordering $\leq$.

Basic classes of QoS domains are displayed in Table 3 and composed according to rules specified in Eq. 1. The use of this algebraic framework allows us to reason, in an abstract way, about the behavior of interaction paradigms and their effect on choreography performance. The framework may be invoked by any choreography description language to incorporate QoS composition. It specifies calculation of the QoS increments via the associated algebra with domains $\mathbb{D}$, partial order $\leq$, and operations $(\oplus, \bigvee, \bigwedge)$. Note that the algebra is "general" enough to incorporate multiple units for each domain. For metric $\mathcal{S}$, the domain $\mathbf{Q}$ can be subjective to scaled preferences such as $\{low, medium, high\}$. The operation $\oplus$ is treated as $\bigvee$, modeling instances where "high" security data is passed to a "low" security service.

4 QoS Analysis of Interactions

In order to upgrade choreography interactions with QoS assessment, we equip every *atomic transaction* $\mathcal{T}$ in a client-service, publish-subscribe, or tuple space interaction with a QoS increment; $\mathcal{T}$ represents an end-to-end interaction enabling sending and receiving of data. From Fig. 1, $\mathcal{T}$ includes end-to-end data transfer between components enabled by the XSB connectors. This produces a tuple of $(\mathcal{T}, q)$ that may be propagated along the choreography. As the choreography does not have a centralized QoS control mechanism, this propagation of tuples can be combined with the algebraic operators to aggregate these increments. As the QoS values are random variables, the collected increments may be used either at *design time* (using statistical data) or for *run-time* monitoring.

4.1 QoS Model for Generic XSB Transactions

The XSB connector can represent end-to-end transactions for any one of the CS, PS, TS connectors. One-way interaction can be abstracted as follows: a sender can `post` *data* – representing a *message*, *event* or *tuple* – with a validity period `lease`; this is procured (using `get`) within the `timeout` period at the receiver side. The peers initiate their actions independently.

Model for Timeliness. Fig. 2 depicts an one-way XSB transaction as a correlation in time between a `post` action and a `get` action. The `post` and `get` operations are asynchronous and have individual time-stamps. The `post` operation is initiated at t_{post_0}. At t_{med}, the posted data arrive at the intermediary *medium*; we introduce this notion to represent the broker/data space in the case of PS/TS, or the remote CS middleware entity. A timer is initiated at t_{med}, constraining the data availability to the `lease` period t_{lease}. Note that the `lease` period may be set to 0, as in the case of CS messages. Similarly at the receiver side, the `get` operation is initiated at t_{get_0}, together with a timer controlling the `timeout` period $t_{timeout}$. If `get` returns before the `timeout` period with valid data (not exceeding the `lease`), then the transaction is successful. We consider this instance also as the end of the `post` operation. Hence, if the transaction is successful, the overall QoS increment is:

$$\delta = \bigvee(\delta_{post},\ \delta_{get}) \tag{2}$$

where δ_{post} and δ_{get} represent the durations of the two corresponding actions. In the case of failure, there is no overlapping in time between the two actions. In other words, only one of them takes place, and goes up to its maximum duration, i.e., δ_{med} + `lease` for `post` or `timeout` for `get`, while the other's duration is `0`. Hence, the QoS output is once again as in Eq. 2. Finally, we note that, while we present the synchronous data reception case, the case with asynchronous `get` and callbacks follows similar timeliness composition models.

Model for Security/Data Quality. In order to model the data security level associated with each transaction, we equip the *data-carrying* `post` and `get` operations, depicted in Fig. 3, with a security level. Note that the `post` operation

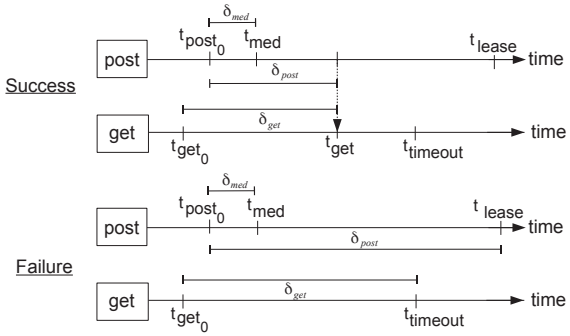

Fig. 2. Analysis of `post` and `get` δ increments for success and failure.

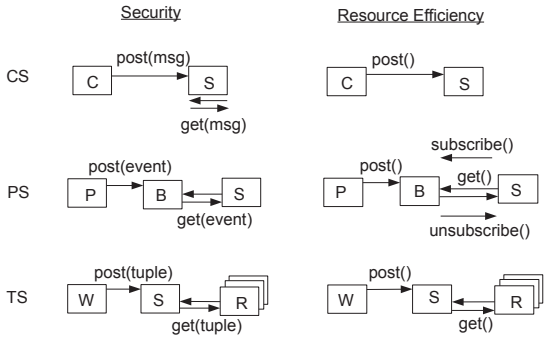

Fig. 3. Analysis of `post` and `get` $\mathcal{S}$ and λ increments for success and failure.

here refers to the interaction between the sender and what we called above the *medium*, i.e., the PS broker, TS data space, or remote CS middleware entity. Locally executed actions, such as `get(msg)` in the case of CS schemes, come equipped with good security levels. In case of TS schemes, as there is a shared channel between peers (unlike the exclusive channel of CS, PS), the security levels are worse than, for example, PS. For a successful transaction, the supremum of the security levels linked with the actions are taken, which means the worst security level among the supported ones:

$$\mathcal{S} = \bigvee(\mathcal{S}_{post},\ \mathcal{S}_{get}) \tag{3}$$

In the case of failure, Eq. 3 still holds, with the operation that did not take place carrying a `null` security level. Finally, for asynchronous data reception, the security level composition is similar to the synchronous case presented here.

Model for Resource Efficiency. When measuring resource efficiency, we include the subset of all *networked* primitives related to the `post` and `get` operations, as in Fig. 3. In the case of PS, for instance, the subscription level primitives are taken into account. Note that we only consider synchronous data reception in our evaluation; for asynchronous callbacks, resource efficiency can be evaluated in a similar fashion. In case of success, the resultant resource efficiency is:

$$\lambda = \lambda_{post} \oplus \lambda_{get} \tag{4}$$

Primitives	Arguments
post	scope, data, q_post
get_sync	↑scope, ↑data, timeout, ↑q_get_sync
get_async	scope, ↑callback(scope, data), ↑handle, ↑q_get_async
end_get_async	handle, q_end_get_async
post_get_sync	scope, in_data, q_post, ↑out_data, timeout, ↑q_post_get_sync
post_get_async	scope, in_data, q_post, ↑callback(out_data), ↑handle, ↑q_post_get_async

Table 4. Extending the XSB API for QoS Analysis.

In case of failure, Eq. 4 still holds, with the missing operation contributing a `null` value to the metric.

4.2 Upgrading the API

We append the API arguments of Table 2 with *QoS parameters* that may be either self-measured by the peers or aggregated through a third party service. The QoS increments and composition are presented in Table 4. The `post` operation is given an initial QoS value `q_post`. This is, for the example of timeliness, the timestamp t_{post_0} (see Fig. 2), which is then used at the receiver to calculate the final returned output QoS increment. For a successful `get_sync`, the value specified in Eq. 2 is returned with `q_get_sync`. In the case of failure, the resulting timeliness value can be measured either by the receiver (`timeout`) or by a probe installed at the medium (δ_{med} + `lease`).

4.3 Model for QoS Propagation

The QoS model for generic XSB transactions and the related API introduced in Sections 4.1 and 4.2 can be applied for measuring QoS in heterogeneous choreographies where CS, PS and TS systems are interconnected via an XSB bus (see Fig. 1). In particular, the model and API introduced for XSB can be easily transcribed to the corresponding primitives and transactions of CS, PS and TS. Additionally, they can be used directly for the transactions performed on the XSB bus interconnecting heterogeneous systems. For multiple sequential choreography transactions, QoS increments can be propagated along with the transaction data and be passed from one transaction to the following one. Hence, QoS values can be calculated, propagated and aggregated along end-to-end choreography links, such as the ones depicted in Fig. 1. For example, in the case of timeliness for an one-way CS-XSB-PS transaction, we need to aggregate the three involved transactions:

$$\delta = \bigvee(\delta_{send},\ \delta_{receive}) \oplus \bigvee(\delta_{post},\ \delta_{get}) \oplus \bigvee(\delta_{publish},\ \delta_{getnext}) \tag{5}$$

In a similar fashion, for timeliness or other QoS metrics, QoS increments can be composed for both one-way and two-way interaction.

5 Results: QoS in Choreography Interactions

Choreographies involve heterogeneous interactions between services, things (sensors/actuators), computational entities and human participants. The use of our QoS analysis enables the following:

1. *Bottom-up Choreography Designs*: where the interactions are fixed but the choreography enaction and the expected QoS can be modified. The composition models take into account the nature of the interaction and their effect on the composed QoS. This is primarily done at *design-time* with previously collected statistics.
2. *Top-down Choreography Designs*: fixed choreography specifications that may be implemented by varied interaction paradigms. At *runtime*, similar functionality may be replicated by services/things in a registry – leading to late binding. Focusing on QoS, we study the possibility of replacing an interaction with another to prevent deterioration of output QoS.

For example, an improvement in performance δ by the interactions is traded off with deteriorating λ. Thus, if cost of bandwidth is to be taken into consideration, a choice can be made at design time to select a particular interaction paradigm over another. This involves discovering services that are implemented with specific interaction paradigms. In case this is not exposed, the worst case performance for each domain must be expected at design time. At runtime, if re-configuration [15] or replacement/late-binding occurs, changes that may be expected through varied interaction paradigms may be evaluated. For instance, reduced $\mathcal{S}$ may be traded off with improvements in δ.

5.1 Comparison of Tradeoffs

To compare the effect of CS/PS/TS paradigms on QoS metrics such as timeliness, security and message efficiency, simulations were performed according to the models provided in Section 4.1. As any particular implementation is affected by the network load, middleware and individual applications' QoS increments, we assume some general characteristics in our simulations. The interactions are assumed to follow tight space-time coupling for CS/PS/TS to prevent failed transactions (even though our analysis in Section 4.1 can handle this). The details for the specific interactions are:

- *Client-Service* - At (t_{post_0}), every client posts a message to a single server. Two measurements are made: the QoS increments associated with one-way post-get messages; the QoS increments with round-trip two-way request-response invocations. CS assumes tight space-time coupling and that the server is available and connected to the client. The response `get(msg)` is linked with the end-to-end QoS increments.
- *Publish Subscribe* - At (t_{post_0}), a publication to a broker is initiated, which is then forwarded to the peers. We assume that the broker is efficient and that it forwards the messages to all subscribed peers synchronously (`get_next`). The broker intermediary adds some latency δ_{broker} and has some effect on the security level $\mathcal{S}_{broker}$. The subscription level messages `subscribe`, `unsubscribe` are considered during message efficiency calculations. While PS schemes typically allow only one-way publisher to subscriber messaging, to compare with the two-way CS case, we assume that the applications may behave as a publisher+subscriber for the two-way interaction.
- *Tuple Space* - At (t_{post_0}), data is written to a tuple, which may be read by peers. Synchronous write-read scenarios are studied, with the data space

QoS Increments	MATLAB call
δ_{send}, $\delta_{receive}$, δ_{pub}, $\delta_{getnext}$, δ_{out}, δ_{read}	`nctrnd(1,10)`
δ_{broker}, $\delta_{tuplespace}$, δ_{post}, δ_{get}	`nctrnd(1,2)`
$\mathcal{S}_{send}$, $\mathcal{S}_{receive}$, $\mathcal{S}_{pub}$, $\mathcal{S}_{getnext}$, $\mathcal{S}_{out}$, $\mathcal{S}_{read}$, $\mathcal{S}_{broker}$, $\mathcal{S}_{post}$, $\mathcal{S}_{get}$	`randi(3)`
$\mathcal{S}_{tuplespace}$	`randi(2)`
λ_{send}, $\lambda_{receive}$, λ_{pub}, $\lambda_{getnext}$, λ_{out}, λ_{read}, λ_{sub}, λ_{endsub}, λ_{post}, λ_{get}	`exprnd(5)`

Table 5. Simulation Parameters in MATLAB.

being efficient in matching tuples. Note that there may be QoS increments introduced between writing and reading from the tuple, captured by $\delta_{tuplespace}$, $\mathcal{S}_{tuplespace}$. Of particular interest is the data integrity/security of the tuples, as these may be modified/removed by any of the peers. Two-way interaction is additionally studied, with a writer to a tuple later functioning as a reader.

– *XSB* - When converting between these schemes via a common bus protocol provided by the XSB, QoS increments are produced. These are appended for various domains using $\delta_{post}, \mathcal{S}_{get}$ and so on. The conversion also increases the bandwidth resource usage that must be taken into account.

Based on [7], the QoS random variables are modeled as follows: δ as a *heavy-tailed* (`nctrnd`) distribution; $\mathcal{S}$ as a *uniform* (`randi`) distribution; λ as an *exponential* (`exprnd`) distribution. The simulations are done in MATLAB with increments and random variables provided as in Table 5. We assume uniform performance of the interactions, with δ_{send}, δ_{post}, δ_{pub} drawn from distributions with similar mean and variances. Slight variations are provided, with faster response times pertaining to the broker, tuple space and bus protocols. The security level of the tuple space is set to be lower than other interaction paradigms, as the data can be maliciously modified by peers. These values can differ according to the implementations, network load and resource management accorded.

As shown in Fig. 4, there are differences in performance of these schemes for the evaluated QoS metrics. The cases considered were: *one-way interaction*: CS send-receive, PS publish-subscribe, TS write-read; *two-way interaction*: CS invocation, PS publish-subscribe-(re)publish-subscribe, TS write-read-(re)write-read. In case of *timeliness* δ, the one and two-way CS schemes performed superiorly to corresponding TS and PS schemes. For *security* $\mathcal{S}$, as an intermediate broker or data space are employed by the PS/TS schemes, the levels are consistently lower than that of the CS scheme. The security level of the TS scheme is lower than that of the PS scheme due to the ability of peers to access common data. *Message efficiency* λ considers individual subscriptions needed by the PS scheme, which increases the number of messages per interaction.

We continue this evaluation in Fig. 5 with the effect of using the intermediate XSB connectors on these metrics. The increments are studied for two-way interaction across connectors. While there are not significant differences in domains δ and $\mathcal{S}$ for the CS-PS-TS interconnection, the CS-TS interconnection has lower message efficiency λ. Having these statistics in mind, it is possible to study the runtime replacement of a particular connection with another, as we see in the next section.

5.2 Substituting Interactions

In large spaces of services and devices [12], late-binding and replacement procedures are commonly employed. The replacement of heterogeneous systems in

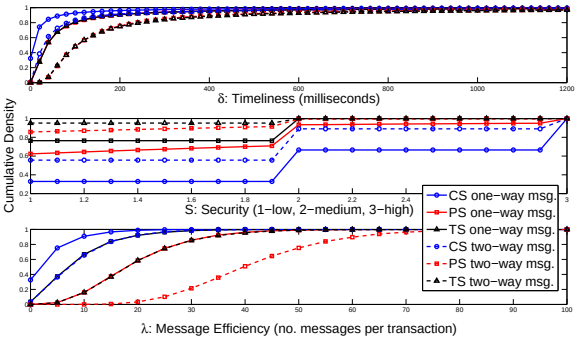

Fig. 4. QoS composition with CS, TS, PS paradigms.

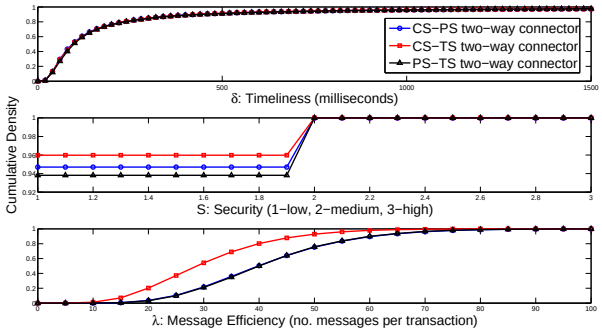

Fig. 5. QoS composition across the XSB connector.

such cases should take into account the interaction paradigms and their corresponding QoS. A level of control is to check if the replacement would not affect the particular QoS metric in hand. In order to compare the statistics provided by two interaction paradigms, we employ nonparametric hypothesis tests.

The two-sample *Kolmogorov-Smirnov* test [5] is a nonparametric hypothesis test that evaluates the difference between the cumulative distribution functions of two sample data sets. This statistic may also be used to test whether two underlying one-dimensional probability distributions differ:

$$\mathbf{KS}_{n,n'} = \bigvee_{q} \mid F_{1,n}(q) - F_{2,n'}(q) \mid \tag{6}$$

where $F_{1,n}$ and $F_{2,n'}$ are the empirical distribution functions of the first and the second sample data sets with n and n' elements in the vectors, respectively. The hypothesis test result is returned as a logical value: $\mathbf{h} = 0$ indicates an acceptance of the null hypothesis at α significance level: that the data in vectors q_1 and q_2 are from the same distribution; $\mathbf{h} = 1$ indicates the rejection of the null hypothesis at α significance level: the alternative hypothesis that q_1 and q_2 are from different distributions. The null hypothesis is accepted at level α if:

$$\sqrt{\frac{nn'}{n+n'}}\mathbf{KS}_{n,n'} \leq K_\alpha \tag{7}$$

δ: **Timeliness**

Connectors	CS-PS	CS-TS	PS-TS
CS-PS	–	`h = 0; KSstat = 0.0115`	`h = 1; KSstat = 0.0309`
CS-TS	`h = 0; KSstat = 0.0115`	–	`h = 1; KSstat = 0.0240`
PS-TS	`h = 1; KSstat = 0.0309`	`h = 1; KSstat = 0.0240`	–

$\mathcal{S}$: **Security**

Connectors	CS-PS	CS-TS	PS-TS
CS-PS	–	`h = 0; KSstat = 0.0128`	`h = 0; KSstat = 0.0088`
CS-TS	`h = 0; KSstat = 0.0128`	–	`h = 0; KSstat = 0.0216`
PS-TS	`h = 0; KSstat = 0.0088`	`h = 0; KSstat = 0.0216`	–

λ: **Message Efficiency**

Connectors	CS-PS	CS-TS	PS-TS
CS-PS	–	`h = 1; KSstat = 0.3335`	`h = 0; KSstat = 0.0102`
CS-TS	`h = 1; KSstat = 0.3335`	–	`h = 1; KSstat = 0.3413`
PS-TS	`h = 0; KSstat = 0.0102`	`h = 1; KSstat = 0.3413`	–

Table 6. KS Tests applied to various connectors.

We make use of this test on the observations of the CS, TS, PS schemes when applied to connectors, such as in Fig. 5. This is to determine whether a particular scheme can be replaced with another when querying for a particular QoS metric $(\delta, \mathcal{S}, \lambda)$. We set this tests in MATLAB with $\alpha = 1\%$ as `[h,KSstat] = kstest2(q1,q2,0.01)` in Table 6.

This provides some interesting insights: assuming certain distributions on the underlying interactions (as in Fig. 5), for timeliness δ, the CS-PS interaction can be suitably replaced by the CS-TS interaction; for security $\mathcal{S}$, all interactions are replaceable with the 1% confidence interval selected; for message efficiency λ, the CS-PS interaction can be suitably replaced by the PS-TS interaction. Though this can change according to measurements and collected statistics, this procedure can be applied in general cases to safely replace interactions. For example, if contractual obligations and SLAs need to be met in certain domains, deterioration of the QoS metrics due to replacement would be deterred. This sort of comparison not only takes into account the probabilistic nature of the QoS metrics, but also the tradeoffs provided due to the multi-dimensional QoS evaluation and corresponding interaction paradigms.

6 Related Work

QoS issues in web services span multiple topics, such as optimal late-binding (discovery, selection, substitution) and contract management (SLAs, negotiation, monitoring). Relevant QoS analysis techniques are used by Zeng et al. [23] for optimal decomposition of global QoS constraints into local constraints for composition in the case of service orchestrations. An algebraic formulation based on multi-dimensional probabilistic models is proposed in [21] to compose QoS metrics in the case of web services orchestrations. This has been used to support optimization problems for decision making in orchestrations [14]. In our work, we make use of this algebraic framework and provide an extension for the case of heterogeneous choreography QoS composition.

While QoS issues in composite services based on centralized control (orchestrations) have received some attention, the metrics relevant to choreographies are an active area of research. In [17], Mancioppi et al. provide a structured overview of the possible metrics to be incorporated within choreographies. A generalized

stochastic Petri net model is proposed in [8] to compose QoS in choreographies. In [2], the MOSES framework is proposed as an efficient and flexible technique for runtime self-adaptation in service oriented systems. Adaptive and self-healing choreographies have been studied with the survey by Leite et al. [16] providing a systematic overview of model, measurement, agent and formal methods driven techniques for adaptation. In [11], Goldman et al. use a linear programming framework to predict the QoS of BPMN based web services choreographies. A constraint based model for QoS dependent choreographies is proposed in [13]. However, these techniques assume typical client-service interactions for analysis.

With an increasing number of devices being interconnected through the Internet of Things [12], extensions to the standard (client-service interaction based) ESB middleware adapters [3] are required. In the *CHOReOS* project [4][10], the XSB connector is provided, which incorporates multiple interaction paradigms including PS and TS schemes. In order to extend such middleware with QoS, in [18], metrics are integrated in the middleware architecture for discovery, configuration and deployment analysis. In [6], the characteristics of publish-subscribe interactions that are crucial to QoS composition are studied in considerable detail. In [22], the publish-subscribe middleware interaction is upgraded with the *Harmony* overlay messaging to prevent runtime QoS deterioration. Our work builds on heterogeneous interaction paradigms and enhances them with QoS composition rules. While QoS in the typical web services setting is done at the application level, capturing the fine-grained interactions within heterogeneous paradigms provide us with a detailed outlook of QoS aggregation policies. These may be exploited during design-time selection or runtime replacement.

7 Conclusions

QoS analysis in choreographies typically considers homogeneous client-service interactions and is performed at the application level. Choreographies of heterogeneous devices, such as those in the Internet of Things, require further fine-grained QoS analysis of underlying interaction paradigms. In this paper, we study the effect of heterogeneous middleware connectors, interconnected via the *extensible service bus* from the *CHOReOS* project, on choreography QoS metrics. Using multi-dimensional, probabilistic QoS metrics and an algebraic model for composition, this procedure reveals some interesting results. The tradeoffs in particular QoS domains may be studied along with interactions, for efficient selection during design-time. Through hypothesis tests, such as Kolmogorov-Smirnov, runtime replacement of a particular interaction paradigm with another can be performed. In the near future, we intend to apply these analysis techniques on real-world implementations of large scale heterogeneous choreographies, like the ones currently being developed in the *CHOReOS* project.

References

1. A. Barker, C. D. Walton, and D. Robertson. Choreographing web services. *IEEE Trans. on Services Computing*, 2:152–166, 2009.

2. V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola. MOSES: A framework for QoS driven runtime adaptation of service-oriented systems. *IEEE Trans. on Software Engineering*, 38(5), 2012.
3. D. A. Chappell. *Enterprise Service Bus*. O'Reilly Media, 2004.
4. CHOReOS. Final CHOReOS architectural style and its relation with the CHOReOS development process and IDRE. Technical report, Large Scale Choreographies for the Future Internet, `http://www.choreos.eu/bin/Download/Deliverables`, 2013.
5. W. J. Conover. *Practical Nonparametric Statistics*. Wiley, 1999.
6. A. Corsaro, L. Querzoni, S. Scipioni, T. S. Piergiovanni, and A. Virgillito. Quality of service in publish/subscribe middleware. *Global Data Management*, 8:1–19, 2006.
7. P. Cremonesi and G. Serazzi. End-to-end performance of web services. In *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002 Tutorial Lectures*, pages 158–178. Springer-Verlag, 2002.
8. A. P. Diaz and D. M. Batista. A methodology to define QoS and SLA requirements in service choreographies. In *17th Intl. Wksp. on Computer Aided Modeling and Design of Communication Links and Networks*, 2012.
9. E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley Professional, 1999.
10. N. Georgantas, G. Bouloukakis, S. Beauche, and V. Issarny. Service-oriented Distributed Applications in the Future Internet: The Case for Interaction Paradigm Interoperability. In *European Conference on Service-Oriented and Cloud Computing (ESOCC)*, 2013.
11. A. Goldman, Y. Ngoko, and D. Milojicic. An analytical approach for predicting QoS of web services choreographies. In *Middleware for Grid and eScience*, 2012.
12. D. Guinard, S. Karnouskos, V. Trifa, B. Dober, P. Spiess, and D. Savio. Interacting with the SOA-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE Trans. on Services Computing*, 3:223–235, 2010.
13. D. Ivanovic, M. Carro, and M. V. Hermenegildo. A constraint-based approach to quality assurance in service choreographies. In *ICSOC*, 2012.
14. A. Kattepur, A. Benveniste, and C. Jard. Optimizing decisions in web services orchestrations. In *ICSOC*, pages 77–91, 2011.
15. A. Kattepur, N. Georgantas, and V. Issarny. QoS composition and analysis in reconfigurable web services choreographies. In *Intl. Conf. on Web Services*, 2013.
16. L. A. F. Leite, G. A. Oliva, G. M. Nogueira, M. A. Gerosa, F. Kon, and D. S. Milojicic. A systematic literature review of service choreography adaptation. *Service Oriented Computing and Applications*, pages 1–18, 2012.
17. M. Mancioppi, M. Perepletchikov, C. Ryan, W.-J. van den Heuvel, and M. P. Papazoglou. Towards a quality model for choreography. In *ICSOC/ServiceWave*, 2010.
18. K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li. QoS-aware middleware for ubiquitous and heterogeneous environments. *IEEE Communications Magazine*, 39:140–148, 2001.
19. M. Richards, R. Monson-Haefel, and D. A. Chappell. *Java Message Service*. O'Reilly, second edition, 2009.
20. L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly, 2007.
21. S. Rosario, A. Benveniste, and C. Jard. Flexible probabilistic QoS management of transaction based web services orchestrations. In *IEEE Intl. Conf. on Web Services*, pages 107 –114, 2009.
22. H. Yang, M. Kim, K. Karenos, F. Ye, and H. Lei. Message-oriented middleware with QoS awareness. In *ICSOC*, pages 331–354, 2009.
23. 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*, 30:311–326, 2004.