Non-functional Data Collection for Adaptive Business Processes and Decision Making*

Bao Le Duc
Orange Labs
38-40 Av. du Général Leclerc
92794 Issy les Moulineaux, France
bao.leduc@orange-ftgroup.com

Pierre Châtel
Thales Communications France
1-5 avenue Carnot
91883 Massy, France
pierre.chatel@thalesgroup.com

Nicolas Rivierre
Orange Labs
38-40 Av. du Général Leclerc
92794 Issy les Moulineaux, France
nicolas.rivierre@orange-ftgroup.com

Jacques Malenfant
Université Pierre et Marie Curie-Paris 6, CNRS, UMR 7606 LIP6
104 av. du Président Kennedy
75016 Paris, France
Jacques.Malenfant@lip6.fr

Philippe Collet
Université de Nice Sophia Antipolis, CNRS, UMR 6070 I3S
Route des Colles, BP 145
06903 Sophia Antipolis Cedex, France
Philippe.Collet@unice.fr

Isis Truck
Université Paris 8
2 rue de la Liberté
93526 Saint-Denis Cedex, France
truck@ai.univ-paris8.fr

ABSTRACT

Monitoring application services becomes more and more a transverse key activity in SOA. Beyond traditional human system administration and load control, new activities such as autonomic management as well as SLA enforcement raise the stakes over monitoring requirements. In this paper, we address a new monitoring-based activity which is selecting among competitive service offers based on their currently measured QoS. Starting from this use case, the late binding of service calls in SOA given the current QoS of a set of candidate services, we first elicit the requirements and then describe M4ABP (Monitoring for Adaptive Business Process), a middleware component for monitoring services and delivering monitoring data to business processes wishing to call them. M4ABP provides solutions for general requirements: flexibility as well as performance in data access for clients, coherency of data sets and network usage optimization. Lessons learned from this first use case can be applied to similar monitoring scenario, as well as to the larger field of context-aware computing.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Miscellaneous; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Measurement, Performance, Reliability, Management, Experimentation

Keywords

1. INTRODUCTION

Computing is now in the time of distributed systems everywhere. But, distributing tasks over a network of computers entails not only the independent failure property, but also large variations in the quality of service (QoS) enjoyed by clients when calling distant applications and services. As such distributed systems are more and more constructed in a SOA settings where service offers can no longer abstract away their QoS properties, as these become a competitive argument among sometimes large numbers of equivalent service offers, as in the recent evolution towards semantic SOA where more services can be considered equivalent by relaxing syntactic constraints.

Traditional monitoring of systems for performance and human administration now becomes a much wider activity where monitoring data collected on services are published, widely or not, towards clients to provide for decision making upon offers. In SOA, QoS commitments are published as Service Level Agreements (SLA) along with service offers in registries. Most QoS-aware SOA currently use this information in a "static" way: by ensuring at service selection time that the required QoS by business processes are matched by the QoS offered by services, also to guarantee the overall SLA offered by the business process to its clients. But such static approaches are insufficient since they hardly take into account the large variations in QoS that can be experienced during business process execution, nor are they robust to failures. Hence, adaptive business processes should use cur-

*This work is supported by the French ANR (Agence Nationale de la Recherche) through the SemEUsE research project.

MW4SOC '09, November 30, 2009, Urbana Champaign, Illinois, USA

rent QoS and other run-time information made available by services to adapt themselves to such variations and events.

Two run-time activities can be associated with QoS: QoS contract enforcement and QoS-based control. In the better-known QoS contract enforcement settings, monitoring can concentrate on the ranges of acceptable QoS values in order to detect and notify only when values become out of these ranges. Yet, this is not an easy task, as trade-offs are often needed between the probability to detect such SLA violations against the cost of monitoring when too much measurements impair system performance. QoS-based control is even more demanding. In this case, the monitoring system does not know the exact use of data, which are simply delivered to clients, most of the time distant, that process them in a decision-making settings. In both cases, however, arises the need for tackling the *quality of information*, that is the quality of service of the monitoring system. These stringent requirements, like flexibility and performance in data access, the coherency of data, as well as network usage optimization, call for full-fledged distributed monitoring and data delivery middleware subsystem. In this paper, we propose M4ABP (Monitoring for Adaptive Business Process), a middleware component for monitoring services and delivering timely and coherent monitoring data to business processes, so that they can be used in run-time decision-making settings.

The rest of the paper is organized as follows. Section 2 presents the technical framework of our contributions. Section 3 explicits the requirements for a service monitoring middleware component using the late-binding viewpoint as illustration. Section 4 shows how M4ABP responds to these requirements with a set of well-integrated solutions. Section 5 provides performance numbers from some experimental scenario validating these solutions. Related work are discussed in Section 6 and Section 7 closes the paper by describing future work.

2. TECHNICAL FRAMEWORK

2.1 Monitoring

Monitoring is as old as computing. It is mundane activity for operating systems, databases, applications servers and so on. Two major issues are raised in monitoring systems: measurement, or data collection, and data access. The typical trade-off in measurement is frequency of the measurement against the probability to fail to observe some important event. High frequencies entail high performance costs, while the gravity of the failure to observe some events is often usage-dependent. Hence, any monitoring system shall allow for configuration of measurement frequencies to tailor them to clients usage. When multiple clients need the same data, the monitoring system must arbitrate the different needs to keep measurement costs under acceptable levels. Precision as well as the capability to provide for sets of coherently measured data are also crucial aspects in measurement.

Data access was of less concern as long as clients where connected to the observed systems through high speed local data channels, like in system administration console for centralized data centers. Nowadays, distant clients are more and more the norm, such that the delivery of data and its access by distant clients becomes an issue. Availability of networks with guaranteed transmission delays is of course the cornerstone of timely delivery of data to the clients. In the context of best-effort networks, time-stamping provides for at least being aware of the freshness of data when used on the client-side. Both in time-guaranteed and best-effort networks, the delay in data transmission is a crucial issue. Too long delays impair the freshness of data, but also can have a very negative impact on clients if they have to wait for the transmission of data. Provisioning for no-delay access on the client side is therefore mandatory is most applications. Another important issue in the delivery is network bandwidth consumption. Besides tailoring the frequency of transmission (as in measurement) to the needs of the client, being able to transmit only once a data when required by multiple clients on one site is another crucial feature. Multicast transmission, thanks to the underlying network, to different sites can also help in lowering the bandwidth consumption.

2.2 SOA and Web Services

SOA and Web services have recently gained broad industry acceptance as established standards. They provide for greater interoperability and some protection from lock-in to proprietary vendor software. However SOA can be implemented using any kind of service-based technology. In this framework, two distinct roles are identified. *Service providers* implement (generic) features made available to applications as *Web services*, thanks to SOA standards like, e.g., service registries. *Service consumers* request and use services available on the network according to their specific requirements through service invocations made by *business processes*.

To cope with the dynamism of the Web, the binding of Web services (from providers) to business processes (of consumers) should be established on the fly, at runtime. Failure or disconnections are likely, especially in the context of pervasive SOA. Variations in the QoS of services is also a concern, given the very large base of equivalent service offers. To achieve this, and to provide for high interoperability among heterogeneous service offers and requests, this binding must be done as late as possible, taking into account the latest information of services when they are needed by the business process to be adaptive.

Monitoring of services is the key of adaptive business process. Yet, monitoring services is slightly different from the traditional server or application monitoring. Web services must be able to respond to very high demands, which forces to put behind the service entry point applications deployed on clusters of servers, and sometimes on several distant clusters. Monitoring must therefore aggregate data coming from a potential large number of measurements sites in a semantically coherent way before providing them to the clients. Mundane, but relevant issues are the form of connection offered to clients to get the data. Should monitoring requests and monitoring itself be seen as a web service? If push mode transmission of data is probably the preferential one to get the data, should a pull mode be also provided?

2.3 Non-Functional Constraints

QoS and SLA are well-known in SOA. It is crucial for business process architects to have methods and tools allowing them to specify, compute and guarantee QoS of their compositions both at static time and runtime. The need for practical solutions for service selection based on differentiated QoS is exacerbated by the number of functionally equivalent services that is likely to grow larger over the Web as well as pervasive environment where numerous services,

representing functionally equivalent devices spread over areas, shall be subject not only to static but also run-time selection.

Classic approaches rely on static QoS information attached to (SLAs) to handle QoS in service compositions. They perform the selection of services whose advertised QoS offers can satisfy the (potentially multiple) QoS requirements of a composition before a composition is deployed on an orchestration engine [12]. Some works leverage these approaches with domain specific languages which provide a better separation of concern to allow an architect specifying QoS and behavioral constraints, and associated mechanisms over parts of their compositions [6, 4, 5].

Few works make the transition to run-time usage of QoS data, even though the variations in QoS as well as failures become issues in SOA, paving the way to adaptive business processes. No doubt that QoS management in adaptive business processes presents numerous challenges, which prevented from a larger availability of such tools in currently deployed SOA platforms. The policy-based constructs proposed in [4] for example allow a business process to react to QoS variations and to enact the re-planning of service selection at runtime. These constructs are executed by a platform that cooperates in a non-intrusive way with a BPEL orchestration engine. However, re-planning is good when not done too often. When QoS variations or service failures (including disconnections) occur frequently, it is better to plan for adaptation rather than taking a repairing approach afterward.

3. MONITORING REQUIREMENTS

This section explicits the different requirements for a monitoring middleware component in the light of the QoS-aware late-binding use case deployed in the SemEUsE platform[1]. In this platform, the aim of the late-binding approach is to establish pairings between a process and services on the fly. This should overcome the limitation of many orchestration engines, in which services are selected upon requests to registries at the beginning of execution, and never reconsidered unless error handling is called upon a failure or SLA violation.

Whilst late-binding could be used to verify that candidate services observe their SLA before calling them, it is not its primary role in our context. This verification is left to other specific tools. In fact, QoS-aware late-binding really cares about selecting the best service according to their current QoS, and on the way, caters for failures by preferring services where QoS data are available to others. Coarsely speaking, each service invocation point is equipped with a capability to point several candidate services, to subscribe to monitoring of these services and to a decision-making engine. This engine will consider the current QoS data of all candidate services, emanating from the monitoring middleware, in order to elect the one to be invoked, given service consumers preferences. From our late binding use case, we have provided a first list of requirements on a monitoring middleware for such QoS-based control. We now analyze these requirements.

Coherency of data. Every piece of data used in one decision must reflect the same state of the system. As data providers are potentially heterogeneous, it is hard for data consumers to reason on. The data collection component should then provide a way to quantify the quality of a given data set. In the specific case of late binding, as well as in more general cases, we consider that this quality is especially characterized by the obsolescence of data and the temporal correlation between multiple QoS dimensions.

Flexibility in data access. For a single feature or QoS requirement, each service provider could respond using its own "vocabulary" or measurement units (ex. multiple functionally-equivalent weather Web service providing temperature in either Fahrenheit or Celsius values). On top of that, a service consumer is very likely to simultaneously work with multiple service providers. To improve flexibility, words, monitoring should overcome discrepancies over syntactic and semantic aspects of collected data, while efficiently shielding its clients from such concerns. This is especially true in a late binding context, where the primary concern is on optimal service selection, independently of the apparent heterogeneity between a service request and the pool of available candidates.

Performance in data access. Decision-making on the client-side shall not suffer from delays in accessing data so that it has the slightest possible impact on business process performance. In the case of late binding, performance is deeply impacted by the time from late binding sends out a data query to it received query results. We therefore consider as crucial to optimize this metric, rather than the network overhead and computational overhead, which also affect overall performance. These latter metrics are affected by the number of monitored data sources, the number of queries, and the amount of generated data over a time unit.

It is also obvious that a system cannot perform well in both performance and constraints on the quality of information (QoI): design choices affect performance - QoI trade-offs. A monitoring service can be considered perfectly scalable when it can maintain low query response time, low resource consumption and high information quality, in the large number of monitored resources and a large number of queries.

Network usage optimization. Transmitting monitoring data shall have the slightest possible impact also on network performance. This aspect meets the previous requirement as trade-offs must also be realized between requested data and their associated qualities (freshness, correlation...).

Besides, in our web-based distributed context, some level of data protection, or security at large, should be considered. Indeed, the answer to a specific QoS monitoring request should be delivered only to its rightful consumer(s) by implementing, for example, some kind of secured identification protocol. By design, in SemEUsE, security is built upon the underlying framework, the PeTals Open Source ESB[2] and is not directly integrated in the monitoring or late-binding components.

4. SYSTEM DESIGN AND ARCHITECTURE

Facing the above requirements the M4ABP middleware component provides the following solutions. Data collected from several sources are timestamped with a global clock (with sufficient precision) and are temporally filtered to provide a set of data taken in the same narrow time window.

[1]http://www.semeuse.org

[2]http://petals.ow2.org/

Consequently, coherency is provided through **temporal filtering** (cf. Section 4.1). Required QoS data are matched with offered ones using equivalence relationships defined by QoS ontologies, so that flexibility is provided by **ontology-based access** (cf. Section 4.2). Data are transmitted to the client-side in push mode, and then buffered until the business process fires data requests. This ensures performance in data access through a form of **colocalization** (cf. Section 4.3). When several clients on the same node requires the same data from the same services, the middleware pushes the data and buffer it once for all of them. This **mutualization in data buffering** enables some optimization in network usage (cf. Section 4.4).

The system architecture of the monitoring service is shown in the figure 1. Service providers typically expose *data sources* which advertise a set of QoS dimensions representing current QoS levels of their services.

QoS-aware *late binding* queries QoS data of relevant services and executes its binding algorithm. The *monitoring service*, collects QoS data from data sources and then processes and stores this data. It provides a *data query interface* which allows late binding to retrieve monitored data as it needs. Considering that services are associated with SLA contracts defining, among other things, QoS meta-data, such as data source information. The monitoring service can be optimized by only observing those metrics relevant to SLAs.

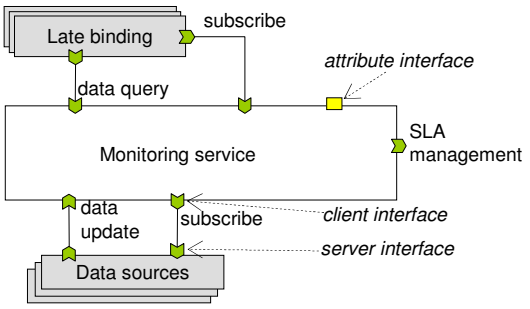

Figure 1: System architecture of M4ABP

When designing our monitoring framework, we followed a component-based approach in order to benefit from characteristics such as component self-containment and hierarchical decomposition. An overview of all components involved in the monitoring service is given in figure 2.

The following subsections discuss our response to the previously established monitoring requirements and constraints.

4.1 Temporal Filtering

Intuitively, the QoI-based approaches introduce the idea of how to quantify the degree of exhibited data in relation to the reality. There are various definitions concerning different concepts and metrics, which are mainly due to the different objectives of the system where they are used. We handle the coherency of data requirement by using a temporal filter over sequenced and timestamped QoS data. From a data consumer point of view, we distinguish two QoI dimensions of our temporal filter:

- *Age*: A QoS data is obsolescent during the time evolution. This measure specifies the minimum temporal precision of a QoS data . The age of a QoS data d_i at time t is defined as:

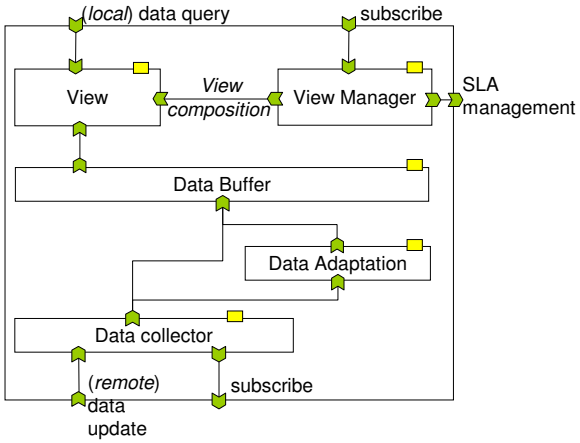

Figure 2: Component diagram of M4ABP

$$A(d_i, t) = t - d_i.ts$$

where $d_i.ts$ denotes timestamps of QoS data d_i.

- *Coherency*: QoS data are temporally correlated. This measure specifies the minimum temporal correlation in a tuple of QoS data. The coherency of a n-tuple QoS data $(d_1, ..., d_n)$ is defined as:

$$C(d_1, \ldots, d_n) = max(|d_i.ts - d_j.ts|), \forall i = 1 \ldots n, j = 1 \ldots n, i \neq j$$

A n-tuple $(d_1, ..., d_n)$ of n-QoS dimensions is conformed to the QoI constraints of a given couple $<age, coherency>$ at time t iif

$$A(d_i, t) \leq age, \forall i = 1 \ldots n \text{ and } C(d_1, \ldots, d_n) \leq coherency$$

The timestamps associating each data requires clock synchronization[3] between data providers.

A *View* encompasses a temporal filter to guarantee coherency of data before delivering to data consumers. This temporal filter takes a streaming data of QoS dimensions of services in the enclosed view and temporally produces a sequence of QoI-conformed tuples. Note that there are different strategies for implementing the temporal filter. For example, if there are multiple QoI-conformed tuples, then an average value or latest tuple may be applied.

4.2 Ontology-based access

QoS ontology, which is instructed from SLAs, allows defining semantic matching between QoS providers and QoS consumers in using their equivalence relationships. For example, an equivalent measurement units can be made equal using conversion functions provided by their ontology.

The flexibility in data access requirement is resolved by integrating ontology-based data adaptation inside the monitoring service. *Data Adaptation* aims at (1) transforming QoS ontology to executable data adapter and (2) executing data adapter to represent QoS data in consumer requirement. The former is done once after data consumers issue data queries. The later is executed during time evolution over sequenced QoS data before entering the *Data Buffers*.

[3]Network Time Protocol [14] can maintain time to within 10 milliseconds over the public Internet

4.3 Colocalization

We tackle the data *query response time* by colocalizing the monitoring service with its consumer late binding. This architectural strategy comes into *Data Buffers* which temporally buffer QoS data transmitted from data sources until late binding issues data queries. Conceptually, each *Data Buffer* is responsible to cache current QoS data of a data source.

When late binding subscribes its candidate services to be monitored, the *View Manager* gets a list of concerned data sources from associated SLA contracts, then configures *Data Collectors* which are in charge to synchronize remote data sources with the *Data Buffers*. The data transmission between remote data sources and *Data Collectors* is open to both push and pull modes. The push mode is encouraged due to its efficiency. Finally, the *View Manager* composes a *View* which includes a set of *Data Buffers* and *Data Collectors* relating to concerned services.

To answer the data query of late binding during decision making time while guaranteeing the QoI constraints, a *View* locally accesses QoS data from its enclosed *Data Buffers* and returns to late binding after applying temporal filter.

4.4 Mutualization in data buffering

Multiple clients interested in the same candidate service leads the intersection between multiple data queries. Mutualization in data buffering aims at establishing mutual communication channels and data processing spaces for common data sources.

Indeed, during *View* composition for a new QoS data subscription, the *View Manager* first looks up in the existent *Data Buffers* for each concerned data source. If one has already synchronized with that data source, then it reuses this buffer and associated communication channel rather than establishes new collectors and buffers. This simple algorithm allows several data consumers in the same node to share the same data from the same services. In consequence, the monitoring service can optimize resource consumption in both network bandwidth and computational resources.

5. EXPERIMENTAL EVALUATION

The prototype of M4ABP has been implemented to a large extent on top of COSMOS[4], a framework for managing context data in ubiquitous applications, and Fractal [7], a generic component model. It interacts with Orchestra[5], a WS-BPEL 2.0 orchestration engine which allows extending activities with late binding capabilities [9], and Dragon[6], an SOA governance solution which supports SLA management based on the standard WS-Agreement[7].

The prototype allows users to configure the synchronization between M4ABP and its QoS data sources. In our experiment, we use a parameter $p = age - d$ to set the period of synchronization, where *age* is the temporal filtering setting discussed in section 4.1 and d is the guaranteed network latency ($d = 650$ ms in our experiments). We now discuss preliminary results of our experiments with the late binding use case.

[4]http://picoforge.int-evry.fr/projects/svn/cosmos/
[5]http://orchestra.ow2.org
[6]http://dragon.ow2.org/
[7]https://forge.gridforum.org/projects/graap-wg/

Data acquisition with/without M4ABP. First, we evaluate the data query performance of M4ABP. We simulate data queries of business processes nodes equipped with QoS-aware late binding capabilities in two cases. In the first case, the nodes acquire QoS data directly from service providers without the use of M4ABP. In the second case, QoS data are acquired through the M4ABP component integrated into the orchestration engine. In both cases, the response time of QoS data queries is measured from the nodes while the number of queries per time unit changes gradually. The result is shown in the figure 3. Compared to direct data acquisition, M4ABP improves significantly QoS data query response time. This gain is mainly from network delay time.

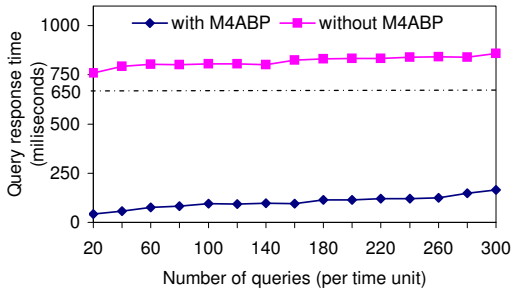

Figure 3: QoS data query response time

System performance. Next, we simulate how the temporal filtering and mutualization factors impact on the system performance. We subscribe 60 data queries, 5 data sources per query, and oscillate the period p, and then report the number of message exchanged over the network. The result of the experiment is given in figure 4. The different lines in the graph correspond to a rate of mutualized services of 0%, 20% and 50% between late binding queries. We observe that (1) the network overhead increases when the *age* constraint decreases, (2) the network overhead is significantly reduced when the rate of multiple clients required QoS data from the same services increases.

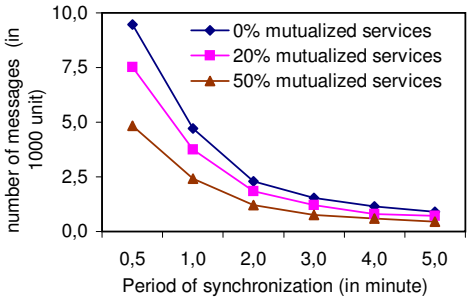

Figure 4: Network usage of monitoring service

6. RELATED WORK

Significant research work has focused on adaptive monitoring. A key idea is to carefully drop some tuples [3, 15] or to tune data sampling frequency [1] in order to reduce network bandwidth and processing load but at the expense of reducing accuracy of query answers and losing transient information. Other techniques have been proposed to address

the problem of minimizing bandwidth utilization or query latency for refreshing updates in presence of constraints on the age or accuracy of cached data [10, 11]. Pre-fetching or caching [15] techniques have also been proposed, like our data buffering system, to reduce data query response time. Our work has some similarities to these results. However, by employing a component-based approach, we provide an open infrastructure to design (self-)adaptive monitoring services addressing requirements such as coherency of data, performance and flexibility in data access.

Research in context-aware management systems is concerned with the acquisition of information to perceive situations and to adapt applications based on the recognized context. The quality of context information has been well studied in [8, 13, 2]. Many QoI related dimensions have been proposed including accuracy, freshness and consistency. The difference between these works and our system is that they focus on context information for a specific client. Our system provides support for performance optimization in data access by multiple clients, as represented by the various nodes of business processes, and intends to manage different QoI requirements on the same information.

As discussed in section 2.3, several works have proposed advanced Domain Specific Languages (DSL) to deal with QoS management concern and behavioral constraints over parts of Business Processes [6, 4, 5]. These works, however, focus on specific constraints and are limited in the measurements and metrics they can address. In contrast, our system supports flexibility in data access. Furthermore, our concepts have been presented using a late binding use case, but we believe they are sufficiently generic to capture a diverse range of adaptive data collection use cases. Future work are intended to confirm this.

7. CONCLUSION AND FUTURE WORK

In this paper, we have described M4ABP (Monitoring for Adaptive Business Process), a middleware component for monitoring services and delivering timely and coherent monitoring data to business processes using them in runtime decision-making settings. The proposed system provides several complementary features: temporal filtering ensures data coherency, ontology-based access enables flexibility, colocalization increases performance in data access, and mutualization in data buffering allows for optimization in network usage.

Preliminary experiments with the late binding use case have been illustrated. They show that the requirements have been met with good performance as well as resource efficiency.

Regarding future work, a short term goal is to cross-validate current requirements with more use cases, taking into account different scenarios and more QoI dimensions, as well as stressing the genericity of the proposed middleware component. Moreover, QoI should become part of a full-fledged monitoring request DSL, and the monitoring component shall be able to accommodate requests for different QoI on the same data without loosing on data mutualization. In the long term, we plan to tackle scalability issues by providing self-adaptive capabilities to M4ABP itself and by enabling several monitoring components to be distributed.

8. REFERENCES

[1] S. Agarwala, Y. Chen, D. Milojicic, and K. Schwan. Qmon: Qos- and utility-aware monitoring in enterprise systems. In *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, June 2006.

[2] M. Anwar Hossain, P. Atrey, and A. El Saddik. Context-aware qoi computation in multi-sensor systems. In *Mobile Ad Hoc and Sensor Systems, 2008. MASS 2008. 5th IEEE International Conference on*, pages 736–741, 29 2008-Oct. 2 2008.

[3] B. Babcock, M. Datar, and R. Motwani. Load shedding for aggregation queries over data streams. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 350, Washington, DC, USA, 2004. IEEE Computer Society.

[4] F. Baligand, N. Rivierre, and T. Ledoux. A declarative approach for qos-aware web service compositions. *Lecture Notes in Computer Science*, 4749:422, 2007.

[5] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-time monitoring of instances and classes of web service compositions. In *Web Services, 2006. ICWS '06. International Conference on*, pages 63–71, Sept. 2006.

[6] L. Baresi, S. Guinea, and P. Plebani. WS-Policy for service monitoring. *Lecture Notes in Computer Science*, 3811:72, 2006.

[7] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper.*, 36(11-12):1257–1284, 2006.

[8] T. Buchholz, A. Küpper, and M. Schiffers. Quality of context information: What it is and why we need it. In *In Proceedings of the 10th HPŰOVUA Workshop, 2003, Geneva, Switzerland*, 2003.

[9] P. Chatel. Decision framework. Technical report, Laboratoire d'informatique de Paris 6, 2009.

[10] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 117–128, New York, NY, USA, 2000. ACM.

[11] N. Jain, P. Yalagandula, M. Dahlin, and Y. Zhang. Self-tuning, bandwidth-aware monitoring for dynamic data streams. In *ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 114–125, Washington, DC, USA, 2009. IEEE Computer Society.

[12] Z. Liangzhao, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for Web services composition. *IEEE transactions on Software Engineering*, 30(5):311–327, 2004.

[13] A. Manzoor, H.-L. Truong, and S. Dustdar. On the evaluation of quality of context. In *EuroSSC '08: Proceedings of the 3rd European Conference on Smart Sensing and Context*, pages 140–153, Berlin, Heidelberg, 2008. Springer-Verlag.

[14] D. Mills. Simple network time protocol (sntp) version 4 for ipv4, ipv6 and osi, 2006.

[15] X. Zhang, J. Freschl, and J. Schopf. A performance study of monitoring and information services for

distributed systems. In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pages 270–281, June 2003.