A QoI-aware Framework for Adaptive Monitoring

Bao Le Duc∗, Philippe Collet‡, Jacques Malenfant† and Nicolas Rivierre∗
∗Orange Labs, Issy les Moulineaux, France
Email: {bao.leduc, nicolas.rivierre}@orange-ftgroup.com
†Université de Nice Sophia Antipolis, CNRS, UMR 6070 I3S, Sophia Antipolis, France
Email: Philippe.Collet@unice.fr
‡Université Pierre et Marie Curie-Paris 6, CNRS, UMR 7606 LIP6, Paris, France
Email: Jacques.Malenfant@lip6.fr

Abstract—Monitoring application services becomes more and more a transverse key activity in information systems. Beyond traditional system administration and load control, new activities such as autonomic management and decision making systems raise the stakes over monitoring requirements. In this paper, we present ADAMO, an adaptive monitoring framework that tackles different quality of information (QoI)-aware data queries over dynamic data streams and transform them into probe configuration settings under resource constraints. The framework relies on a constraint-solving approach as well as on a component-based approach in order to provide static and dynamic mechanisms with flexible data access for multiple clients with different QoI needs, as well as generation and configuration of QoS and QoI handling components. The monitoring framework also adapts to resource constraints.

Keywords—Monitoring, Adaptive systems, Quality of information, Component framework

I. INTRODUCTION

As distributed and pervasive systems are now deployed everywhere with 24/7 availability constraints, monitoring becomes more and more a transverse key activity in enterprise computing. Beyond traditional system administration and load control, new activities increasingly require automated management of the systems, raising the stakes over monitoring requirements. Specific tasks such as scheduling, resource allocation and problem diagnosis make their decisions upon the online and continuous monitoring of the services, systems and infrastructures. Besides, autonomic management and decision making systems are now organized around Service Level Agreements referring to some Quality of Service (QoS) criteria. As large QoS variations are easily observable by clients when calling distant applications and services, there is also a large variation in the monitoring requirements, in terms of the types of monitoring data to be acquired, their lifespan, precision and granularity. This is generally referred as Quality of Information (QoI), i.e., an expression of the properties required from the monitored QoS data [1].

Moreover, deployment contexts have evolved in size and complexity, from intra-enterprise Service-Oriented Architectures (SOA) principles with low-latency network to large-scale inter-enterprise infrastructures with high latency, and finally to pervasive systems with dynamic contexts. Monitoring a distributed system involves extracting information among the deployed processes and their interactions, collecting it efficiently and making them available to the interested users in an appropriate format. The distributed context makes the monitoring activity inherently more complex than the more traditional centralized one, as it forces to handle several control flows, communication delays between nodes, nondeterministic event ordering and an extensive behavioral alteration on the observed system [2].

These challenges are hardly addressed by current monitoring systems. In a SOA context, prior works show that behavioral and basic QoS constraints can be expressed and monitored at runtime [3], [4], but with no QoI or only some implicit ones like statistics on QoS [5]. A monitoring system must currently provide several information flows to multiple clients, with different QoI requests, everything being dynamically reconfigurable. Finally, the monitoring system, being constantly operational, is itself subject to constraints on the resources it consumes to provide its services. Consequently, designing and deploying monitoring systems that are well-adapted to such requirements now become a complex and tedious activity for software architects and system administrators. Automation of this process is clearly needed. Recent works focus on QoI and adaptive monitoring for context-aware computing, data stream processing or transactional systems [6], [7], [8], but no monitoring system is currently adapted to all (changing) requirements together.

In this paper, we present ADAMO, an adaptive monitoring framework that tackles different QoI-aware data queries over dynamic data streams, transform them into probe configurations settings under resource constraints. This process relies on a constraint-solving approach. The framework also factors out the common structure and behavior of monitoring systems so that they can be reusable and extensible. To do so, it leverages component-based techniques so that a common base architecture is provided as an assembly of interacting components. Different parts of the architecture are then configurable, or can be partly generated from high-level descriptions of the monitoring requirements. ADAMO thus aims at providing solutions for i) flexible access to dynamic
Moreover, as the authority structure is typically hierarchical, different commanding officers may require data with different QoI, depending upon their rank or their relation to the monitored entity. Higher rank officer have less stringent requirements, typically an order of magnitude less, when building aggregated global situation reports, while occasional requesters of a particular mean may be satisfied with less up-to-date data.

The overall goal of a monitoring framework as ADAMO is to build, configure and deploy the necessary components between the application and the sensors, and configure these so to match the required QoI while respecting resource and deployment constraints. If the problem appears to be overconstrained, utilities can be assigned to the different data so to guide the tradeoffs between them when computing their transmission frequencies (see Figure 1).

Figure 1. Query examples. In the left, a higher rank officer queries rescue and transportation team positions with less stringent QoI requirements; whereas in the right, lower rank officers query on resources they command with strict QoI requirements.

B. On Adaptive Monitoring

Distributed monitoring is intrinsically a complex activity and current large scale architectures of distributed systems impose new requirements and strong constraints on monitoring. Consumers of the monitoring system are now applications and not only human users. Applications act as multiple clients, requesting for very different QoS data with specific QoI constraints on each of them. Moreover these applications make and change their queries dynamically, adding a new stringent requirement on the monitoring system. On the other side, data sources are also very varied and may be at different locations of the distributed system, thus impacting bandwidth consumption when data are larger.
or transmission rate higher. Finally, system administrators
needs to deploy the monitoring system under resource
constraints, so that the overall resource consumption of the
monitoring system itself is mastered during execution. Many
research approaches have been proposed, to handle data
collection from sources in context-aware computing, to add
QoI capabilities on existing QoS monitoring systems, to
provide adaptive monitoring infrastructures or also to build
full adaptive systems.

To the best of our knowledge, no monitoring system is
currently adapted to all these requirements. Nevertheless
some systems provide really powerful solutions to one or
several specific features of a monitoring system, from adap-
tive capabilities to QoI awareness or consumption regulation.
We thus advocate a framework approach so that generic
parts of a monitoring system can be more easily reused and
extended and that well-adapted monitoring systems can be
instantiated for specific needs. Consequently the framework
must be able to deal with multiple clients needing flexible
and dynamically reconfigurable access to dynamic data
streams with different QoI needs, and to provide automatic
configuration of all monitoring entities and data sources so
that QoI and resource constraints are taken into account.

C. Related Work

This section presents an overview of the research area on
adaptive monitoring and QoI control in software systems.

Context-aware systems are concerned with QoI to per-
ceive situations and adapt applications based on the rec-
ognized context. Quality of Context is well studied in [1],
[9], [10] where many dimensions are proposed, including
precision, freshness and consistency of the monitored data.
These works, however, do not address the architecture of
context-aware systems or the problem of maximizing QoI
over a set of constraints.

Among the different works on context-aware management
systems, Conan et al. [11] propose an architecture based on
components (called context nodes) that are responsible to
produce higher level context information from data gathered
at lower architectural layers. The authors describe several
patterns to compose the individual context nodes in order to
implement the desired logic of a context-aware application.
In [6], they show how to extend their approach to support
Quality of Context (QoC) by using a specialized component
to filter and evaluate QoC from collected information. Al-
though they do not address the problem of maximizing QoI
in overloaded situations their architecture is highly modular
and extensible, and allows to introduce controlled tradeoffs
between QoI requirements and resource consumption.

Poladian et al. [12], [13] focus on adaptive systems based
on multiple concurrent applications running on local com-
puting devices with limited memory, CPU and bandwidth.
They propose an analytical model and an efficient algorithm
to decide how to allocate scarce resources to applications,
and how to set the quality parameters of each application to
best satisfy user and supplier preferences. Their approach
fits well into the framework proposed here to adjust the
monitoring to the current conditions, given QoI objectives.

Data stream processing systems such as sensor networks
or financial services are concerned with the problem of
saving network or compute resource to deliver accurate in-
formation. Babcock et al. [14] propose a load shedding tech-
nique for continuous monitoring queries over data streams.
The key idea is to carefully drop some tuples in order to
reduce bandwidth and processing in overloaded situations.
The authors formalize load shedding as an optimization
problem with the goal of minimizing query inaccuracy
within the limits imposed by resource constraints. Tatbul et
al. [15] extend this approach for distributed stream process-
ing systems. These works propose sophisticated algorithms
and optimization techniques. However they do not address
the design of the monitoring framework to implement them
in a modular and flexible way, or focus on scalability issues
in large-scale distributed stream processing systems [7].

Among the different works on predicting runtime mali-
functions in software systems, Munawar et al. [8], [16]
propose a new approach to monitor multi-tier transaction
systems at a minimal level in normal condition and adap-
tively increase monitoring if a health problem is suspected.
Their approach uses relationships between the monitored
data in the form of regression models to determine normal
operation and areas that need more monitoring in the event
of anomalies. Their work fits well in presence of multiple
metrics to dynamically adjust monitoring to the current con-
dition but focuses on health prediction and doesn’t consider
QoI requirements such as age of the monitoring data.

III. A QoI Model for Adaptive Monitoring

This section presents ADAMO’s QoI model, formalizing
data sources, monitoring queries and system resources. The
model leverages constraint solving to find appropriate fre-
quencies to configure data sources according to clients needs
and resource constraints.

A. A Model for Adaptive Monitoring

Consumers send ADAMO QoI-aware monitoring queries
and receive data streams as result. ADAMO hence addresses
QoI by processing queries in such a way to automatically
translate the requested QoI and resource constraints into data
source configurations.

Definition (Data source). A data source $s$ is a triple
$(\tau_s, \Phi_s, \Pi_s)$ where $\tau_s$ is a data source identifier, $\Phi_s =
(\phi_{s,1}, ..., \phi_{s,n})$ is a data stream generated by $s$, and $\Pi_s$ is a
set of constraints on data source properties.

In this model, monitored values are defined as indepen-
dent data sources, even though some may report to the
same physical entity. The data stream consists of sequences
of data produced in temporal order by some measurement unit or probe. Each element $\phi_i$ contains a data value and a time-stamp representing when the value is generated to enforce QoI constraints. Constraints on data source properties express possible configuration settings, e.g., interrogation mode (push/pull), sampling frequency... The latter are also used to regulate the monitoring and can be assigned a configuration value among the admissible ones for each data source through a configuration $C_s$ imposed at run-time. Sampling frequencies act as filters on the raw data stream to pick the values that will be transmitted to clients by the monitoring framework. It should be noted that $\pi_s$ denotes below the set of properties constrained by a data source.

Example. The data source for the remaining autonomy of helicopter 1 is $ha_1 = (h1_{aut}, ((120, t_0), (118, t_1), ...), \{f_{ha_1} \in (0.5, 1, 2), msgSize_{ha_1} = 1\})$, where the remaining autonomy in the stream is expressed in minutes timestamped with $t_0$, $t_1$, ... (unspecified here), and where the frequency $f_{ha_1}$ and message size properties are constrained to be 0.5, 1 or 2 data per minute and exactly 1kb respectively. The set of data source properties $\pi_{ha_1} = \{f_{ha_1}, msgSize_{ha_1}\}$.

Definition (QoI-aware monitoring query). A query $q$ is a couple $(\pi_q, S_Q)$ where $\pi_q = (t_q, 1, ..., t_q, n)$ is a set of sources from which the consumer wants to get data, and $S_Q$ is a set of QoI constraints imposed by the consumer on all of the data sources in $t_q$.

A query specifies the need of a consumer in the reception of tuples of data (required data sources) under the given QoI constraints. $\pi_q$ denotes below the set of QoI properties constrained by $S_Q$. Currently, ADAMO addresses two different QoI properties: age and coherency.

Definition (Age and coherency constraints). An age constraint imposes a maximal delay between the production of a data by a source and its reception by the consumer. A coherency constraint imposes a maximal delay between any pair of data for the requested tuple to be considered as valid.

Example. The aerial means officer needs helicopter 1 and 2 position and remaining autonomy not older than 2 minutes and a coherency of 30 seconds.

The query is $\{h1_{pos}, h2_{pos}, h2_{aut}, h1_{aut}, \{age \leq 2, coherency \leq 0.5\}\}$. The set $\pi_q$ of monitoring properties constrained by the query is $\{age, coherency\}$.

Definition (Resource). A resource $r$ is a tuple $(i_r, \Pi_r, v_r, \oplus_r)$ where

- $i_r$ is a resource identifier,
- $\Pi_r$ is a list of data source properties impacting the consumption of the resource $r$,
- $v_r$ is a function of the properties $\Pi_r$ giving the consumption of the resource $r$ by a data source $s$ given the settings of its properties $\Pi_s$, and
- $\oplus_r$ is an aggregation function to combine the consumptions of data sources into an estimation of the global resource consumption of the monitoring system.

Definition (Resource constraints). Let $R$ be a set of resources used by the monitoring, $C_R$ is a set of constraints put on these resources.

System resources used by the monitoring encompass bandwidth, CPU, memory... Each of the resources uses available data source properties expressing the consumption of that resource when delivering data to consumers to get the overall estimation of their consumption by the monitoring system in a given configuration of the data sources.

Example. Consider the case where the bandwidth used by the delivery of monitoring data must be kept under 10% of the total bandwidth of the network. The bandwidth resource is defined by $b = (\text{bandwidth}, \{f, msgSize\}, v_b, \text{sum})$ where $v_b(f, msgSize) = f \times msgSize$ and sum simply says that bandwidth consumptions of data sources are summed to get the overall bandwidth consumption of the monitoring. If the total bandwidth is $TB$, the constraint is $C_R = \{\text{bandwidth} \leq 0.1TB\}$.

We denote $Q$ a set of monitoring queries and $S$ a set of data sources. $S_Q$ is the subset of $S$ used by $Q$. The principal challenge for adaptive monitoring is to find a data source configuration $C_{S_Q}$ satisfying a given set of queries $Q$ under the resource constraints $C_R$.

B. QoI-aware Control Capability

The above model is generic and open to extend to new data sources, properties, resource and constraints. As QoI is concerned, ADAMO nevertheless considers age and coherency as primary properties. This section shows how the constraints on these are dealt with in the current implementation of ADAMO. The first lesson learned is that each kind of QoI requires a specific processing, hence extensibility of the platform with regards to QoI and how it is handled is mandatory. To put forward this extensibility requirement, we now introduce an approach to the model resolution in two contexts. First, we look at a resource unconstrained case, and then we add the resource constraints.

In the first context, the system is assumed to have sufficient resources in order to process all data queries. In this case, for any $s$, $C_{S_Q}$ is a configuration that satisfies highest QoI requirements among the set of queries $Q_s$ using $s$. In the second context, resources are constrained, computing $C_{S_Q}$ amounts to find a trade-off between QoI requirements and resource constraints. This trade-off problem varies upon usage contexts as well as how QoI impacts on consumers. For example, when the system has not enough resources, a simple approach is to reduce QoI equally for
Definition (CSP formulation, unconstrained case). Let Q be a set of monitoring queries and S_q the set of resources required by Q, the CSP formulation of the problem is:

1) The set of variables of the problem is
\[ \bigcup_{q \in Q} \pi_q \cup \bigcup_{s \in S_q} \pi_s \]

2) \( \forall s \in S_q \), the constraints \( \Pi_s \) are added to the CSP.

3) \( \forall q \in Q \), let \( a_q \leq v \) be the age constrain of \( q \), then the constraints \( a_q \leq v \), and \( \forall s \in S_q \), \( f_s \geq 1/a_q \) are added to the CSP.

4) \( \forall q \in Q \), let \( c_q \leq v \in \Pi_q \) be the coherency constraint of \( q \), then the constraints \( c_q \leq v \), and \( \forall s \in S_q \), \( f_s \geq 1/c_q \) are added to the CSP.

The CSP obtained using the above definition may not have only one solution, as multiple frequencies for data source may match the desired age and coherency constraints of the queries. In this case, we choose the smallest frequencies in the sets of values satisfying all of the constraints.

Example. Consider ten data sources and three queries from the flood fighting scenario described above. Data sources are position and remaining autonomy for helicopters (hp, ha), position of rescue teams (rp), and position and remaining capacity for transportation teams (tp, tc). Each query \( q_1, q_2, q_3 \) specifies the sources from which the consumer wants to get data and the QoI constraints on age \( a_q \) and coherency \( c_q \) imposed by the consumer on all of these data sources.

The set of constraints of the CSP includes all of the domain constraints of the ten data sources as well as the QoI property constraints of the three queries, to which are added the following constraints linking QoI to data source properties:

\[ q_1 : f_{hp1} \geq 1/10, f_{fp1} \geq 1/2 \]
\[ q_1 : f_{hp2} \geq 1/10, f_{fp2} \geq 1/2 \]
\[ q_1 : f_{hp3} \geq 1/10, f_{fp3} \geq 1/2 \]
\[ q_1 : f_{hp4} \geq 1/10, f_{fp4} \geq 1/2 \]
\[ q_1 : f_{hp5} \geq 1/10, f_{fp5} \geq 1/2 \]
\[ q_2 : f_{hp1} \geq 1/2, f_{fp1} \geq 2 \]
\[ q_2 : f_{hp2} \geq 1/2, f_{fp2} \geq 2 \]
\[ q_2 : f_{hp3} \geq 1/2, f_{fp3} \geq 2 \]
\[ q_2 : f_{hp4} \geq 1/2, f_{fp4} \geq 2 \]
\[ q_2 : f_{hp5} \geq 1/2, f_{fp5} \geq 2 \]

Taking the minimal frequencies satisfying these constraints, data sources of helicopters and rescue teams will have their frequencies set to 2 data per minute, while transportation teams will be set to 1 datum every 2 minutes.
monitoring. At first sight, we just need to add the following constraint (to simplify the notation, assume \( \oplus_r \) is a sum) to the constraint system elaborated for the unconstrained case:

\[
\sum_{s \in S_Q} v_r(\Pi_r|_s) \leq A
\]

where \( \Pi_r|_s \) are the properties that \( r \) depends upon for the data source \( s \). The problem is then to find a configuration \( C_{S,Q} \) that satisfies not only the age and coherency constraints, but also this resource constraint.

**Example.** For the case of the bandwidth constrained not to pass over 10% of the total bandwidth \( TB \), we have \( \Pi_r = \{ f, msg\text{Size} \} \), \( v(\Pi_r) = f \times msg\text{Size} \), and the aggregation function is a sum, hence the resource constraint becomes:

\[
\sum_{s \in S_Q} f_s \times msg\text{Size}_s \leq 0.1 \times TB
\]

(1)

However, the system being constrained in a new way, this can be considered to change the nature of QoI control problem. Indeed, as the resource constraint may impair the satisfaction of the age and coherency constraints of some queries, the user should be able to express preferences among its queries so to concentrate the resource on the most important queries and lower, if necessary, the requirements of the less important ones.

In order to allow the user to express his/her preferences over QoI properties, the query definition is extended with a set of utility functions \( U_q \) that contains one utility function \( \mu_{q,p} \) for each monitoring property \( p \in \pi_q \). \( \mu_{q,p} \) maps configurations \( C_{S,q} \) of data sources used in \( q \) to a utility value in \( \mathbb{R} \). These utilities are combined to get the total utility of a configuration as follows:

\[
U_Q(C_{S,Q}) = \sum_{q \in Q} \prod_{p \in \pi_q} \mu_{q,p}(C_{S,q})
\]

(2)

In this new setting, age and coherency constraints are now seen as minimal requirements, and the problem becomes to find a configuration \( C_{S,Q} \) that maximizes the above utility under the age, coherency and resource constraints.

**Example.** In the bandwidth example, all of the queries have the same set of monitoring properties, \( \{ \text{age, coherency} \} \), so the above global utility becomes for this example:

\[
\mu_{q_1,a}(C_{S,Q}) \times \mu_{q_1,c}(C_{S,Q}) + \\
\mu_{q_2,a}(C_{S,Q}) \times \mu_{q_2,c}(C_{S,Q}) + \\
\mu_{q_3,a}(C_{S,Q}) \times \mu_{q_3,c}(C_{S,Q})
\]

(3)

These utility functions use the same computation to get the age and the coherency of a query, i.e., the age of the query \( q \) is given by the minimal frequency among its data sources imposed by the configuration (it is the same for the coherence):

\[
a_q = \frac{1}{\min_{s \in \Omega_q} C_{S,q}(f_s)}
\]

Adding utility functions \( \mu_{q,a} \) and \( \mu_{q,c} \) for each query \( q_1, q_2 \) and \( q_3 \) provides for an optimization problem where the objective is to maximize the equation 3 under the previous age and coherency constraints and the above resource constraint.

Consider the ten data sources and three queries of the previous example, If the total bandwidth \( TB = 130\text{kb}/s \), the monitoring bandwidth should not pass over 13kb/s, from the resource constraint specified in equation (1). In overloaded situation, the QoI requirements are now expressed as utility functions (see Figure 2) to concentrate the resources on the most important queries. In this case, the utility associated to query \( q_1 \) expresses less stringent QoI requirements on coherency and age than \( q_2 \) and \( q_3 \). The following configuration of frequencies maximizes\(^2\) the global utility specified in equation (3), under the age, coherency and resource constraints.

\[
\begin{align*}
f_{p_1} &= 2 & f_{p_1} &= 2 \\
f_{p_2} &= 2 & f_{p_2} &= 2 \\
f_{p_3} &= 2 & f_{p_3} &= 2 \\
f_{p_1} &= 1/5 & f_{p_1} &= 1/5 \\
f_{p_2} &= 1/5 & f_{p_2} &= 1/5
\end{align*}
\]

This result shows that the utility leads the monitoring to reduce the QoI for transportation teams, and hence bandwidth for their data sources, since they are used only by the query \( q_1 \) which has less stringent QoI requirements.

![utility](utility.png)

**Figure 2.** QoI requirements on coherency (\( \mu_{q_1,c} \)) and age (\( \mu_{q_1,a} \)), expressed as utility functions for queries \( q_1, q_2, q_3 \).

### IV. ADAMO COMPONENT-BASED ARCHITECTURE

We now use the model to present the ADAMO architecture and introduce the various abstractions that enforce QoI needs. We then describe how the framework is implemented and discuss our ongoing work to support reusability and extensibility, notably by using appropriate design patterns.

#### A. ADAMO principles

The main goal of ADAMO is to produce centralized monitoring systems that can be located in given points of a distributed architecture\(^3\). The basic operation supported by

\(^2\)We use Gecode (http://www.gecode.org), a constraint programming toolkit to solve this problem. In this example, 10 data sources and 3 possible frequencies for each data sources generate \( 3^{10} = 59049 \) configurations.

\(^3\)Mastering the deployment of several distributed ADAMO entities is part of future work (see Section V).
ADAMO is to gather data from distributed sources and store them in a buffer system. These data are then processed prior to being delivered to consumers so that different properties are enforced on requested QoI while obeying to resource constraints. In order to provide a reusable and extensible adaptive monitoring framework, the ADAMO architecture must factor out the common structure and behavior from the monitoring specific parts. Doing so results in software artifacts that can be reused with fewer efforts to design and implement a specific monitoring service. ADAMO thus rests on a component-based approach. We detail the resulting design as well as the abstractions made by the framework in the following paragraphs.

At the highest level, the component-based approach allows for structuring the overall architecture of a monitoring system. Figure 3 outlines the main components with their interactions among them and with the external roles described in section II-B.

The application represents consumers of the monitoring system, the query analyzer acts as the front-end to process different kinds of QoI-aware data queries. For example, applications may fire a batch of queries against the monitoring component and then wait for streaming results, or on the other hand they may submit a query on-demand (in our illustration, before the displacement of inhabitants process starts). The query analyzer thus handles queries, initiates data inquiry which may derive from a composite dimension and intersect between multiple consumers, identifies consumer’s QoI constraints, and stores them into query repository for further reasoning. QoI control then finds an appropriate configuration for any data inquiry. Based on the configuration set, data inquiry establishes an inquiry strategy to access remote data sources. The inquired data stream is cached in local data buffers. Further data processing view (called view for short) such as QoI filtering or data transformation is realized before delivering final data to the consuming applications, in push or pull mode.

B. Abstraction of the Framework

In ADAMO various abstraction points are available to clarify domain intents and reduce implementation efforts. This allows software architects to focus on solving a problem without being concerned about less relevant lower level details. In the framework, each component represents a level of abstraction that can be extended to specific adaptive monitoring requirements. For example, QoI control can be extended in order to adopt a new trade-off algorithm taking into account coherency and some resource constraints.

1) Query Analyzer: A query analyzer is in charge of handling and processing data queries. As modeled in III-A, a query consists of two specific parts in which (a) a list of dimensions is used to identify data sources, configure data buffers and views, (b) QoI constraints are used to configure ADAMO, especially data inquiry properties. ADAMO then supports two ways to submit a query: static and incremental. In the static mode all queries are submitted to the monitoring service once and for all. A set of data sources \( S_Q \) is then derived from the set of queries \( Q \). The incremental mode is obviously more complex as queries can be subscribed and removed at runtime. This requires some specific support on existing queries so that data inquiry processes are correctly deactivated. A new set of data sources is then derived from the pre-existent ones and the new query: \( S_Q = f(S_Q, q) \). In both cases, when multiple clients refer to the same data source, the query analyzer makes the necessary adjustments to converge to a single data inquiry, so that duplicated remote data transmissions from data sources are avoided.

Due to the necessary knowledge on data queries for both the query analyzer and the QoI control component, information related to queries, data sources and their relationships is indexed and stored in a query repository.

2) Data Inquiry: The data inquiry component establishes a data inquiry protocol, based on a given configuration \( C_S \) assigned to data source properties. Data source properties include frequency, message size, data transmission mode (push/pull), but also inquiry mode (batching multiple samples, summary techniques). In practice, message size and data transmission are usually chosen at design time while inquiry frequency is used to regulate data transmission.

3) Data Processing: A data processing view produces high-level abstract information from some low-level raw data. It also provide the data to the consuming applications according to the protocol of their choice (pull or push mode). In most cases, raw data sensed from environment may be meaningless for consuming applications or some measurements are not good enough for a given QoI request. ADAMO thus distinguishes two types of data processors.

An Aggregator aggregates data from different sources to reproduce a new data dimension. A particular case is a translator that transforms data from a unique source. For example, the distance delivered by a data source measured in mile can be converted into kilometer.

A QoI-based processor aims at filtering or evaluating QoI for a given data set. In our motivating example, the
view representing query \( q_2 \) (cf. III-B) should filter out position and remaining autonomy of helicopter tuples if they are not coherent in the timing window of 30s. Figure 4 depicts a temporal filter of \(<\text{age, coherency}> = <2 \text{ minutes, } 1/2 \text{ minute}>>\) that uses a sliding window to select the first coherent tuple of two sources.

In both cases, data processor is fed by data buffers. Multiple consumers hence can share their mutual data sources.

![Figure 4. Example - a temporal filter implements QoI based processor.](image)

4) QoI Control: A QoI control component is used to find a configuration of the monitoring service satisfying QoI requirements and resource constraints. Three distinct tasks are associated to this component. First, it gathers inputs to feed the QoI control algorithms described in section III-B. These inputs consist of knowledge from the query repository (the current set of QoI-aware queries \( Q \)), the subset of data sources \( S_Q \) used by \( Q \)), and resource settings specified by an administrator (the set of resource constraints \( C_R \)). These inputs may vary according to how the QoI control issue is handled. Secondly, it executes the QoI control algorithm to find the data source configuration \( C_{S_Q} \) satisfying the current set of queries \( Q \) under the resource constraints \( C_R \). This algorithm can be changed at run-time thanks to dynamic reconfiguration of components [18]. Finally, it delivers \( C_{S_Q} \) to the data inquiry component, in charge of applying dynamically this new configuration into the monitoring system.

The configuration of QoI control is typically executed when a new query is submitted. But executing this on-demand is potentially costly. To tackle this issue, ADAMO proposes two strategies for the administrator. First, it proposes two reconfigurations modes:

- reconfigure all data sources or reconfigure only inactive data sources. Secondly, in ADAMO, it is possible to specify when the reconfiguration are effectively run, based on time-interval (e.g., every 5 minutes) or query unit interval (e.g., every 3 query updates).

C. Implementation and Reuse of the Framework

The prototype of ADAMO has been implemented to a large extent on top of COSMOS [11], a probe framework for managing context data in ubiquitous applications. This enables the framework to easily reuse many data sources through dedicated wrappers, which are also easy to write or to partly generate. As for its component model, ADAMO relies on the Fractal [19] generic component model, which notably provides hierarchical decomposition of components at any level, explicit definitions of required and provided interfaces, as well as full capabilities for dynamic reconfigurations. Building on this rich component model enables software architects to more easily reuse and/or tailor components inside the framework.

Besides several design patterns are used to improve the design, reuse and consistency of the ADAMO architecture. A typical monitoring system instantiated from ADAMO should implement components by extending the abstraction mechanisms described in the previous section. At the highest level, these components must be consistent with each other and the Abstract Factory pattern is then used to ensure this consistency constraint. For example, to tackle a new QoI concept as first-class constraint such as data precision of query results, one should ensure that the concept is taken into account by every concerned monitoring entities, i.e., query analyzer, data inquiry, view and QoI control. Listing 1 shows an excerpt of Abstract Monitoring Factory interface.

```
Listing 1. Abstract Monitoring Factory

public interface AbstractMonitoringFactory {
    public QueryAnalyzer createQueryAnalyzer();
    public DataInquiry createDataInquiry();
    public View createView();
    public QoIControl createQoIControl();
}
```

As building a monitoring service implies creating a set of ADAMO components, the Composite pattern is reused in the architecture to support two specific compositions. The composition capability of View extends the composition provided by the COSMOS probe system so that a data access point dedicated to a data query is assembled from data inquiries, data buffers and data processors. ADAMO composition adds query analyzer, query repository, and QoI control into each ADAMO instance to enable the QoI control capabilities. These compositions rely on Fractal ADL [19]. Figure 5 illustrates this organization with three queries described in III-B.

```
Figure 5. Composition example.
```

As multiple consumers may be interested in the same source, data transmission is improved by creating a single transmission channel between ADAMO and every needed
data source. While the QoI Control component configures the mutual data inquiry to satisfy different requests, the Flyweight pattern enables the reification of the view composition so to data buffers are shared between consumers. As illustrated in Figure 5, data buffers of rescue teams $rP_1$ and $rP_2$ are included in both views of queries $q_1$ and $q_3$.

Finally, the Singleton pattern ensures that each ADAMO instance has only one query analyzer, query repository and QoI control. The Query analyzer then provides a global point of access to consumers (acting as Facade pattern), whereas the latter two maintain coherency on monitoring constraints and monitoring algorithms.

V. CONCLUSION AND FUTURE WORK

In this paper, we have presented ADAMO, an adaptive monitoring framework that tackles different QoI-aware data queries over dynamic data streams. The proposed system relies on a constraint-solving approach and component-based techniques so that common structures and behaviors of monitoring systems can be more easily reusable and extensible. We have shown how it provides solutions to handle multiple clients with different QoI requirements, transformation of QoI needs into probe configuration settings, control trade-offs between QoI needs and resource constraints, and management of data queries in a static or incremental way.

Regarding future work, short term goals are to evaluate the effectiveness of the proposed framework with stress/load testing and to validate its genericity with different scenarios and more QoI dimensions, including precision and significance as proposed in [1], [9]. In the long term, we plan to tackle scalability issues by providing self-regulation capabilities and by enabling several ADAMO monitoring systems to be distributed.

ACKNOWLEDGMENT

The research was partly funded by the French National Research Agency (ANR) through the SemEUsE research project. See http://www.semeuse.org.

REFERENCES