RI 07008, 5 November 2007 Computer Science

IBM Research Report

Data Tagging Architecture for System Monitoring in Dynamic Environments

Bharat Krishnamurthy
IBM Research Division
IBM India Research Lab
4, Block C, Institutional Area, Vasant Kunj
New Delhi - 110070, India.

Anindya Neogi
IBM Research Division
IBM India Research Lab
4, Block C, Institutional Area, Vasant Kunj
New Delhi - 110070, India.

Bikram Sengupta
IBM Research Division
IBM India Research Lab
4, Block C, Institutional Area, Vasant Kunj
New Delhi - 110070, India.

Raghavendra Singh
IBM Research Division
IBM India Research Lab
4, Block C, Institutional Area, Vasant Kunj
New Delhi - 110070, India.

IBM Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

Abstract—Large enterprise systems need continuous monitoring at infrastructure, application and business levels to detect and prevent problem situations. Traditionally, automated monitoring solutions are programmed once at setup based on a set of well-defined monitoring objectives and handed over to the operations team. Such solutions have underlying data models that are often complex and semantically rich but in stable environments, this complexity is generally hidden from the operations team, who only need to make minor configuration changes (e.g. setting thresholds) as and when required. However, the situation is now rapidly changing with enterprise data centers being subject to continuous transformations as new software, hardware and process components get deployed or updated. This puts an immense burden on monitoring activity because not only thousands of different parameters need to get monitored but the addition and modification of service level objectives (SLOs) may happen continuously.

We describe a monitoring system architecture which simplifies the task of authoring and managing SLOs in such dynamic and heterogeneous environments. At the heart of our approach is a lightweight and extensible data model that is derived from more complex configuration models, so as to only expose data relevant for monitoring to the operations team. Simple string-tags derived from this model are then used to label SLOs and associated data streams. The approach localizes programming to the data-sensor layer and makes authoring simpler than the specification of objects in an alternate richer but complex object-oriented representation. We also describe a tag-driven real-time visualization tool that can organize data streams using their accompanying tags and ease user navigation through large volumes of monitoring data.

I. INTRODUCTION

ENTERPRISE systems consisting of servers, storage, databases, networking equipment, applications etc. are hosted at one or more data centers. Monitoring of such data centers is done either on-site or from a remote command center. Monitoring various parameters of the entire IT system is the key to efficient management of a data center environment. The monitoring system collects raw data and processes it to generate basic events or aggregated data that is consumed by different management applications. Though each IT component is often packaged with open monitoring interfaces and tools, e.g., a database server or a network element will have published APIs for querying its performance metrics and/or some associated console for viewing and operating on the data, a monitoring system is required to integrate and process the data collected from heterogeneous sources, in the absence of which, administrators need to watch individual consoles, integrate the data manually and process it to generate alerts or perform some analysis.

A monitoring objective is a specification that defines how to collect a data stream, process it, and generate a type of event or an aggregated data stream. Monitoring objectives are also called Service Level Objectives or SLOs in this paper. A Service Level Agreement (SLA) is viewed as a possible composition of multiple SLOs. However, all SLOs need not participate in SLA evaluation as SLOs may be created purely for internal monitoring. Given a set of well-defined monitoring objectives, a monitoring system is typically configured or programmed once at setup before handing it over to the operations team. Minor configuration changes, such as thresholds, alert recipients etc. may be done by the operations team. However in dynamic environments resulting from corporate acquisitions, mergers, strategic outsourcing of IT service management etc., monitoring objectives cannot be frozen at setup but need to be specified on a continuous basis as new applications or hardware are deployed, or updates and transformations are made. It has been our experience that in many service-provider environments, operations personnel lack sufficient knowledge and skills to use or extend complex data modeling standards, such as Common Information Modeling (CIM) [10], and as such, find it difficult to properly categorize monitoring objectives using an intricate web of classes and

associations. For example, an operations personnel wanting to measure the utilization of a new disk partition created during operations, can write the script to measure the utilization. But, typically she does not know how to integrate the data in the monitoring system or configure some existing alert generation logic, because that requires a deeper understanding of the existing data models and standards.

For this reason, it is necessary to design an alternate data model representation that substantially eases the definition and management of SLOs in dynamic data center environments. In this paper, we report on the design and implementation of a SLO authoring and monitoring system called *Tango*, which has been built around such a data model representation. Tango is actually deployed in two large data centers and used by the operations staff as their primary monitoring tool.

We first describe how we use standard system configuration data and monitoring requirements specifications to design the structured part of our data model. The structured part has well-understood semantics that allows various types of processing to be automatically performed on the monitoring data. In addition, unstructured text tags outside the predefined model may also be attached to the data statically or at runtime. This allows any additional semantics not easily evident in the structured model to be associated with the data. The overall representation of data semantic is a simple concatenation of a set of structured and unstructured string tags, allowing easy authoring of SLOs.

Let us take the example where an operations team member wants to measure the utilization of a disk. In this case, the structured data types taken from higher-level CIM classes may be SystemResource, ComputerSystem etc. Each of these types, also called meta-data, provides meaning and context to the measured data, e.g., they identify the system component and the machine that the data is being collected from. At some point the operations personnel decides to monitor a specific disk partition's utilization and also wants to also record the partition names along with the utilization values. If the partition type is not present in the pre-created structured data model, then it can be supplied as an unstructured free text. The partition name, if known, could be supplied at authoring time or could be generated at run-time using the monitoring script or program.

The goal is to balance the benefits (e.g. uniform interpretation) of well-defined taxonomies like CIM with the ease of use that free text descriptions offer. Our approach has been motivated by the observation that a dynamic data center environment where infrastructure keeps evolving and the operations team has limited understanding of complex taxonomies, displays many of the general characteristics (e.g. unstable entities, amateur users) of domains where pure ontological classifications do not work well [16].

While the focus of Tango is on simplifying SLO authoring and basic data collection/aggregation, higher-level modules may be easily plugged into the system to consume the data and perform various analytics. One such higher level module that is described in this paper is the visualization component of the Tango system. The "dashboard" provides an integrated console to view all measured data and events. The data types are used to customize the views and organize the data in a hierarchical manner, e.g., the user can group the data by server, within each server by metric, and within each metric by partition name and so on. Note that while Tango is able to understand and interpret structured tags, it relies on the higher layer analysis modules, which are often written by experts, to apply text analysis techniques in the context of structured tags to interpret unstructured tag semantics. Accordingly, we are implementing text analysis algorithms in the dashboard to interpret the unstructured tags.

The rest of the paper is organized as follows. Section 2 details the data tagging scheme and the tag representation. Related products and research work are discussed in Section 3. Section 4 describes the overall system architecture built around data tagging along with some experimental results on system scalability. Section 5 shows how a tag-driven visualization tool simplifies data navigation in large repositories. Section 6 concludes the paper with a summary and discussion of future work.

II. DATA MODEL AND REPRESENTATION

In this section we describe how a data model is created, represented, and used in the Tango system. Our choice of data model and representation were guided by practical considerations such as the dynamic nature of data center environments where Tango was to be deployed and ease-of-use by the operations systems personnel.

A. Complexity in dynamic environments

We first explored the possibility of implementing our monitoring solution on top of a CIM data model. CIM (Common Information Model), developed by the DMTF Working Group [10], provides a common definition for a wide range of managed elements (including systems, networks, applications and services) enabling exchange of semantically rich management information between systems in a vendor-neutral manner. CIM is based on a hierarchical, object-oriented format designed to track and depict complex interdependencies and associations between different managed elements. Several vendors are now enabling support for CIM in their management solutions.

In stable environments, a CIM repository containing a complete model of the infrastructure may be set up at the onset. A monitoring solution, designed to interface with this repository, may then be programmed to handle a set of well-defined monitoring objectives, before being handed over to the operations team. Subsequently, minor configuration changes may be done by the operations personnel, and as long as the underlying infrastructure remains stable, the complexity of authoring new objectives may be hidden from the operations team through appropriate interfaces to the CIM infrastructure repository.

However, in dynamic environments undergoing transformation, infrastructure evolves rapidly along with the addition and modification of monitoring objectives. Faced with the task of prototyping a systems management solution for such environments, we noted that operations personnel in general lacked sufficient knowledge and skills to use or extend complex data models like CIM. Consequently, as infrastructure evolved, there was a substantial barrier in keeping the CIM data model up-to-date and in authoring new SLOs based on this model. However, the evolving infrastructure still had to be monitored and problem situations detected and acted upon. What was needed then was a monitoring solution that was lightweight, intuitive to program and fast to deploy, and that could be readily used by the operations team till the environment stabilized and more heavy-duty management solutions could be introduced. These requirements motivated the trade-offs we made in moving from a rich object-oriented approach like CIM with detailed infrastructure classes and associations, to an alternate lightweight semantics using simpler classes and implicit associations in the form of string tags. We explain this approach below.

B. A simplified data model

In a specific engagement, the customer supplied the system configuration data and monitoring requirements as relational databases, spreadsheets and semi-structured text files. In the first step, we examined this infrastructure data (manually) to determine the corresponding CIM classes. We are building a tool that semi-automates this process by searching for typical class instances in the input data, and can lead to significant reduction in the human effort required in determining the CIM class instances. Description of the tool, however, is beyond the scope of this paper. While CIM classes represent domain-independent semantics, part of the data model may be created for a specific domain. For example, in a Telecom engagement, concepts common to the Telco domain in terms of specific applications, metrics or infrastructure setups are modeled as the domain-specific part of the monitoring data model.

Once the data model is defined, SLOs need to be configured to satisfy the monitoring requirements and the semantics of collected data or generated events needs to be associated by a human. For example, a system administrator has to configure a data sensor to collect some specific kind of monitoring data stream and describe what the data means. The CIM classes identified in the previous step may be used for this purpose, but with some simplifications. First, many of these classes contain an extensive set of attributes that are useful for capturing detailed configuration information about the corresponding system entities, but are often redundant from the perspective of monitoring these entities. For example, a very typical monitoring requirement in a data center is to periodically check the availability of a server. In this case, the server name (say, "server1") and the metric name ("availability") constitute necessary and sufficient descriptors of the monitoring result (available/unavailable) generated by a probe trying to ping the server. There are CIM classes corresponding to metric and server (CIM_BaseMetricDefinition and CIM_ComputerSystem), which have name attributes. However, there are also several other attributes that are redundant from the perspective of monitoring the server e.g. Dedicated, ResetCapability etc. of CIM_ComputerSystem, which we leave out. Second, a managed element may be distinguished not only by its own attributes, but also by its associations. For example, to refer to a process proc1, we may not only want to use its name, but also the server it is running on. To reduce the burden of defining and maintaining these associations, we allow users to simply include as many attributes across different classes as they feel is necessary to fully describe the monitoring context. These attributes are then concatenated to generate a tag that uniquely identifies the SLO.

As an example, let us assume that the user wants to monitor the disk utilization on a server to throw alerts when the utilization exceeds a certain threshold. Say, the tag values specified by the user for the collected data are utilization, disk, and server1, which are instances of types BaseMetricDefinition, SystemResource, and ComputerSsystem, respectively.

The individual tags associated with a monitoring specification are concatenated to provide a concise string representation of the data semantics. For example, the set of tags in the above example are concatenated to form the string:

'<BaseMetricDefinition.name=utilization/<SystemResource.name=disk>/<Computer System.name=server1>'.

The first element in the tag representation corresponds to the metric being measured, while the rest of the tag defines the context of this measurement.

Now suppose, another monitoring specification is entered possibly by a different

user to measure the throughput of an application. The tag values specified by the user are printBill, throughput, server1 corresponding to the types in the data model, Application, BaseMetricDefinition and ComputerSystem. By using fast string matching algorithms on the two semantic representations, a piece of analysis logic can relate the two monitoring data streams in real-time. Note, however, that since our approach has been tailored for dynamic environments where maintaining a formal model of infrastructure elements and associations is difficult, monitoring data streams have to be related purely on the basis of information contained in the tags, and the level of inference/analysis that can be achieved is thus commensurate with this information content.

Apart from simplicity and consequent ease of use, an additional advantage of our semantic representation technique is that architecturally, the approach naturally facilitates a publish-subscribe type of message bus implementation. The unique tag specified by the user for a SLO becomes the "topic" on which metric values for this SLO are published to interested agents (subscribers). We will return to this topic in Section IV.

C. Incorporating Unstructured Data Types

As mentioned in the Introduction, the specific tag instances in Tango could be picked from a list of pre-created tag choices as part of the defined structured data model or if they are not easily evident in the defined model, they could be entered as free text by the user. This affords flexibility in defining new monitoring requirements that require extensions to the pre-created model. These free-text tags may be specified statically at authoring time or can be generated by the data collection logic at runtime. In the above example on measuring disk utilization, the data collection logic can generate the pairs by collecting per partition disk utilization. Thus the utilization values can be tagged with partition names generated on the fly in addition to the three tags defined earlier. The representation of the data semantics in this case is

'<BaseMetricDefinition.name=utilization>/<SystemResource.name=disk>/<Comp
uterSystem.name=server1>/<part.name=hda>'

The tag type part and attribute name specified by the user is a free text description of the tag value 'hda' attached by the data collection logic at runtime. The burden rests with an application, which understands the concept of a disk partition, to process the tag description, value, and use an internal dictionary along with the structured data model tags, to associate 'part' with the concept of a disk partition.

In a sense, the incorporation of unstructured tag types and tag values shifts the complexity from authoring to analytics modules. The underlying assumption is that analytics application writers are domain experts who can use existing unstructured text mining algorithms and tools to infer types from the free text descriptions and/or the tag values. Also, the operations people, who create unstructured tag descriptions and tag values, have the level of sophistication to provide enough keywords in the free text to make it feasible for the analytics applications or data consumers to infer the required type information.

III. RELATED WORK

There are numerous commercially available management solutions for monitoring enterprise systems at infrastructure, application, and business levels. In the domain

of network management alone, there is a plethora of tools [1], offering a wide range of monitoring capabilities for network security, performance, connectivity and topology. Application monitoring tools [2] [3] [4] that track availability and application performance through simulated end-user transactions, have gained popularity. Business intelligence tools [5] [6] supporting data integration, analysis and reporting, are helping managers monitor the overall health of the business. There are also product families offering integrated monitoring capabilities across different levels like HP Open View, IBM Tivoli, and CA Unicenter [7][8][9]. To support interoperability between all such management solutions from different vendors (with their own built-in data models) and facilitate unified administration, the DMTF Working Group has developed the Common Information Model (CIM) [10] that is poised to emerge as an industry-wide accepted standard.

As pointed out in the preceding sections however, when the infrastructure is rapidly evolving, the skill profile of operations team members makes it difficult for them to program monitoring solutions based on formal data models with rich classes and associations. Instead, to simplify authoring of objectives, we found it useful to allow user-defined types - along with some widely understood pre-defined types - leaving interpretation of new types to analytics modules that use unstructured information management techniques. Our approach of combining pre-defined and user-defined types in this way has interesting parallels on the web. On the one hand, web ontologies [11] are being actively developed to support uniform interpretation and integration of heterogeneous content. On the other, an alternative called "collaborative tagging" – allowing users to freely attach tags or keywords to shared content - is gaining popularity in settings where rigid taxonomies are found to be too restrictive. For example, this form of tagging (or "folk taxonomy" [12]) is now an integral part of many social media applications (e.g. flickr.com, del.icio.us, technorati.com etc.) that allow users to tag a variety of content including photographs, bookmarks and articles.

There is also a growing body of literature on the successful use of unstructured information analysis techniques in systems management. For example [17] uses text clustering on problem ticket text to determine a set of problem categories. The work in [18] applies text mining techniques to categorize messages in disparate log files into common event types like "start", "stop", "connection", "create" etc, based on the likelihood inferred from the training set of labeled documents. We anticipate that such approaches will be readily applicable to the processing of unstructured tags in our monitoring system and the discovery of semantics inherent in user-specified data.

IV. SYSTEM ARCHITECTURE

A. Overview

Tango is a distributed agent-based system written in Java. Fig 1 shows an overview of the Tango architecture. The box at the bottom represents the various sources from which data streams may be collected through standardized or proprietary *touchpoints*. An example data source may be a WebSphere Application Server, which supports the published Performance Monitoring API (PMI) as a touchpoint to collect performance data; again, an order management application may keep its performance data in an internal database exposed through a proprietary interface or touchpoint. This data is collected and analysed by Tango, whose three main components are shown on top of the data sources box in Fig. 1. These components are (i) *Sensors* running on *agents* that collect data from the data sources (ii) A

central *Tango server* that contains all processing logic for data aggregation and event generation and (iii) An *objective authoring tool* which lets users configure the sensors and the server's processing logic in a non-intrusive manner, as per the monitoring requirements. These components communicate through a JMS compliant *publish-subscribe cluster* as shown in the figure. We will now describe each of these components in detail.

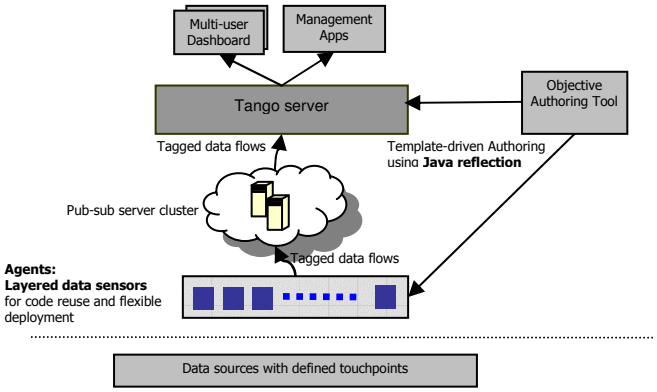

Figure 1: System Architecture Overview

B. Data Collection through Sensors

Sensors are configured in agents to collect data from known touchpoints, which may support local or remote interfaces. If a touchpoint on a machine can be accessed only locally then an agent needs to be installed on the machine to run the corresponding data collection sensor on the machine. Agents can be also upgraded with new sensor classes on the fly. Since an agent is written in Java, sensor classes run within a sandbox with minimum possibility of impact on the host machine.

Data sensors running on an agent may be layered, where the bottom-most sensor layer connects to the touchpoint and the higher sensor layers incrementally process the information in a form that can be published to the Tango server. The current Tango implementation supports sensors in two layers. The bottom layer is called a *provider* and the upper layer is called a *collector*. The collectors are often application specific but the number of provider classes is typically equal to the number of unique touchpoint types. Layering in sensors enables code reuse. Additionally, different layers of a sensor can be spread across agents to distribute the processing load; such layers can communicate using the publish-subscribe cluster.

An example agent architecture is shown in Fig. 2. Let us suppose that there is a CRM and a Billing application on a server. The touchpoint for the CRM application is a DB2 database whereas the touchpoint for the Billing application is an Oracle database. A single query may be used to retrieve the response time and throughput data from a touchpoint. In the example, a single JDBCProvider class implements interactions with various types of databases, including DB2 and Oracle. Thus two instances of this class are created for the two sensors for CRM and Billing touchpoints. The SQL queries are configured in the JDBCProvider to retrieve the response time and throughput timeseries samples periodically. Further, for each JDBCProvider instance, two collectors are instantiated to demultiplex the query result and extract response time and throughput values.

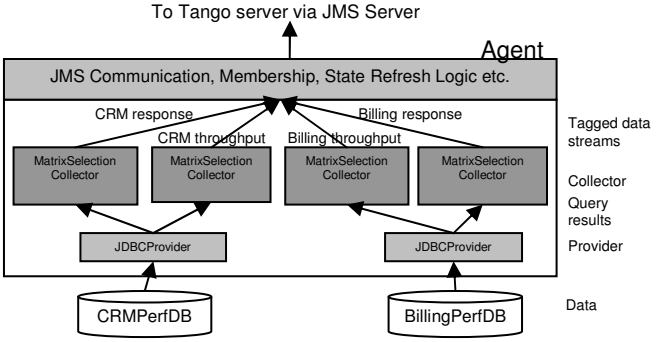

Figure 2: Agent Architecture

Collectors are responsible for tagging all data items they publish. The topic for publishing a data item into the publish-subscribe system is a default ordered concatenation of the set of tags assigned to a data item. Data consumers, such as processing logic in the Tango server, pick up corresponding tagged data items from the messaging bus.

The control plane of the agent-based system involves installation and update of sensor state in agents so that they can collect data which is routed through potentially distributed sensor layers to the appropriate processing logic in the Tango server. In addition, the agents can dynamically join or leave the system or can lose their state information. For example, when a production server is rebooted the agent will disappear along with all associated configuration state. Configuration state cannot be persisted on the production machine to keep the agent light-weight. The control plane agent management protocol implemented in Tango is similar to an acknowledgement-free announcement protocol built on the publish-subscribe system. The control message announcements are published periodically from the Tango server as soft state refreshes containing agent configuration state. An agent can pick up messages addressed to itself and check for new or updated state corresponding to itself on the server.

C. Data Processing by Server

Fig. 3 shows the Tango server internals. The server classes contain processing logic, such as for *data aggregation* and *event generatio*n, which can be applied on the tagged data streams produced by the sensors. The server classes can be instantiated on the fly and configured (using the objective authoring tool) to receive and process data streams of certain tags. Aggregated data streams with associated tags can be persisted in a local repository on the server or published on to the JMS server cluster to be picked up by an analysis application. Events generated as a result of the processing can also be persisted in the repository with accompanying tags. Additionally, events are always published on the JMS cluster to be picked up by visualization tool instances that show events in real time.

New server classes can be plugged in easily. For example, we implemented a simple event filtering module which had a few filtering rule templates based on tagged events generated by the basic event generation logic. This module was plugged in to intercept, buffer, and filter events before they reach the repository or JMS communication modules.

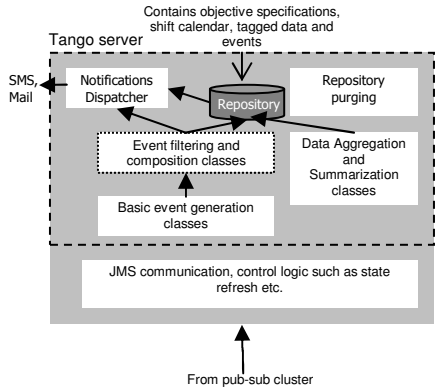

Figure 3: Tango server internals

Additionally, the Tango server also contains algorithms for *tag-driven data purging* of the repository as opposed to time-based purging. In the latter, all data beyond a certain threshold time is deleted. In Tango's purging algorithm, data is retained based on its speculated importance. Importance is derived from several considerations such as tags, proximity to events, age etc. Details of the purging algorithm are beyond the scope of this paper.

The *notification dispatcher* (ND) process that runs on the Tango server can be configured to receive events from the JMS server or from the repository. Event generation in Tango is tied to a series of stages that process the metric data before the relevant contacts are notified. Basically, these stages decide: i) Whether an event has occurred ii) Who needs to be informed about the event iii) How the concerned person/persons need to be notified and iv) What the content of the notification message would be. The first step involves an escalation matrix, which is described in detail in the following subsection. The event generation logic fills in the applicable escalation level, or the cell identifier of the escalation matrix, in the event. The cell identifier corresponds to a collection of users and groups. The time of the event and the information on work shifts are then factored in to identify the contacts available at the given time. Once the list of contacts is enumerated, their availability status is obtained from the repository along with their contact information and preferred mode of contact which could be email, SMS or a voice call. Depending on the availability of users, their contact preferences and the urgency of the situation, the final list for notification is created. ND then customizes the body of the message for each individual contact based on the considered factors. For instance, a notification through email contains all the available information including the corresponding logs for the event whereas an SMS only contains the event description along with the time and value of the event. It might be useful in some cases to also consider the presence information available from the cellular network to know the proximity of the contacts to the physical location of the event. This would ensure that those events that need physical proximity for resolution (e.g.: network cable unplugged) would be attended to at the earliest.. Similarly on an event closure, the event generation logic identifies the appropriate contacts for the event.All the information required by ND for the aforementioned steps with the exception of presence and user status are made available through the authoring tool.

A key novelty in Tango architecture is the use of class reflection to loosely couple the components and ensure plug-and-play of new components in Tango server or agent to process or sense data. Class reflection technology is widely used within the system to enable template-drive configuration by the authoring tool. Every configurable class, such as providers, collectors, event generation classes etc, need

to support the reflection interface so that the objective authoring tool can present templates to the user. In the following subsection, we describe the process of objective authoring.

D. Objectives Authoring

We now describe how the authoring tool lets the users easily configure both the Tango server as well as the sensors running on the agent machines through template-driven authoring.

The configuration of an objective or SLO is done by the Objective Authoring Tool. Tango provides some support for objectives composition; however more sophisticated composition can be implemented as an application using the tagged data in the Tango repository, or runtime feed from the JMS server cluster. A sample objective that monitors the queue size of an application is shown in Figure 4. All templates presented in screenshots are created by querying the appropriate classes from the authoring tool using Java reflection.

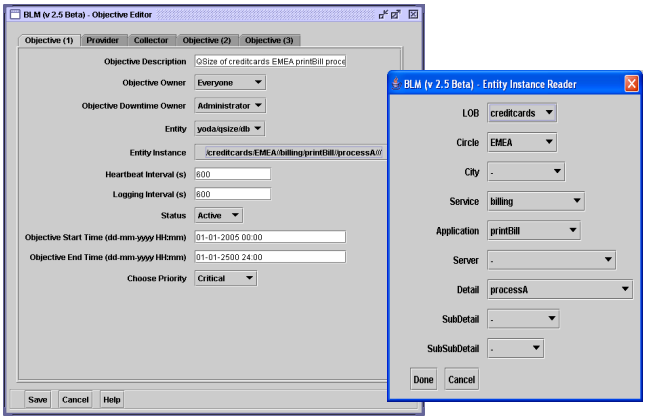

Figure 4: Objective authoring pane

The screenshot in Fig. 4 shows the first pane of an objective being authored. The template of the first pane shows a variety of administrative information like access control parameters, period of operation, frequency of data logging, priority of events generated, heartbeat intervals to detect non-working objectives etc. Most importantly, it shows the tags that are taken from structured or unstructured data models in the Tango server. If an ontology is specified for a Tango deployment, then structured tags are picked from the ontology. For example, in this case, application, LOB, city, service, server, metric, are shown as part of the defined ontology. However, the tag processA is an unstructured piece of text. In fact, additional tags can also be attached to the data stream by the agent at run-time. For example, processA could be a run-time tag attached to the data stream by an appropriate sensor logic without being configured statically in the authoring tool. Section 2 discusses tagging and tag representation in more detail.

The authoring tool can process the tags from the structured data model and look up the appropriate sensor classes, i.e. provider and collector classes, to collect the data and instantiate the appropriate Tango server processing logic classes to handle the collected data stream. Alternatively the user can manually pick the

classes from the existing list. New sensors or processing logic classes need to be written only in case the user is not able to create a specification with the existing classes.

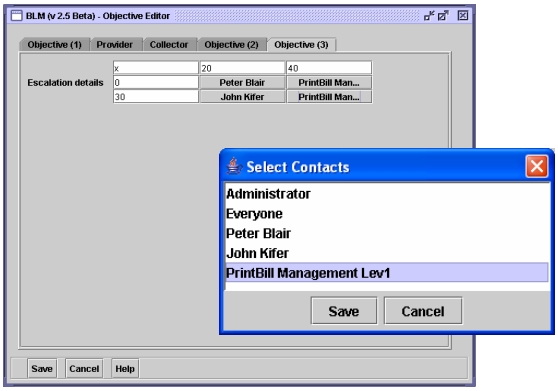

Figure 5: Escalation matrix

Configuring Tango Server: The authoring tool may be used to easily configure the event generation and notification capabilities of the Tango server component. Continuing the above example, since queue size is a numeric value, multiple thresholds may need to be applied on the value to generate alerts. Based on this, the authoring tool presents an escalation matrix to the user, Fig. 5. The escalation matrix is essentially a state diagram that represents all the 'bad' states that an objective can be in. A cell in the matrix corresponds to a state and contains the logical notification group for that bad state. In this case, a bad state is characterized along two dimensions, severity of queue size (along the columns) and elapse of time (along the rows). For example, in the 2x2 matrix, the cell *[col=20,row=30]* implements the rule *"If qsize is >20 for 30 minutes, notify John Kifer"*. Note that if, instead of queue size, the objective was to monitor availability of a process, then a one-dimensional (time-based) escalation matrix would have been generated.

An objective enters into a problem phase, identified by a unique problem identifier associated with the event, when it first enters the escalation matrix, i.e. one of the bad states. Alerts are generated at matrix entry and exit and, also whenever transitions are made from one cell to another. Logical contact groups keep getting added, or deleted, to the set of logical notification groups for an event as transitions are made to higher escalation, or lower escalation, levels respectively. Finally, when the objective makes a transition out of the escalation matrix, i.e. into 'good' state, the problem identifier is closed and the existing set of logical notification groups is contacted and contacts in this group are alerted of the closing of event.

Configuring sensors: The authoring tool greatly eases the task of configuring sensors for an objective. The tool presents the existing sensor classes to the user, and once a selection has been made, it can query the corresponding provider and collector classes using reflection, and present templates to the user to configure the sensor. For the queue size monitoring example, the templates for the provider and collector are shown in Fig. 6. The provider queries the database for the queue size and possibly other parameters to amortize the query cost. The collector shown in the figure selects the queue size value for EMEA from the query result.

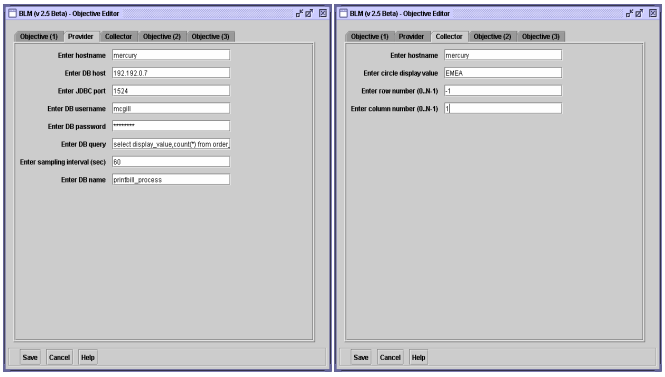

Figure 6: Sensor configuration

Once an objective has been configured using the authoring tool as described above, the system will automatically pick up the objective, instantiate a sensor for it, configure the event generation and notification capabilities of the server for this objective, and set up topics on the publish-subscribe cluster using which the sensor and the server can communicate.

E. Overhead Experiments

The following overhead related experiments were performed in an actual data center deployment of the Tango system. The Tango Server is hosted on an IBM p 9113-550 with 16GB of RAM and 4 CPUs running at 1.6 GHz each. Close to 1500 SLOs are configured to monitor around 500 servers.

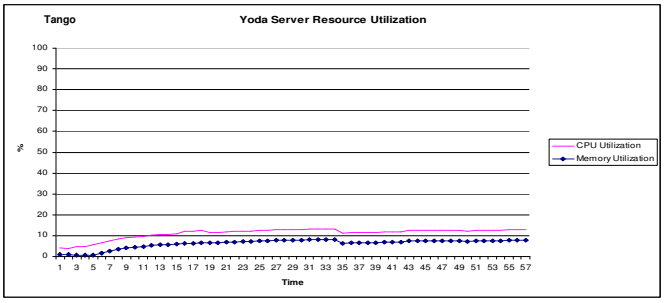

Figure 7: Tango server overheads

The average CPU utilization for the server component, consisting of 3 Java processes each handling around 500 SLOs, was found to be 6.14% and the memory utilization was 4.95%. Fig. 7 shows the timeseries plot of the CPU and memory utilization of the server showing stable low utilization for several hours.

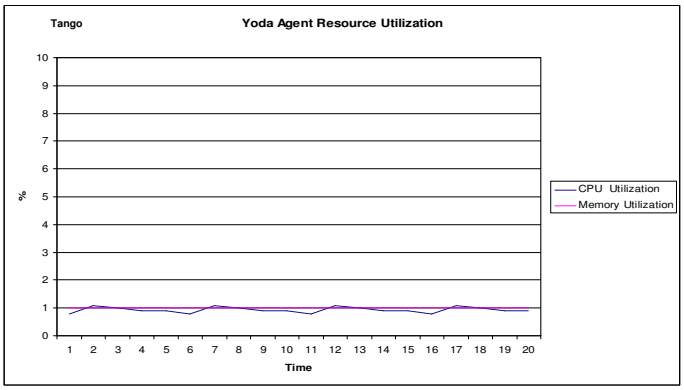

Figure 8: Tango agent overheads

A typical Tango agent's CPU and memory utilization timeseries plots for several hours are shown in Fig. 8. The agent measured was configured to handle 235 objectives. The average utilization of this agent was 0.9% of CPU and 1% of the available memory. The utilization measurements do not include the actual running of the code that performs the data sensing and processing activity at the agent but the overheads of agent itself to support the operation of the configured sensors. The system currently generates over 1600 automated event notifications besides collecting over 500,000 data samples per day.

V. TANGO DEPLOYMENT

We have successfully deployed Tango at a very large data center undergoing rapid transformation. A second deployment is in progress at the time of this writing. The system has been in operation at the first data center for over 12 months. The objectives have been configured by low skilled operations staff using the Tango authoring tool and a pre-created tag dictionary. Occasionally tag types and tag strings have been contributed in the dictionary when existing ones have been found insufficient.

Analysis of the SLO tag values in the first deployment reveals 18 types of metrics, 45 applications and 500+ servers. Fig. 9 shows the frequency distribution of SLOs across the most common metrics. As would be expected in monitoring engagements, availability of machines is the most common property that is monitored (around 21% of SLOs), while utilization and latency related metrics are also very common. The distribution of SLOs by applications is quite skewed, with only 6 of the 45 applications accounting for 52% of the SLOs. The SLO distribution across servers is, however, more even, with the top 7 servers accounting for only 9% of the SLOs. The tag types and values used in a Tango deployment thus provide a useful index into the overall span of the monitoring activity as well as the relative importance of individual infrastructure elements from the service level perspective. Tags also help to slice and dice the data along various dimensions to generate informative views as shown in Section VI on tag-driven visualization. However, organized views may not suffice given the large volume of event notifications. One needs sufficient labor cost to handle even 1600 events per day. In Section VII we also describe ongoing analysis on tagged events to reduce the large volume of events to a smaller number of problem situations.

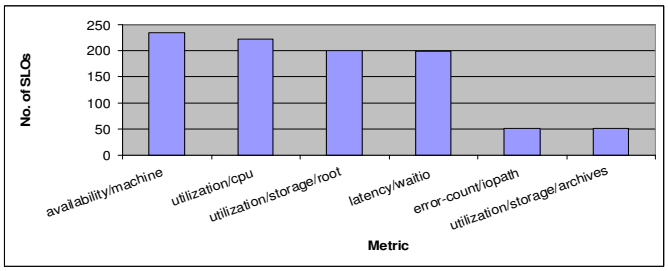

Figure 9: SLO distribution across metrics

VI. TAG DRIVEN VISUALIZATION

In this section we describe the Tango dashboard and show how it utilizes the tags described previously to systematically organize and present the metric data. The tag corresponding to a data item attaches meaning to the data; different elements of the tag provide different contexts in which the data was measured. Users of the dashboard can use the contexts to group data and customize their views.

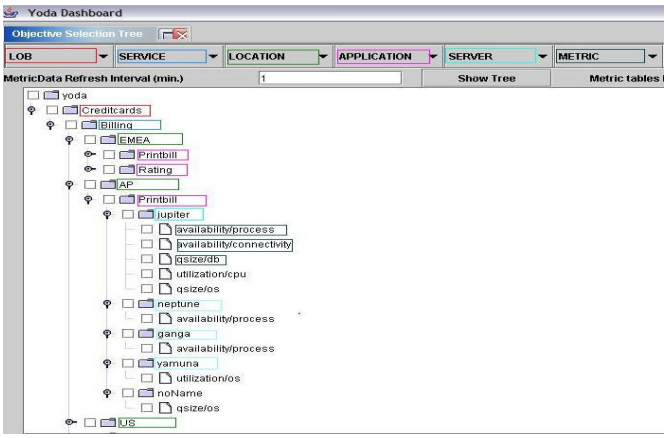

Figure 10: Dashboard tag hierarchy selection

Fig. 10 shows the default view of the dashboard application. The default view allows the user to select meta-data that the data should be grouped by. Selection of multiple meta-data leads to nested grouping in the specified order. In the example shown in Fig. 10 the user has selected the meta-data, LOB, service, location, application, server, and metric, in that order of nested grouping. The actual tag instances are CreditCards for LOB, Billing for service, and EMEA, AP, US for location. The applications being monitored are PrintBill and Rating, while jupiter, neptune, ganga, etc. are servers. Finally qsize/db, availability/process, utilization/cpu are instances of metric. In the example, the metric data is grouped by server, then applications, locations, services and finally line of business. Once the meta-data has been selected and ordered, the default view presents the corresponding tags in a hierarchy, or tree. This is called a context hierarchy. Different selections and ordering of the meta-data tags would result in different context hierarchies. Reorganization of the context hierarchy to get different views of the data can be done in real-time using the tags associated with the data. The dashboard can also present the data in a tabular manner by mapping the above tree to a nested

spreadsheet structure, which is a more comfortable option for many operations people.

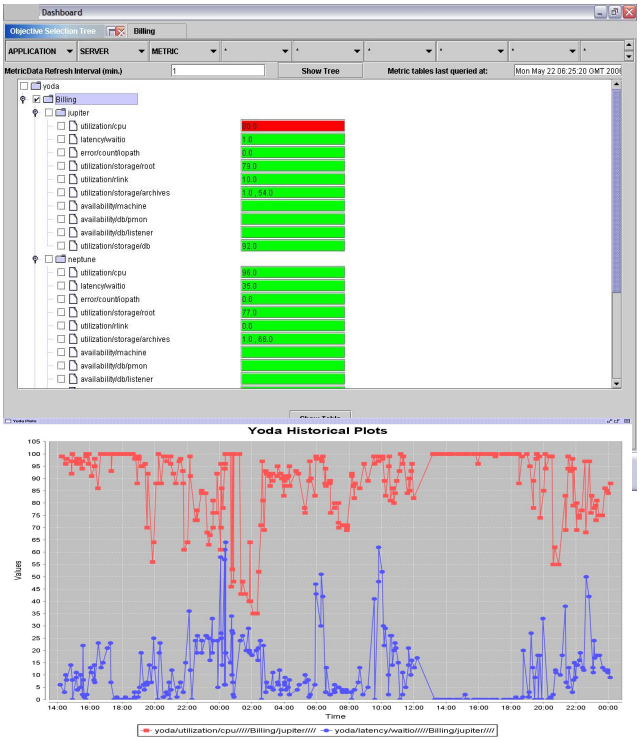

Figure 11: Dashboard views

In the dashboard, data is associated with each level of the hierarchy. While the leaf data is always the measured data, i.e., data from the monitoring system; the data at a higher "node" in the tree could be measured data or aggregated data. In the example, of Fig. 11, the data at the node jupiter could be an aggregation of the metric data corresponding to the measurements of availability/process, availability/connectivity, qsize/db, utilization/cpu and qsize/os. The user could supply aggregation functions, or there could be built-in functions that could aggregate by doing simple operations like union, intersection, addition etc. The dashboard can interpret the type and use inbuilt intelligence to decide the aggregation operations, e.g., the dashboard would do a union of the five data streams in the above example.

Let us take a few examples to explain the use case scenarios of the dashboard. In a data center environment different users have different roles and thus need different views on the data being collected, e.g., a business executive would be interested in the performance of her line of business, while a data center operator would be interested in the minute operational details of a server. In the dashboard, the business executive can select the LOB tag at the root of the context hierarchy and would get a summary of the performance of all applications and infrastructure in her line of business. In contrast, a data center operator managing servers would group data by the server tag and hence view all metrics associated with the servers, as shown in the top half of Fig. 11. A data center operator responsible for storage across the entire operation will group by a storage metric and hence in one screen, view the storage across all servers and be alerted to any events in the storage. As stated earlier, given the type of the data the dashboard can chose from among a set of inbuilt aggregation functions. For example., for the metric utilization/storage, the

dashboard would aggregate across all servers by finding the average storage usage utilization over all servers.

Though the above discussion has focused on selecting structured meta-data to customize views, the idea can be extended to the unstructured tags also. We are currently developing an algorithm to interpret the unstructured information and infer meta-data. These inferred meta-data can be used then to group data much like as described previously. The important feature of our unstructured data is that it is accompanied by structured data that not only constraints the "similarity matching" that can be done on it, but also provides an "index" into a system management ontology which can be then be used for interpreting the unstructured data. This can be done using modifications of standard unstructured information management algorithms [13].

VII. CONCLUSION AND FUTURE WORK

In this paper we described a monitoring system that uses a hybrid data model consisting of structured and unstructured parts to describe the semantics of monitoring data and events. The representation of data semantics is in terms of string tags. The benefit of using the tagging architecture is that the authoring of specifications becomes simpler and more intuitive for operations people who are less skilled in complex object-oriented data models. We also described a sample visualization application that uses the string tags to organize real-time views of the data.

We plan to build tag-based reusable monitoring ontologies for various domains through actual engagements. This will further ease the programming of a monitoring solution and thus reduce the deployment time in a new environment.

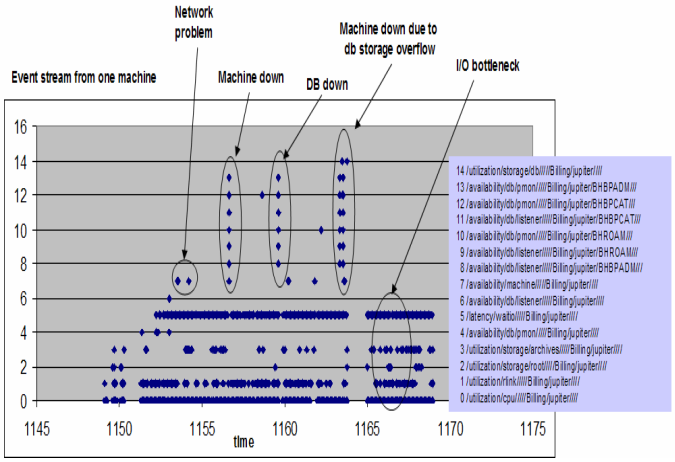

Figure 12: Extracting problem situations

We are also exploring both online and offline analysis of tagged events for better event management and notification. On the one hand, pre-defined rules defined on tags may be used to filter and/or correlate events in real-time, based on known relationship between tag values. On the other hand, offline analysis of tagged events may also reveal interesting problem situation patterns. Such situations may be represented by clusters of tags whose events occur within a specified time window. With a sufficiently rich historical event database, support/confidence metrics may also be associated with these situations, and presented to system administrators to verify, edit and label. Fig. 12 shows such a clustering of events and possible problem

situations that we have discovered by analyzing the event database of an actual deployment. These known patterns can then be used to generate richer notifications by clustering events based on problem situations.

REFERENCES

[1] Network Monitoring Tools. http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html#app

[2] Website Monitoring. http://www.webmetrics.com/

[3] Performance and Availability Monitoring http://www.mercury.com/us/products/business-availability-center/end-user-management/

[4] Tools for Database, Application and Windows Management. http://www.quest.com/foglight/

[5] Business Intelligence from Business Objects. http://www.businessobjects.com/

[6] Business Intelligence and Performance Management Software Solutions from Cognos. http://www.cognos.com/

[7] http://h20229.www2.hp.com/

[8] HP OpenView. http://www-306.ibm.com/software/tivoli/

[9] Enterprise Systems Management Solutions from CA. http://www3.ca.com/solutions/Solution.aspx?ID=315

[10] DMTF – Common Information Model (CIM). http://www.dmtf.org/standards/cim/

[11] OWL Web Ontology Language. W3 Recommendation, 2004. http://www.w3.org/TR/owl-features/

[12] Folksonomies – Cooperative Classification and Communication Through Shared Metadata. A. Mathes, University of Illinois Urbana-Champaign, 2004. http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html

[13] Unstructured Information Management Architecture. http://www.research.ibm.com/UIMA/

[14] Elements of Information Theory. T.M. Cover and J. Thomas. Wiley Series in Telecommunications, 1991

[15] The Minimum Description Length Principle in Coding and Modeling. A. Barron, J. Rissanen and B. Yu. IEEE Transactions. on Information Theory. Oct 1998, pp 2743-2760

[16] Ontology is Overrated: Categories, Links and Tags. C. Shirky, 2005. http://www.shirky.com/writings/ontology_overrated.html

[17] Knowledge Base Maintenance Using Knowledge Gap Analysis. S. Spangler and J. Kreulen. Proceedings of 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 462-466, 2001

[18] Mining Log Files for Data-Driven System Management. W. Peng, T. Li and S. Ma. ACM SIGKDD Explorations Newsletter, Volume 7, Issue 1, pp 44-51, June, 2005

[19] IBM Workplace Dashboard Framework, http://www-142.ibm.com/software/workplace/dashboards

[20] Apple Dashboard www.**apple**.com/macosx/features/**dashboard**/

[21] Betts, Mitch (April 14, 2003). "Management Dashboards Becoming Mainstream:" ComputerWorld.