A Business Centric End-to-End Monitoring Approach for Service Composites

Geetika T. Lakshmanan, Paul Keyser, Aleksander Slominski, Francisco Curbera, Rania Khalaf
IBM T. J. Watson Research Center
Hawthorne, NY, USA
{gtlakshm,pkeyser,aslom,curbera,rkhalaf}@us.ibm.com

Abstract— Enterprise applications today are composed of multiple independently executing services and processes that collectively provide a solution to a business problem. These composite applications contain a heterogeneous collection of services that execute in a variety of runtimes making them difficult to manage while maintaining a business centric point of view, as opposed to a service point of view. This paper introduces a business centric monitoring framework to bridge the gap between the business and service levels in complex business applications. Our technical approach focuses on using business information invariants to define one or more monitor sets in order to relate service activity to business composite execution. We apply this framework to enable end-to-end monitoring of composite business applications. In this paper we present an initial prototype of our business centric monitoring approach using monitor sets for monitoring a simple loan application composite implemented on IBM’s WebSphere Business Modeler, Process Server and Business Monitor. Our prototype implementation demonstrates the convenience, effectiveness and ease of design and deployment of our monitoring solution to attain a single end-to-end business centric view of a collection of heterogeneous services executing together. Our work also exposes potential challenges as we extend this work to support more powerful end-to-end monitoring.

Keywords-monitoring; service composites; business; end-to-end; monitor set

I. INTRODUCTION

Line of business (LOB) applications in large organizations today are composed of multiple functional components and are implemented by many different applications and services. The execution of these service components is implicitly or explicitly orchestrated to achieve the business goals of the particular LOB processes. Heterogeneous technologies, natural componentization and the rich complexity of interactions make LOB applications ideal targets for the service oriented methodology. Service encapsulation, interoperability standards and loose coupling between components are natural and strong requirements. The scale of the applications and their end-to-end nature does not encourage a single runtime implementation strategy such as a single business process executed under the control of a single process engine. In addition, application users wish to manage the business function structure of the LOB application, which is not always easily aligned with the service deployment structure. This is particularly true of legacy applications that are not created through a top down service oriented modeling discipline.

In such scenarios, where there is no single controlling middleware platform and where the alignment between the business and service layers is indirect, it is not possible to rely on middleware mechanisms alone for delivering unified, end-to-end monitoring of the LOB application as a whole. While the application is a consistent unit from a business perspective, there is a gap between the business definition and the management capabilities provided at the system level. This paper is concerned with the problem of bridging this gap, while maintaining a business centric point of view. In other words, we propose to enable monitoring of LOB applications as coherent business entities, and not as a set of system components. It should be noted that in this paper we focus on business centric monitoring which is different from system level monitoring. There are several existing system level monitoring approaches that aim to optimize performance and resource utilization, however this is not the subject of this paper.

We aim to achieve a level of runtime end-to-end monitoring over composite LOB applications that is consistent with their business goals, in the absence of integrated system control. We assume that the execution of composite LOB applications is supported by a set of invariant information fields that are shared across components and which enable distributed coordination of business logic and information. Our methodology involves allowing the user to identify the components they wish to monitor from a business centric point of view. We do not make assumptions about the components being tied to a particular workflow engine or BPM product. In this paper we present an initial prototype of our business centric monitoring approach using monitor sets for monitoring a simple loan application service composite on IBM’s WebSphere Business Process Management (BPM) suite of products that include Business Modeler and Business Monitor. Our prototype implementation demonstrates the convenience, effectiveness and ease of design and deployment of our monitoring framework. Our work also exposes potential challenges as we extend this work to support more powerful end-to-end monitoring.

II. RELATED WORK

Monitoring service compositions is the subject of considerable research. Existing approaches differ in their individual goals and this influences their monitoring methodology. Baresi et al. describe an approach to monitor WS-BPEL processes focusing on runtime validation [1]. They wish to monitor phenomena such as pre-conditions and post-conditions associated with the invocations of web
services, invariants that can be attached to WS-BPEL scopes, and punctual assertions indicating a property that must hold at a precise point of execution. They require users to specify rules, where each rule consists of a monitoring expression that needs to be evaluated, parameters that serve as meta-level information that defines the context of the monitoring rule, and the exact location in the WS-BPEL process where the rule should be evaluated. Barbon et al. describe a monitoring approach for distributed business processes implemented in BPEL which supports run-time checking of assumptions and conditions that the process and its underlying services are expected to satisfy [2]. They provide an expressive language for the specification of instance monitors and class monitors that allow them to support monitoring. Spanoudakis et al. propose a framework for runtime monitoring of the compliance of systems composed of Web Services [3,4]. Their approach involves using event calculus to specify the requirements to be monitored. Harada et al. build an engine to detect contextual correlation of process events that can be used for detecting money laundering [5]. Wetzstein et al. target their monitoring approach towards determining the factors that influence the performance of business processes and most frequently contribute to the violation of key performance indicator (KPI) target values [6]. They provide a framework for performance monitoring and analysis of WS-BPEL processes which consolidates process events and Quality of Service (QoS) measurements. They use machine learning techniques in order to construct tree structures which represent the dependencies of KPIs on process and QoS metrics. Robinson et al. describe an approach to monitor interactions among heterogeneous services by tracking their commitments [7]. They provide an expressive requirements-monitoring language that employs the notation of UML and its Object Constraint Language (OCL). Their work builds on the research by Desai et al. [8], who argue for process specifications with commitment semantics. Our goals in providing monitoring for service composites are fundamentally different from these aforementioned approaches [1,2,3,4,5,6,7,8]. We wish to provide end-to-end business-centric monitoring whereas many of the existing approaches have very specific objectives that they wish to accomplish via monitoring such as verification of compliance objectives or scrutiny of assumptions. Furthermore these existing approaches are applicable to very specific kinds of processes (for instance WS-BPEL specific), and require users to have detailed technical knowledge of the service composite and its low-level implementation, in order to formulate expressions for intricate monitoring. Our monitoring methodology is distinct from these aforementioned approaches due to following concrete reasons:

- User is not required to have a detailed technical understanding of how the components in the composite are implemented. The user is only required to see an end-to-end visual representation of the business composite.
- We support monitoring of different kinds of components. In addition to WS-BPEL business processes, a service component can be any SCA component (for instance a piece of Java code that calls a web service) provided that there exists a XML schema definition (XSD) of the messages passed in and out of the component. In section IV we explain why this is necessary.
- Applicability to different process monitoring platforms (not tied to a specific process engine).

Our monitoring strategy correlates individual instances of each business component to piece together end-to-end process instances. Our work is motivated by conversations with business architects of LOB applications who have expressed a need for intuitive and simple mechanisms to automatically enable end-to-end monitoring of a set of heterogeneous composite services. There is a need for monitoring tools that can be driven by users but do not require users to have deep technical understanding of the low level runtime infrastructure.

A number of vendors in the business process management space currently offer business activity monitoring (BAM) software. IBM’s business process management suite of products integrates monitoring with business process lifecycle management. Although existing IBM products support BAM [9], the intuitive end-to-end monitoring approach for business composites described in this paper is not currently available and requires manual configuration. There are a number of other vendors besides IBM. TIBCO for instance offers BusinessFactor which analyzes business operations in the context of business objectives, past performance, and current conditions so that customers can identify and address risks and opportunities. Oracle Business Activity Monitoring (Oracle BAM) is a complete solution for building interactive, real-time dashboards and proactive alerts for monitoring business processes and services. Savvion lets customers create metrics to measure what is important to them and monitor them from a dashboard. Savvion’s dashboards outline problems, probable causes, and recommend corrective actions to avert further problems. HP has created a platform for business operation management that uses data mining to support process monitoring as well as analysis and prediction [10]. Many of these commercial monitoring solutions are BAM centric while others use monitoring as an intermediate step to answer intricate questions such as the prediction of parameters [10]. Our proposed solution to monitoring can be seen as located between BAM and BPM. We exclusively focus on providing end-to-end monitoring of heterogeneous service composites. The real-time access to key performance indicators (KPIs) and metrics via dashboards provided by BAM can easily complement our end-to-end monitoring approach, and furthermore can be integrated to provide end-to-end composite level performance indicators and metrics. In this regard we extend BAM in the direction of BPM with the exception that we do not require processes to be managed by BPM systems. We allow monitored activities and events to come from diverse sources (including informal processes). We can achieve effective monitoring even when we do not have data for every task in every process. Our goal is to
collect enough monitored data to piece together an end-to-end monitoring composite view. If processes are running in BPM systems we access them and combine their data with data collected from other sources such as email events, document updates, etc. Thus we can fortify BAM tools with business provenance to provide BPM-like capabilities without requiring a full-strength BPM system.

III. MONITORING LINE OF BUSINESS APPLICATIONS

We assume a two level business function decomposition, in which a business application is a top level composite with one or more business components which are then mapped into deployed application components, as shown in Fig. 1. We use the term service component to represent the deployed application components that realize business components. A business component is mapped to either a single service component or a collection of service components that collectively realize the business function assigned to the component. Fig. 1 shows how components at the service level are wired together. Each service component is accompanied with an XML schema definition of the messages it ingests or produces. A mapping specification assigns one or more deployed service components to each business component.

Figure 1. A Business composite and its mapping to service components.

We refer to the business documents and data ingested or transformed by business operations as business items. In other words, each business item may be part of the input or output of one or more business components in the composite. We refer to the data fields that further define business items as business item attributes. Such data fields could be strings, integers or more complex data objects.

The system architecture of our monitoring strategy is summarized in Fig. 2. As shown in the figure, in order to monitor a composite a user needs to provide an end-to-end view of the business composite (this could be a simple Visio diagram) and a monitor set that indicates the user’s monitoring needs. A monitor set consists of a set of invariant information fields that are shared across business components. We make this definition more precise in section III(A). The first step in Fig. 2 is “Monitor Model Generation.” Each business component results in a low level monitoring model. A monitoring system uses this to monitor the execution of corresponding service components. Events generated by service components are received by low level monitoring models. Low level models extract the values of “invariant” properties as defined by a monitor set (also referred to as business payload) from messages received from low level service components, and initialize “metric” fields with their values. “Metric” fields are defined in a message which is referred to as an outbound event. The monitor set specification dictates the design of this event definition.

Outbound events are received by a high level monitoring model as inbound events and their payload is used by high level monitoring models for correlating individual instances of business components that belong to the same business composite instance. High level monitoring models and low level monitoring models thus share the same event definitions. The monitor set specification allows high level monitoring models to identify and piece together corresponding instances of the same end-to-end composite based on the events they receive from the lower level monitoring models. The user provided monitor sets determine the number of low and high level monitoring models and their relationship in the monitoring hierarchy. Notice that our solution does not require information about the implementation, message order, endpoint wiring, or other such specifics of the constituent services. The emphasis of this paper is on the first step in Fig. 2 which is the generation and content of monitoring models which conduct end-to-end monitoring. Once generated, these monitoring models can be deployed on any monitoring system to track and correlate end-to-end composite instances (the remaining steps in Fig. 2). We describe the details of our prototype implementation in section IV.

We have implemented our monitoring solution for a loan application business composite. We begin with an end-to-end picture of this scenario (Fig. 3(a)) that consists of three business components: (1) Loan_Initiation: a user applies for a loan, (2) Review_Loan_and_Update: loan is reviewed and updated and (3) Loan_Completion: loan is approved or rejected. Review_Loan_and_Update is a manual human centric process, while Loan_Initiation and Loan_Completion are BPEL processes. Loan_Initiation (Fig. 3(b)) and Loan_Completion (Fig. 3(c)) map to one or more service components. In Loan_Initiation, the LoanApplicationType business item is provided as input to the Receive_Loan_Application service component and is
A business composite instance contains all the instances of a composite. The values of these attributes are constant during each individual execution of the composite. A business composite instance contains all the instances of its business components that are directly associated with the same monitor set attributes. In order to correlate end-to-end composite instances successfully, we assume that monitor set attributes are unique for each business instance.

An example of a monitor set, MonitorSet_MS, is shown in Fig. 4. LoanID is an attribute of this monitor set that maps to the LoanId attribute of the LoanApplicationType business item and to the LoanID attribute of the LoanType business item. The LoanApplicationType business item is referenced by the Loan_Initialization process, and the LoanType business item is referenced by the Loan_Completion process.

The XML representation for a monitor set is inspired by WS-BPEL correlation syntax. For brevity we show the XML syntax in Fig. 5 and show the informal grammar used in several XML standards including WS-BPEL. This notation uses an XML element instance, but the values indicate the data types instead of values. Cardinality of an element or attribute is denoted by appending ’*’ for zero or more and ‘+’ one or more. The complete XML Schema for a monitoringProject is provided in [11].

```
<monitoringProject
    xmlns="https://w3.ibm.com/ProcessCorrelationAPI"
    targetNamespace="http://my.example/my-monitoring"
    xmlns:m="my-monitoring"
    xmlns:p="Loan_Initiation"
    xmlns:q="Loan_Completion"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <property
    name="m:LoanID" type="xsd:string" />
  <propertyAlias
    propertyName="m:LoanID"
    query="/q:LoanApplicationType/LoanId/text()" />
  <monitorSet
    name="MonitorSet_MS"
    properties="m:LoanID" />
</monitoringProject>
```

Figure 4. A monitor set called MonitorSet_MS

```
<monitoringProject
    targetNamespace='xsd:QName'>
  <property
    name="xsd:QName" type='xsd:QName' />
  <propertyAlias
    propertyName='xsd:QName'
    query='xsd:string' />
  <monitorSet
    name='xsd:NCName'
    properties='list of xsd:QName'/>
</monitoringProject>
```

Figure 5. XML syntax for a monitor set.

The monitor set definition can be extended, in particular by specifying ways to “bridge” two or more monitor sets, in order to achieve correlations more powerful than the ones explored in this paper. This is the subject of our future research.

B. Matching Messages to Composite Instances

When a message is handled by a component and that component’s operational mapping specification determines that it is relevant, then its payload is checked for any property values for which a propertyAlias has been defined on this message type. If any such values are found, then the message is said to contain (one or more) properties. A message is said to contain a monitor set if all the property values related to one monitor set are found within the message payload.

Messages with the same monitor set values are associated with the same composite instance. Once the value of a property is set for a composite instance, it cannot be modified. Monitor set values are unique keys within the set of instances of a composite.

The box A1 in Fig. 6 illustrates the operation of instance matching for composites. We use the loan application scenario with two instances. Each instance has its monitor...
set values shown to its right. In the business composite tree the right child component is Loan_Initiation business component and the left is Loan_Completion business component. Let's consider how a message is matched. If a message of a component does not contain a monitor set, it is ignored. Otherwise, the monitor set in the message is compared against all monitor set values of the composite’s instances. If all the property values match that monitor set’s values in an instance, then the message is associated with the corresponding component of the matched composite instance.

Consider a new loan application message that is used by the Loan_Initiation component. The message contains LoanId 1255233 corresponding to property value for LoanID as 1255233. Therefore, the message has the MonitorSet_MS as defined above. Fig. 6(A2) illustrates how the message is matched. The message is checked against the monitor set values of the loan application instances and matches the top one. The message is therefore associated with the Loan_Initiation component of that composite instance (Fig. 6(A3)).

Next, another message is used by Loan_Initiation with LoanId 555555. It also contains the MonitorSet_MS, but its values for MonitorSet_MS do not match any instance of the loan application (Fig. 6(B1)). Thus, a new composite instance is identified with a new monitor set value of MonitorSet_MS containing these new values for LoanID (Fig. 6(B2)). The message is associated with the Loan_Initiation component of this new instance (Fig. 6(B3)).

Our goal in this paper is to achieve an initial prototype of our end-to-end monitoring solution on a real-world business process management platform using simple business correlations. Although simple, such a prototype can provide important insights as to whether we successfully meet our design goals for simplicity, ease of use, and portability in developing a monitoring solution for composite applications that do not necessarily share a common middleware infrastructure. We share insights into potential challenges in section V. Extending this implementation to support more powerful business correlations is the subject of our current research.

In this section we describe details of our implementation of an end-to-end monitoring solution on IBM’s WebSphere Business Process Management suite of products. In particular we begin with a diagram view of the loan application composite in WebSphere Business Modeler [12] and create a monitoring solution that runs on WebSphere Business Monitor [13], IBM’s business activity monitoring software that allows the creation of BAM dashboards and provides BPM-level monitoring for WebSphere Process Server (WPS). Monitor has simple-to-use tools to create monitoring models for a single business component but it does not support monitoring a composite of multiple business components. We leverage existing support for creating monitoring models for a single business component [14] and extend it to create a monitoring solution for a business composite. Our contribution includes a generic monitor set specification that can be used to monitor across any business composite by automatically generating a hierarchy of monitoring models.

A. System Architecture

The steps of our monitoring solution are as follows:

1) User creates one or more monitor sets. User then selects a monitor set, and identifies the business composite that they wish to create a monitoring solution for using the selected monitor set.

Our monitoring solution code does the following:

2) Automatically customize an XML event definition for the monitoring models to share based on the monitor set. Specifically create a field in the event definition that corresponds to each monitor set key.

3) For each monitor set key determine which business item attributes they reference, and for each of those business items, determine which business components they reference. For each of these business components, generate a low level monitoring model. A low level monitor model tracks instances of service components that together belong to an instance of the same business component.

4) In each low level monitoring model create an outbound event whose XML schema definition is determined in step 2 above. On the basis of the monitor set definition create (metric) fields in each outbound event that correspond

![Figure 6. Matching messages to composite instances.](image-url)
to the values of the business items that map to each monitor set key. Define a metric in each low level monitoring model that contains the location of each business item value that corresponds to a monitor set key.

5) Generate a high level monitoring model that contains an inbound event definition for each outbound event defined above, and metrics that correlate information received by each inbound event to piece together business component instances of the same composite. The metric definition depends upon the monitor set definition.

We now describe each of the above steps in detail.

B. Defining a Monitor Set using an Editor

We have created an editor with a simple interface for specifying monitor sets for a business composite (Fig. 7) and have implemented it as an eclipse plug-in for WebSphere Business Modeler [15].

![Figure 7. Monitor set editor.](image)

After creating a monitor set key and specifying its type, the user can then bind (Fig. 8) the key by clicking a button that invokes a list of business items in the business composite (which have one or more attributes of the type that match the monitor set key type). After selecting a business item, the user can click next to obtain a list of business item attributes (of the type that match the monitor set key type) and bind the monitor set key to one of these business item attributes.

![Figure 8. Binding monitor set keys to business item attributes.](image)

C. Generating an XML Event Definition

The monitoring solution code creates the following XML event definition (Fig. 9) for the outbound (and inbound) events and saves it in a file called “ModelOutboundEvent.XSD”. This event definition stores the invariant properties for MyMonitorSet_MS (created in Fig. 7). Notice that each CorrelationString element in Fig. 7 corresponds to the value of a monitor set key. Events based on this definition are inserted inside monitoring models described in the next section.

```xml
<xsd:schema

targetNamespace="http://MultiModuleMonitoring_lib"

xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:complexType name="ModelCorrelationEvent">

<xsd:sequence>

<xsd:element minOccurs="0" name="ModuleName" type="xsd:string"/>

<xsd:element minOccurs="0" name="ModuleId" type="xsd:string"/>

<xsd:element minOccurs="0" name="ActivityId" type="xsd:string"/>

<xsd:element minOccurs="0" name="EventId" type="xsd:string"/>

<xsd:element minOccurs="0" name="CorrelationString0" type="xsd:string"/>

</xsd:sequence>

</xsd:complexType>

</xsd:schema>
```

![Figure 9. XML event definition for inbound and outbound events.](image)

D. Generating Low Level Monitoring Models

For each key, \( k \), in the monitor set that the user selects, the monitoring solution code determines which business item attributes bind to it. For each business item attribute \( b \) in this list, the code determines which business components they serve as inputs to or outputs of and creates a list of these “dependent” business components, making sure to exclude the business composite. For each of these business components the code leverages existing support in WebSphere Modeler to generate a low level monitoring model if one does not exist already. A low level monitor model tracks instances of service components that together belong to an instance of the same business component. The code now inserts a metric and an outbound event inside the low level monitoring model. The metric and outbound event (shown below) contain as payload the value of the \( b \) business
item attribute that is an input to or output of this component and which binds to the key \( k \) of the monitor set selected by the user.

WebSphere Business Modeler and WebSphere Integration Developer have existing support to automatically generate low level monitoring models that receive messages from heterogeneous (including WS-BPEL) service components that contain the values of business item attributes for running instances of the business components they are referenced by. Due to space limitations we defer the details of how this is accomplished to a detailed future work report.

The XML snippet in Fig. 10 shows the location inside the low level monitoring models of different business components in the business composite. By comparing the values of these metrics received from different monitoring models that monitor parts of the end-to-end composite, the high level monitoring model is able to identify corresponding business component instances of the same end-to-end composite instance.

Figure 10. Defining a metric to store the value of monitor set key, LoanId.

```xml
<metric displayName="LoanId" id="LoanId" type="xsd:string"/>
<map>
<outputValue>
<singleValue expression = "ReceiveLoanApplication/ReceiveLoanApplicationENTRY/Input/LoanId"/>
</outputValue>
</map>
</metric>
```

Figure 11. Outbound event in the Loan_Initiation monitor model.

The monitoring solution code creates two low level monitoring models for the loan application, one for Loan_Initiation and one for Loan_Completion. The XML snippet in Fig. 10 shows the location inside the low level messages received from service components by the Loan_Initiation business component that corresponds to the value of business item attribute LoanId that corresponds to the key of MyMonitorSet_MS called LoanId.

The XML snippet in Fig. 11 shows the outbound event inserted inside the monitoring model for Loan_Initiation (note that it has a CorrelationString0 value that corresponds to the MyMonitorSet_MS key called LoanId).

E. Generating a High Level Monitoring Model

For each outbound event, the monitoring solution code creates an inbound event in a high level monitoring model and defines a correlation predicate that correlates the payload (values of monitor sets) of the inbound events received from the low level monitoring models of different business components in the business composite. By comparing the values of these metrics received from different monitoring models that monitor parts of the end-to-end composite, the high level monitoring model is able to identify corresponding business component instances of the same end-to-end composite instance.

Figure 12. Inbound event received by the high level monitor model.

```
The monitoring context instance is uniquely identified by this key. Monitoring context instances subscribe to events, update their state based on information in the events, and emit events to report a business situation. They are created, hosted, and terminated by WebSphere Business Monitor. After WebSphere Business Monitor has determined which
```

```
The XML snippet in Fig. 12 shows the inbound event received by the high level monitor model for the loan application process sent from the low level monitoring model for the Loan_Initiation business component. Notice the correlation predicate tag. The inbound event is matched by the high level monitoring model according to the value of the correlation predicate. In WebSphere Business Monitor, Monitoring models contain monitoring contexts, which define the set of information to be collected at run time. A monitoring context is created for the LoanID, and each monitoring context instance is uniquely identified by this key. Monitoring context instances subscribe to events, update their state based on information in the events, and emit events to report a business situation. They are created, hosted, and terminated by WebSphere Business Monitor. After WebSphere Business Monitor has determined which
monitoring context instances should receive an inbound event, it delivers the event to each event entry point with a matching correlation predicate. The high level monitoring model has a corresponding inbound event definition that it receives from the low level monitoring model of Loan_Completion. For the loan scenario, the correlation predicate validates whether the value of a LoanID received from the Loan_Initiation low level monitor model matches the value of a LoanID received from the Loan_Completion low level monitor model, which means that the corresponding Loan_Initiation and Loan_Completion business component instances belong to the same instance of the loan application business composite.

We were able to view successfully correlated instances of the end-to-end loan application business composite in Business Space powered by WebSphere [16].

V. CONCLUSIONS AND FUTURE WORK

This paper introduces a solution to seamlessly support end-to-end monitoring of business composites. The technical underpinning of the approach is the use of information invariants across components to relate service instances and events to instances of business composites. The result is the ability to identify and monitor the execution of end-to-end business composite applications as coherent business units. The use of our business correlation approach along with monitor sets covers some basic (but common) needs in business centric composite monitoring. This prototype implementation gives us an initial validation of the effectiveness of our approach. Downloadable plug-ins that execute our monitoring solution for IBM’s WebSphere Business Modeler [12] are publicly available [15]. These plug-ins enable users to choreograph a monitoring solution for multi-module BPEL processes from WebSphere Business Modeler.

We are currently developing extensions to the basic framework described here to support more complex business scenarios, particularly ones where multiple monitor sets are necessary to correlate an end-to-end business composite. Such scenarios require the specification of the mapping between different monitor sets and we are conducting research on designs of intuitive user interfaces to accomplish this. In parallel, we are implementing our monitoring strategy for business composites whose components are heterogeneous and not necessarily BPEL processes. We are thus evaluating the efficacy and extensibility of our solution in this comparatively more challenging business scenario. We are also investigating a diagram-driven monitoring specification, and conducting research on ways to automatically verify the efficacy and accuracy of user defined monitor sets.

REFERENCES


