# A Decentralized Mechanism
for Application Level Monitoring of Distributed Systems

Constantin Serban
Applied Research
Telcordia Technologies
One Telcordia Drive, Piscataway, NJ 08854
serban@research.telcordia.com

Wenxuan Zhang, and Naftaly Minsky
Department of Computer Science
Rutgers University
110 Frelinghuysen Rd., Piscataway, NJ 08854
{wzhang,minsky}@cs.rutgers.edu

***Abstract*—For a complex distributed system to be dependable, it must be continuously monitored, so that its failures and imperfections can be discovered and corrected in a timely manner. This work is concerned with the monitoring of large, open and heterogeneous systems, at their application level. Our objective is a monitoring technique that satisfies the following properties: scalability with respect to the size of the system and with the complexity of the monitoring task; the ability to deal reliably with heterogeneous components; and the ease and flexibility of deployment.**

Our approach to monitoring is based on a middleware called Law-Governed Interaction (LGI), which is a decentralized coordination and control mechanism.

***Keywords*-collaborative monitoring; self management; LGI;**

I. Introduction

It is widely recognized that the run-time monitoring of the operations of distributed software systems, which attempts to identify undesirable, suspicious, or unusual behavior, is critical to the integrity and dependability of the systems. This is true in a variety of ways: (a) monitoring is a necessary complement to the testing of a system [1], (b) it is required for the detection of intrusions, from within the system or from the outside [2], and (c) it is a critical element of any system management activity [3].

In this paper we introduce a mechanism designed for the application level monitoring of large and *open* distributed systems. By "open" we mean a system that consists of a distributed and heterogeneous collection of autonomous components, which are loosely coupled, may be written in different languages, running on different platforms, and designed, constructed, and even maintained by different organizations, as it often happens in collaborative applications.

Given the lack of knowledge of, and control over, the internals of the various components of such an open system, we adopt a black-box view of these components. That is, we assume no knowledge of, or control over, the internals of the components of the system to be monitored. And we intend to monitor only the flow of messages into and from the components engaged in collaboration.

We take the main purpose of such monitoring to be the speedy detection of *improper system behavior*, where the proper system behavior is assumed to be specified by a set of *normative properties*. These properties are assumed to be specified via predicates defined over the exchange of messages between various system components, and between them and the outside; and they may be sensitive to the history of such exchange. It is the violations of such properties that needs to be detected by the monitoring mechanism[1]; we will often refer to the detection of violations of a certain property p, as the monitoring of this property. This general approach has been called *specification-based monitoring* by Inverardi et. al. [2].

The monitoring mechanism introduced in this paper has been designed with the following set of objectives in mind:

1) *Generality:* The violation of arbitrary properties defined over the message exchange, and over the history of such exchange, should be detectable.
2) *Minimal and Scalable Detection time:* The delay between the violation of a normative property and its detection should be minimized; and should be made as independent as possible of the size of the system, and of the message traffic in it, thus providing *scalability.*
3) *Non-intervention:* The behavior of the system to be monitored should be shielded from interference from the monitoring activities (except, perhaps, of minor effects on the performance of the system).
4) *Ease of deployment:* The monitoring mechanism, and its detection logic should be easy to deploy; and it should be modular and flexible with respect to the addition of new components, and the changing of the API of existing components of the system system being monitored.
5) *Range of applicability:* The monitoring mechanism should be independent of the application domain, and of the language used for programming the system to be monitored.

[1] However, with a suitable definition of what we call "normative properties," this mechanism can be used for blanket monitoring, intended for gathering statistics about system behavior, if such is desired.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8336
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8336

Our approach to achieving these objectives is motivated by two observations that rule out most conventional approaches to monitoring the type of systems we have in mind.

First, since we assume no knowledge of, or control over, the internals of the various system components, we cannot instrument the components themselves for sensing the message exchange—as it has been done in [4]. Moreover, we cannot rely on each component to provide a monitoring interface, as employed by most conventional system-management standards—such as SNMP (Simple Network Management Protocol) [5], and WSDM (Web Services Distributed Management) [3].

Second, since the system to be monitored may be widely distributed over the Internet, we cannot rely on instrumenting the network for sensing the messages—as it has been done in [6], in particular.

Thus, we have to rely on a middleware to sense the flow of messages in the system, and to facilitate the detection of violations of the given normative properties. Such middleware needs to satisfy the following requirements in order to be consistent with our objectives: (a) it needs to be *decentralized*, in order to be scalable, as required by objective 2 above; (b) it needs to be *stateful*, in order to be able to handle properties defined over the history of communication, as required by objective 1; and (c) it needs to be secure, to protect against malicious attacks. A middleware that satisfies these requirements is a decentralized control mechanism called Moses (based on the concept of Law Governed Interaction (LGI)[7], [8] . We have chosen this middleware as the basis for our monitoring mechanism.

The rest of this paper is organized as follows. Section II introduces an abstract model of a monitored system; Section III presents an overview of the LGI mechanism; Section IV presents the architecture of our monitoring mechanism; Section V discusses related work; and Section VI concludes the paper.

II. A Model of a Monitored System

We start in Section II-A with an abstract model of a monitored system, in line with our general approach. To elaborate on this model we introduce in Section II-B a case study of a system to be monitored; and then, in Section II-C, we provide a broad classification of what we have called normative properties. We conclude this section with a discussion of the problem at hand.

A. An Abstract Model

We define a monitored system to be a triple $\langle B, P, I \rangle$, where:

- B is the *base system* being monitored. It consists of a collection $\{c_1, ..., c_k\}$ of components, called *base components*, or *b-components*. The messages exchanged between the b-components, and between them and their environment, are called *base messages*, or *b-messages*.

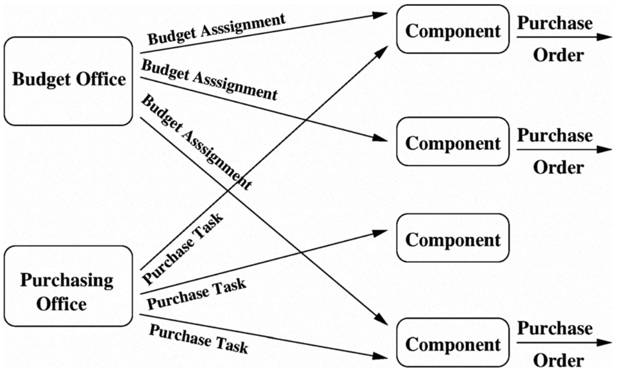

Figure 1. EP:Enterprise Purchasing Activity

- P is a given *set of normative properties* $\{p_1, ..., p_n\}$, defined over the exchange of b-messages. These are properties that the base system needs to satisfy, and whose *violations* are to be detected by the monitoring mechanism.
- I is the monitoring *instrumentation* associated with the base system, and whose purpose is to detect violations of properties in P.

The immediate purpose of the monitoring is the unintrusive *detection* of any violation of any of the properties in P, and the reporting of such a violation to a designated component (or components) of the instrumentation I. We are aware of the fact that some of the detected violations would require corrective actions to be applied to the base system—which is, indeed, one of the main purposes of monitoring. But although such corrective actions are part of our research plan, they are beyond the scope of this paper.

Several additional comments about this model are in order. First, we make no assumptions about the base systems, except that the various b-components send and receive b-messages via the Moses (LGI-based) middleware. This, is required for reasons outlined in the Introduction. Second, we do not specify in this paper, the source of the normative properties. But the general expectation is that they would be formulated by such stakeholders as the programmers of the various components, the system architects, and the people who maintain the systems throughout its evolutionary lifetime. Finally, although this model does not specify the structure of the instrumentation I, this is an important part of our mechanism, which will be addressed in Section IV-A.

B. A Case Study

We will coach further discussions of this model in terms of a case study we conducted for experimenting with our monitoring approach and for evaluating its efficacy. In this case study we monitored various purchasing activities in a simulated distributed enterprise, to which we refer as the Enterprise Purchasing (or EP). A somewhat simplified

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8336
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8336

description of these activities, which we will use in this paper, is depicted schematically in Figure 1.

The purchasing activities considered in this case study are regulated by two distinguished components: the *Budget Office* (*BO*), and the *Purchasing Office* (*PO*). The budget office assigns purchasing budgets to other system components, by sending them messages of the form `budgetAssignment($)`. The purchasing office component provides other components with purchasing tasks, of a form that does not concern us here. Depending upon the circumstance, a purchasing task can be interpreted either as a permission to purchase the specified items or as an instruction to do so by a specified deadline. The purchasing activities themselves are carried out by various components sending purchase orders to various internal servers, or to servers over the Internet[2].

Note that all the examples in this paper are coached in terms of this simple case study; space limitations precluded discussions of the results of the experiments we performed on it.

C. On the Nature of the Normative Properties

Structurally, normative properties are predicates defined over the flow of b-messages, and over the history of such flow. In this section we classify these properties into three types, which we call: *local*, *local-regular*, and *non-local*. In the following section we discuss the monitoring techniques required by each of these classes.

1) Local Properties: A local property is one that is defined over the incoming and outgoing messages of a single component. Such properties can be concerned with the following types of issues:

- The conformance of the incoming and/or outgoing messages with certain patterns, syntax, or formats. An example of such property is: *"budget requests arriving at the budget office should have the following structure:* `budget-Request(Amount)`*"*.
- A desired relationship between the incoming and outgoing messages. As an example, consider the following: *"There should be no more than M purchase tasks pending at a specific component c."*, where a purchase task is interpreted as a direction to buy an item. This property is local, because it can be verified by observing all the purchase task messages arriving at the individual component *c*, and all the purchase orders issued by it.

Note that although a local property is defined with respect to a single component, it might characterize not only the behavior of this component, but that of the components it interacts with. For example a property defined over the delay between a purchase order sent by component c to some server s, and the arrival at c of a reply from s, reflects information about the behavior of s, although it is a local property at c.

2) Local-Regular Properties: A property p is called local-regular, if it is a local property that needs to be satisfied by all components in the base system, or by a well defined subset of system components, named the *domain of p*.

The following is an example of such a property: *"The total cost of the purchase orders issued by any system component does not exceed the budget assigned to it by the budget office."* This property is local because its violation can be detected at each individual component, by observing the budget assignment messages it receives from the budget office, as well as the purchase order messages it sends; and it is also regular since it applies to all the components in the base system.

Local-regular properties are particularly important because they represent system-wide standards, which are essential to the manageability and comprehensibility of any large scale system. Moreover, as we shall see, the detection of violations of such a property is challenging, because they can occur at any component in its domain.

3) Non-local Properties: A non-local property p is a predicate defined over the history of communication of multiple, distributed, components. Or, in other words, it is a property that is neither local, nor local-regular. As an example, consider the following property, again in the enterprise purchasing context : *"if the purchasing office broadcasts a list of k items [$M_1 ... M_k$] to be bought jointly by* all *the components engaged in EP, then no item should be bought more than once."* Since a single item can be purchased by several different components, at different times, the violation of this property cannot be detected locally.

D. On the Monitoring of Normative Properties

We offer here two general observations about the monitoring of normative properties, followed with more specific observations about the monitoring of the three classes of properties introduced above.

First, the monitoring of a given property needs to be carried out as close as possible to the relevant b-components, namely to the components in which the violation of this property can be detected. Such *localization* is critical to the scalability of monitoring, with respect to both the size of the base system, and the traffic of messages in it. Second, note that the detection of violations of history dependent properties often does not require the keeping of the full relevant history of communication. For example, in order to determine that a component does not exceed its communication bandwidth quota, it is not necessary to record all the messages that the component has sent and received; it is sufficient to maintain a count that represents the cumulative size of such messages. Thus, properties in P can be expressed in terms of such synoptic abstractions, which we call *monitoring states*.

[2] We assume that there is no centralized clearing house that mediates all the purchase orders in the enterprise.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8336
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8336

Now, *local properties*, can be monitored very scalable by employing wrapper techniques, where individual wrappers are responsible for observing the local property of a wrapped component. These wrappers need to be stateful for an efficient monitoring of history dependent properties.

The monitoring of *local-regular* properties is significantly more challenging. Individual wrappers are not an appropriate solution due to the difficulty to ensure that *all* the wrappers in the domain of a given property p employ the right logic to monitor p. This is particularly problematic because a single wrapper may be required to monitor several such properties.

The monitoring of *non-local* properties is even more problematic. Each such property involves a relationship between distributed events (sending and arrival of messages), occurring in several (may be many) components, and spanning a certain period of time. The occurrence of such events would have to be reported to some aggregator—one of the components of the instrumentation division I—which will attempt to detect the violation of the property at hand. Such aggregation unavoidably reduces the scalability of the monitoring mechanism. But such lowering of scalability can be minimized by using different aggregators for different properties, as we intend to do.

III. A Brief Introduction to LGI

Law-Governed Interaction (LGI) has been originally developed as an access control (AC) and coordination mechanism for distributed systems. More precisely LGI is a message-exchange mechanism that allows an open and heterogeneous group of distributed *actors* to engage in a mode of interaction *governed* by an explicitly specified and strictly enforced policy, called the "law" of this group. By "actor" we mean an arbitrary process, whose structure and behavior is left unspecified. An actor engaged in an LGI-regulated interaction, under a law $\mathcal{L}$, is called an *$\mathcal{L}$-agent* (or simply an "agent," when the identity of the law does not matter); the messages exchanged under a given law $\mathcal{L}$are called $\mathcal{L}$-messages; and the group of agents interacting via $\mathcal{L}$-messages is called an $\mathcal{L}$-community. LGI turns a set of disparate actors, which may not know or trust each other, into a *community* of agents that can rely on each other to comply with the given law $\mathcal{L}$. This is done via a distributed collection of generic components called *private controllers*, one per $\mathcal{L}$-agent, which mediate all interactions between these agents, subject to a specified law $\mathcal{L}$(as illustrated in Figure 2).

The private controllers are hosted by what we call *controller pools*—each of which is a process of computation that can operate several (in the hundreds) private controllers, thus serving several different agents, possibly subject to different laws[3]. This section provides only a very brief overview of LGI. For more information, the reader is referred to the LGI manual [8] , web references , and to a host of published papers.

Agents and their Private Controllers:: An $\mathcal{L}$-agent x is a pair $x = \langle A_x, T_x^{\mathcal{L}} \rangle$, where A_x is an *actor*, and $T_x^{\mathcal{L}}$ is its *private controller*, which mediates the interactions of A_x with other LGI-agents, subject to law $\mathcal{L}$. The role of the controllers is illustrated in Figure 2, which shows the passage of a message from an actor A_x to A_y, as it is mediated by a pair of controllers, first by $T_x^{\mathcal{L}}$, and then by $T_y^{\mathcal{L}}$—both operating, in this case, under the same law.

The Concept of Law, and the Semantics of LGI:: Our concept of law differs structurally from the conventional concept of an AC policy (such as that of XACML) mostly in that it is local—in the sense that an LGI law can be complied with, by each member of the community subject to it, without having any direct information of the coincidental state of other members. This locality is important because it enables the decentralization of law enforcement, and thus provides for scalability even in the case of stateful policies.

It is important to note that, despite the fact that locality constitutes a strict constraint on the structure of LGI laws, it does not reduce their expressive power, as has been proved in [8]. In particular, despite its *structural locality*, an LGI law can have *global effect* over the entire $\mathcal{L}$-community—simply because all members of that community are subject to the same law—and can, thus, be used to establish *mandatory*, community-wide constraints.

The following is an abstract definition of LGI laws: *A law $\mathcal{L}$is a* function $L(e, s)$*, which returns a list of primitive operations, called the* ruling *of the law, for any possible regulated-event e and any possible control-state s.*

Note that the ruling of the law is not limited to accepting or rejecting a message, but can mandate any number of operations like changing and issuing a message, providing laws with a strong degree of *initiative*. Also, the operations in the ruling may update the control-state of the agent, thus providing for *stateful policies*. Finally, the ruling may impose an *obligation* on the agent, which provides a *proactive capability*. More specifically, LGI laws are formulated using an event-condition-action pattern. In this paper we will depict a law using the following pseudo-code: $upon\langle event\rangle if\langle condition\rangle do\langle action\rangle$, where the $\langle event\rangle$ represents one of the regulated events, the $\langle condition\rangle$ is a general expression formulated on the event and control-state, and the $\langle action\rangle$ is one or more operations mandated by the law. We currently use two languages for specifying laws—one is based on Prolog, and the other one on Java. Note that, despite the pragmatic importance of a particular language, the semantics of LGI are basically independent of the language.

The Hierarchical Organization of Laws:: LGI provides for the laws to be organized into hierarchies. Each such hierarchy, or tree, of laws $t(\mathcal{L}_0)$, is rooted in a "base law"

[3] We often use the term "controller" for either a controller-pool or for a private-controller—expecting the ambiguity to be resolved by the context.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8336
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8336

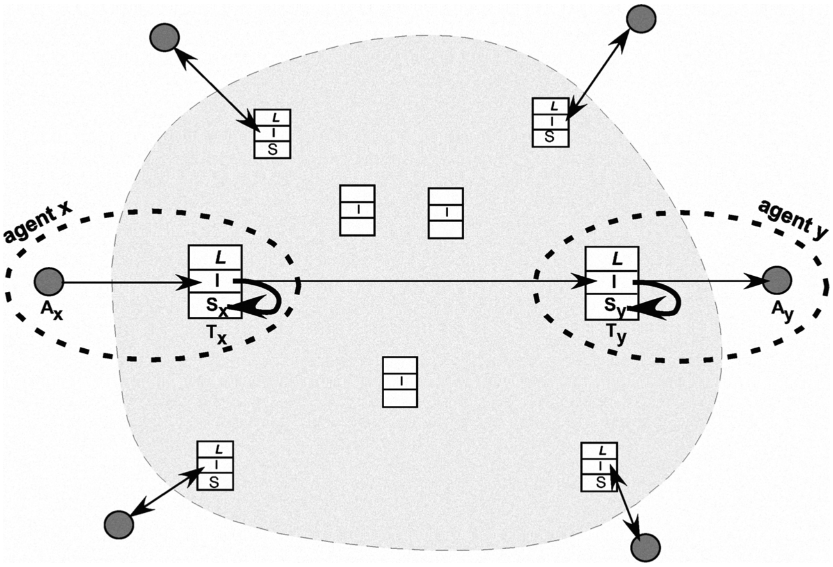

Figure 2. Interaction via LGI: Actors are depicted by circles, interacting across the Internet (lightly shaded cloud) via controllers (boxes) operating under law L. Agents are depicted by dashed ovals consisting of actor–controller pairs. Thin arrows depict messages; thick arrows depict state modifications.

$\mathcal{L}_0$. Each law in $t(\mathcal{L}_0)$ is said to be (transitively) *subordinate* to its parent, and (transitively) *superior* to its children. Given a pair of laws $\mathcal{N}$ and $\mathcal{M}$ in $t(\mathcal{L}_0)$, we write $\mathcal{N} \prec \mathcal{M}$ if $\mathcal{N}$ is subordinate to $\mathcal{M}$.

Semantically, the most important aspect of this hierarchy is that if $\mathcal{N} \prec \mathcal{M}$ then $\mathcal{N}$ *conforms* to $\mathcal{M}$, in the sense that law $\mathcal{N}$ satisfies all the provisions of its superior law $\mathcal{M}$.

LGI provides a very efficient mechanism, outlined in [9], for constructing such law-trees, top-down. This is done, broadly, as follows. Starting from a given law $\mathcal{M}$ in a tree, one can *refine* it into a subordinate, and conforming, law $\mathcal{N}$. Once formed, the laws in the law ensemble are independent entities, with no dynamic relationship between a superior law and its subordinates—although different laws in the tree may be able to interoperate. The law tree is, therefore, a very modular structure.

The Performance Evaluation of LGI: The overhead incurred by LGI is quite affordable, and is negligible for many applications communicating over WANs. We have evaluated the performance of the current implementation of LGI on various platforms: Sun Solaris, Linux and Windows, and with different Java Run-time Environments (see [8] for more details). With a commodity Linux workstation (Intel 3.2GHZ CPU, 1GB memory) and SUN JVM 1.4.0, the typical overhead brought by a single event evaluation (as is the verification of a message passing through a controller,) is about 50 micro seconds, which is very small compared with typical communication time between two Java applications using TCP/IP (usually about 1ms for LAN communication). The typical overhead brought by actor-to-controller communication is about 130 micro seconds, which is also small. In terms of throughput, our evaluations show that a single controller-pool can handle more than 18,000 events per second, which is stable even when it holds more than 1,000 private controllers.

For monitoring purposes, in addition to the small communication overhead, the LGI mechanism also shows good scalability with the number of components involved in communication, the number of messages exchanged between these components, as well as the number of properties to be monitored. This scalability is addressed throughout the rest of this paper.

IV. LGI-BASED MONITORING

We start in Section IV-A, with the description of an architecture of a monitored system $\langle B, P, I \rangle$, based on the model of such system introduced in Section II-A. In Section IV-B we outline the manner in which this architecture can be used to monitor various kinds of normative properties. In Section IV-C we discuss the deployment of this LGI-based monitoring.

A. The Monitoring Architecture

According to the model introduced in Section II-A, a monitored system consists of two part, which we call *divisions*: the base system B, and the instrumentation I, as depicted in Figure 3. We start by elaborating on these divisions, and specifying the relationship between them. We then discuss the various types of messages involved in this architecture. We conclude this section with a discussion of the organization of the laws that facilitate the monitoring itself.

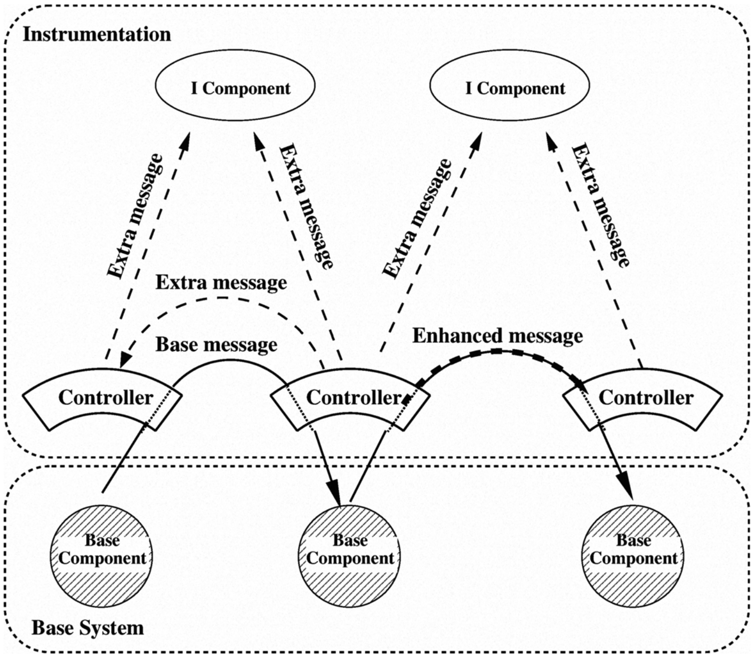

Figure 3. LGI-based Monitoring Architecture

1) The Base Division:: We assume that b-components communicate via LGI. That is, each b-component is associated with its private LGI controller that mediates all its communication. Thus, b-components become LGI agents under this architecture (using the terminology introduced in Section III). The laws under which these agents operate—and there are several such laws, as we shall see below—are designed for the monitoring of the set P of normative properties.

It should be pointed out that no automatic means for converting a given set of normative properties—stated, for example, in first order logic,—into LGI laws are presently available. However, research is underway for doing such a conversion for a subset of local and local-regular properties.

2) The Instrumentation Division:: The LGI controllers associated with the b-components belong to the instrumentation divisions I, as shown in Figure 3. But this division contains additional components, called *i-components.*

Each i-component c is responsible for the detection of the violations of one, or more, non-local properties in P. This is to be done by accepting notices about the relevant messaging events occurring in various b-components, sent to x by the controllers associated with the various b-components; and then by analyzing the notices aggregated in them. Different i-components may thus be responsible for monitoring different properties in P, or, sometimes, the same property with respect to different parts of the base system. We do not discuss here the internal functionality of the i-components.

3) The Messages:: This architecture employs the following types of messages:

- *Base Messages:* These are the original messages of the base system, which have not been modified by the monitoring mechanism.
- *Enhanced Messages:* These are base messages to which extra information is appended by the controller of the source of the message, and intended for the controller of its destination. The purpose of such piggybacking is to facilitate the verification of certain types of non-local properties, as explained in Section IV-B. As we shall see, this extra information is stripped down by the controller of the destination, so that the b-components themselves receive only the original b-messages.
- *Extra Messages:* These are messages exchanged between different parts of the instrumentation, in order to facilitate the detection of the failures of the normative properties.

4) The Organization of the Laws: The controllers associated with the various b-components operate under a three-level hierarchy of laws, depicted in Figure 4. Before explaining the nature and purpose of this hierarchy we point out that it is not the only way for organizing the laws under our monitoring architecture, but it is a reasonable way with which we have experimented, and it is easy to imagine variants of it.

The root of this law-hierarchy is, what we call, the *non-intervention* law, or $\mathcal{L}_{NI}$. This law has exactly one immediate subordinate, called the *base system* law, or $\mathcal{L}_B$. And $\mathcal{L}_B$, in turn, has a set of so called *component-laws* $[\mathcal{L}_1....\mathcal{L}_k]$ as its subordinates.

Now, each b-component c operates under exactly one

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8336
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8336

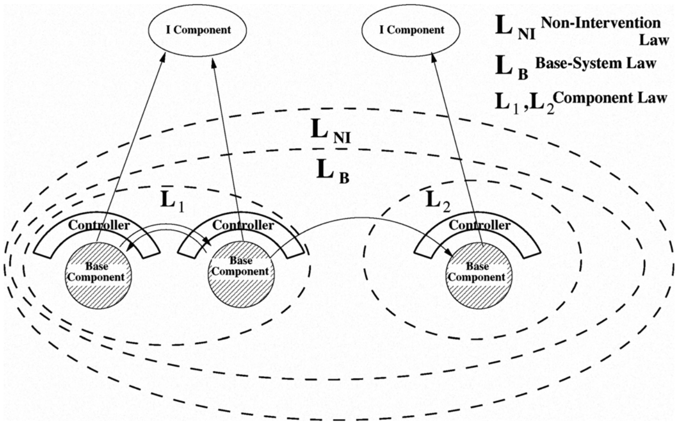

Figure 4. The Organization of Laws

component-law $\mathcal{L}_i$. Given the conformance semantics of the LGI law-hierarchy, this means that the messages sent and received by component c are mediated in conformance not only with $\mathcal{L}_i$, but with laws $\mathcal{L}_B$ and $\mathcal{L}_{NI}$ as well. We are now in a position to discuss the functions of the various laws in this hierarchy, starting with the root law.

The root law $\mathcal{L}_{NI}$: is written to ensure non-intervention in the operations of the base system. More specifically, this law provides the following guarantees: (a) b-messages are delivered to their destination (so, they cannot be blocked by subordinate laws); (b) enhanced messages are stripped down from their piggybacked information prior to their delivery at the destination, and (c) no extra messages are delivered to b-components. These provisions are guaranteed, because they cannot be violated by any subordinate law, which, by its definition, conforms to law $\mathcal{L}_{NI}$.

Note that this particular law is essentially part of the monitoring framework, that can be used for any type of base system, and for any given set of normative properties. This is the way we achieve our objective of non-intervention.

The law $\mathcal{L}_B$,: which is subordinate to $\mathcal{L}_{NI}$ but superior to all component-laws $\mathcal{L}_i$, can provide the following functionalities: First it can ensure that any given local-regular property p is monitored in the same way at all b-components in the domain of p, which may be the entire base system. Second, it can be written to monitor non-local normative properties.

A Component Law $\mathcal{L}_i$: is responsible for monitoring the local properties defined for individual component, or for a group of components that have identical local properties.

The fact that local properties are implemented as part of a law $\mathcal{L}_i$, which is subordinate (and thus conforming) to law $\mathcal{L}_B$, provides with an important degree of flexibility, as follows. If an individual component is changed or a new component appears in the system, its property can be implemented in its own law, independently of the local-regular or non-local properties that may apply to the same component. From the perspective of other controllers, $\mathcal{L}_B$ prevents the changes in $\mathcal{L}_i$ to be visible to the rest of the monitoring instrumentation. The conformance of $\mathcal{L}_i$ to $\mathcal{L}_B$ also ensures that the more stable non-local and local-regular properties are shielded from changes in local properties that are bound to be tied to a certain implementation of a particular component.

B. The Monitoring of Various Types of Properties

Below we will show the mechanism for verifying some of the properties for the Enterprise Purchasing activity, and we will present the corresponding fragments of $\mathcal{L}_B$. Due to space limitations we will not show $\mathcal{L}_{NI}$ and $\mathcal{L}_i$.

Local-Regular Properties: A local-regular property can be monitored by observing the communication of every component in the system or the domain of that property. A local-regular property has to be incorporated as part of $\mathcal{L}_B$, since *all* the components of the system are subject to it. Let us consider again the local-regular property introduced in Section II-C2: *"The total cost of the purchase orders issued by any system component does not exceed the budget assigned to it by the budget office."*. The fragment of $\mathcal{L}_B$ concerned with this property is presented in Figure 5. Let us assume that the budget office assigns budgets to other components using messages of the form `budgetAssignment(BudgetAmount)`, and that the purchase orders are messages of the form `purchaseOrder(Item, Amount)`. The verification process is organized in two rules. Rule $\mathcal{R}1$ is invoked every time a *budgetAssignment* message arrives at the controller of a component. Before the delivery of such message, the controller updates a variable called *Budget* in its control-state. In Rule $\mathcal{R}2$, every time a component issues a *purchaseOrder* message, the amount in the purchase order is compared against the same *Budget* variable in the control-state; a violation is identified if the former exceeds the latter. Subsequently, the amount of the purchase order is deduced from the budget of the component, and the controller propagates the message without change. Note that, once a violation is detected, the controller reports it by sending a message to a specialized instrumentation component responsible for collecting such violations, as mentioned in Section II.

Note that this property is verified entirely locally, at the controller of each component, thus the implementation scales well with the number of components, and with the number of messages witnessed by a component. Also note that local properties are treated similarly, since they are verified by individual controllers. The monitoring of local properties, however, is implemented as part of $\mathcal{L}_i$, thus it applies to individual components, and not to the whole system.

Non-Local Properties: Non-local properties are properties that cannot be verified locally, by observing the communication of a single component. One method we use for veri-

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8336
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8336

```
Preamble:  Law(LB, conforms(LNI))
R1)
 upon arrived(BO,
     budgetAssignment(BudgetAmount), X)
     do(Increase(Budget, BudgetAmount))
     do(Deliver)
R2)
 upon sent(X,
     purchaseOrder(Item,Amount), Y)
     if(Amount > Budget) {
     do(ReportViolation) }
     do(Decrease(Budget, Amount)
     do(Forward)
```

Figure 5. Fragment of $\mathcal{L}_B$ verifying budget consistency

fying such properties relies on the specialized *i-components*. The i-components can collect information supplied by the local controllers and thus they can verify a given property. We do not assume a single i-component for the entire system: different properties can be verified using different i-components. Accordingly this solution is scalable with the number of properties to be verified. This is an important feature not supported by most monitoring mechanisms which either use a single component that collects information from the entire *infrastructure*, or a *physically* organized hierarchy of specialized components for centralizing such information.

Let us consider the non-local property introduced in II-C3, in the context of the enterprise purchasing : *"if the purchasing office broadcasts a shopping list of k items [$M_1 ... M_k$] to be bought jointly by* all *the components engaged in EP, then no item should be bought more than once."* In order to verify this property, we employ an i-component, called the Item Aggregator, that collects information about all the items in the shopping list. We assume that the joint purchasing is assigned through a message of the form `jointPTask(ItemList)` broadcasted to all the components in EP. A purchase order is represented by a `purchaseOrder(Item, Amount)` message. The controller of each component will identify the purchase orders concerning the items in the shopping list, and will report them to the Item Aggregator using extra-messages. The Item Aggregator can subsequently verify whether an item has been purchased twice.

Figure 6 shows a fragment of the $\mathcal{L}_B$ law designed to monitor this property. Rule $\mathcal{R}1$ is invoked whenever a component receives a joint shopping list from the purchasing office. A copy of this list is saved in the local control-state. In Rule $\mathcal{R}2$, whenever a component issues a purchase order, the item in the purchase order is compared against the *ShoppingList*. If present, an *inform* extra-message containing the item is sent to the Item Aggregator. Since this rule applies to all the components in the system, the Item Aggregator will collect the information about each item in the *ItemList* purchased by any component; consequently it can verify any duplicate purchase. Here we do not show how the Item Aggregator will employ this trivial verification. We also assume that the address of the Item Aggregator is known in advance.

```
Preamble:  Law(LB, conforms(LNI))
R1)
  upon arrived(PO,jointPTask(ShopLst),X)
      do(Save(ShopLst)) do(Deliver)
R2)
  upon sent(X, purchaseOrder(Item), Y)
      if(Item in ShopLst) {
      do(Fwd(ExtraMsg(Aggregator,Item)))}
      do(Forward)
```

Figure 6. Fragment of $\mathcal{L}_B$ verifying a joint purchase

The verification of many non-local properties can also be performed without the help of i-components, in a more scalable manner. The controllers can collaborate instead in order to verify such properties. Consider the following example. In the EP system, let us assume that a `Human Resources` (HR) office dynamically assigns buyers into different departments, using `assignDept(DptNo)` messages. Let us furthermore assume that the buyers within a department are allowed to transfer budget to each other, using a `assignBudget(B)` message. Consider the monitoring of the following property: *" no buyer transfers its budget outside its own department"*. This property is non-local, since it cannot be evaluated by observing the communication of a single component. LGI-based monitoring verifies this property as follows. The controller of each component will first detect each incoming `assignDept(DptNo)` message, and it will save the `DptNo` value for each component. Whenever a component sends a `assignBudget(B)` message to another component, the controller will *enhance* the message with the previously saved `DptNo` value, thus *piggybacking* this information to the controller of the receiver. Upon arrival, the controller of the receiver will thus have both `DptNo` values at hand, and it will be able to verify the property. The fragment of $\mathcal{L}_B$ that verifies this property is presented in Figure 7.

```
Preamble:  Law(LB, conforms(LNI))
R1)
  upon arrived(HRO,assignDept(DN),X)
      do(save(DN)) do(Deliver)
R2)
  upon sent(X, assignBudget(B), Y)
      do(Fwd(X,[assignBudget(B),DN],Y))
R3)
  upon arrived(Y,[assignBudget(B),D],X)
      if(D <> DN) {do(ReportViolation)}
      do(Deliver(Y,assignBudget(B),X))
```

Figure 7. Fragment of $\mathcal{L}_B$ verifying the budget transfers

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8336
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8336

C. The Deployment of the LGI-based Monitoring

In order to enable the LGI-based monitoring for a base system, the b-components have to be programmed such that they use the LGI primitives as their only means of communication. Additionally, an instrumentation infrastructure has to be deployed in order to enable this communication and to verify the monitored properties.

The monitoring instrumentation consists of a set of controllers operating under laws as specified in IV-A4, and a number of i-components designed to collect the information for verifying aggregate properties. The controllers are to be deployed globally across the Internet, and in reasonably close proximity to any component in the base system. Similarly, before the monitored system starts its execution, the i-components should be deployed and their addresses should be known in advance.

In order to enable an LGI message exchange between the components in the base system every component has to be associated to a specific controller. This controller is to operate under a specific law hierarchy that reflects the properties observed by that component. Additionally, every component is to be assigned a control-state reflecting both the identity of the component, and the state that might be necessary for verifying the properties. These operations take place automatically during the initialization of each components, with the help of a special component called the *Registry*.

The following exchanges are performed during the initialization stage:

- the component will contact the registry and provide it with its identity;
- the registry will reply with the address of an associated controller, a hierarchy a laws, and an initial control-state specific to this component
- the component will adopt the provided controller, under the specified laws, and with the specified initial control-state

After this initial exchange, the b-components will operate as black boxes, according to their internal logic, and will use the send and receive LGI primitives for communication. Automatically, their traffic will be directed through their corresponding controller, and the traffic will become subject to verification of the properties implemented in the corresponding hierarchy of laws.

Note that this scheme provides robustness against incomplete or faulty instrumentation as follows. If a certain b-component is not associated with any controller, that component will not be able to communicate with other components that are associated with a controller. This is a consequence of a specific protocol and authentication mechanism employed in the controller-to-controller communication. Also, if a b-component is associated, by mistake, with an improper law $\mathcal{L}_B^{'}$, the component again will not be able to communicate with the other components operating under $\mathcal{L}_B$, due to the LGI mechanism that verifies that the source and the destination of a message operate under the same law.

V. Related Work

There is a large number of publications that address the issue of monitoring in distributed systems, adopting a wide range of methods for achieving this purpose. Most such works do not share some of our objectives. In particular, (1) [4], [10], [11] take a white box view of the components; (2) [12], [13], [11], [4] perform their monitoring off-line; and (3) [14], [15], [16] monitor specialized kinds of systems, like computer clusters and networks. Below we will discuss a number of recent efforts that are more closely related to ours, but which do not entirely satisfy our requirements.

Spanoudakis et. al. [17] present a model for monitoring a system built according to the Service Oriented Architecture, and whose components (i.e. web-services) are considered black-boxes. The system subject to monitoring is assumed to interact through a centralized BPEL engine, and the monitoring itself is centralized, thus unscalable for a widely distributed application domain. In the same context, Machiraju et. al. [18] present the Web Services Management Network (WSMN), a mechanism for monitoring Service Level Agreements (SLAs) using a network of local and cooperating intermediaries. Similar to us, they have a black-box view of the components, and their approach is decentralized. However, they do not have a mechanism to deploy and enforce a global policy systematically, on every intermediary. Accordingly, their mechanism is not suitable to monitor the important local-regular and non-local properties.

Khanna et. al. [6] provide a generic monitoring architecture whose objectives are very similar to ours. However, this work is based on the assumption that the instrumentation has access to every local area network (LAN) used by the system at hand, so that the monitors can eavesdrop on the communication. Their assumption is often not valid for large open systems, as we have pointed out before. Another difference is that [6] organizes the monitors into a physical, thus rigid, hierarchical structure. As argued in Section IV-B, this organization does not scale well with the number of properties and can introduce additional, unnecessary traffic among the instrumentation components. Finally, their local monitors listen to the system and process the messages in an asynchronous manner. Thus, such monitoring does not facilitate immediate corrective response following the detection of a property violation.

The closest work to ours, in terms of both its objectives and its approach, is that of Inverardi et. al. [2]. This paper proposes a distributed monitoring mechanism for verifying properties defined over the interactions in a system, whose components are assumed to be black boxes. The authors start with a global state-machine based model of what we call the normative system behavior. Once formulated, this

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8336
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8336

global model is distributed automatically into a collection of local adaptors, or filters, representing the monitoring instrumentation. However, as this paper admits "[this] approach might still suffer of the well known state-explosion problem" [2], when generating the global state-machine, as well as during the process of its distribution to local filters. Another problem with this approach is its lack of flexibility with respect to the addition of new components, since such an addition would require reformulation of the entire global system model.

VI. Conclusions

This paper introduced a specification based monitoring mechanism for open distributed systems, whose main purpose is a speedy and scalable detection of *improper system behavior*, where the proper system behavior is assumed to be specified by a set of *normative properties* defined over the flow of messages in the system. The monitoring is carried out by employing a stateful and decentralized control mechanism called LGI, without any knowledge of, or control over, the internal behavior of the components of the system.

The speedy and scalable monitoring is accomplished by carrying out much of the monitoring close to the occurrence of the relevant communication events. Thus, structurally *local* and *local-regular* properties are monitored entirely locally, while *non-local* properties are monitored either semi-locally, or by means of a collection of aggregators, in a semi-decentralized manner.

Finally, the monitoring architecture proposed here protects the base system being monitored from any interference by the monitoring activities. However, this architecture has been designed to be coupled with a *corrective mechanism* that can apply appropriate remedial measures, in response to problems uncovered by the monitoring of the system. The discussion of this coupling between the monitoring and the corrective mechanisms is beyond the scope of this paper.

References

[1] G. Spanoudakis, C. Kloukinas, T. Tsigritis, K. Androutsopoulos, C. Balls, and D. Presenza, "A4.d1.1 review of the state of the art," *European Serenity Project: System Engineering for Security and Dependability*, 2006.

[2] P.Inverardi, L.Mostarda, M.Tivoli, and M.Autili, "Synthesis of correct and distributed adaptors for component-based systems: an automatic approach," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, 2005.

[3] O. T. Committee, "Web Services Distributed Management: Management Of Web Services (WSDM-MOWS) v1.0, OASIS standard," March 2005, available from: http://www.oasisopen.org.

[4] M.Y.Chen, A.Accardi, E.Kiciman, J.Lloyd, D.Patterson, A.Fox, and E.Brewer, "Path-based failure and evolution management," in *NSDI'04*, 2004, pp. 309–322.

[5] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol (SNMP)," 1990, rFC: 1157, available from http://www.ietf.org/rfc/rfc1157.txt.

[6] G.Khanna, P.Varadharajan, and S.Bagchi, "Automated online monitoring of distributed applications through external monitors," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, pp. 115 – 129, April 2006.

[7] N. Minsky and V. Ungureanu, "Law-Governed Interaction: a Coordination and Control Mechanism for Heterogeneous Distributed Systems," *TOSEM, ACM Transactions on Software Engineering and Methodology*, vol. 9, no. 3, pp. 273–305, July 2000.

[8] N. H. Minsky, "Law-Governed Interaction (LGI): A Distributed Coordination and Control Mechanism (An Introduction, and a Reference Manual)," Rutgers University, Tech. Rep., June 2005.

[9] X. Ao and N. H. Minsky, "Flexible regulation of distributed coalitions," in *LNCS 2808: the Proc. of the European Symposium on Research in Computer Security (ESORICS) 2003*, October 2003.

[10] K.Sen, A.Vardhan, G.Agha, and G.Rosu, "Efficient decentralized monitoring of safety in distributed systems," in *Proceedings of the 26th International Conference on Software Engineering (ICSE2004)*, 2004, pp. 418–427.

[11] P. Reynolds, J. L. Wiener, J. C. Mogul, M. A. Shah, C. Killian, and A. Vahdat, "Pip:detecting the unexpected in distributed systems," in *NSDI'06*, 2006.

[12] T.Gschwind, K.Eshghi, P.K.Garg, and K.Wurster, "Webmon: a performance profiler for web transactions," in *4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, 2002.

[13] M.K.Aguilera, J.C.Mogul, J.L.Wiener, P.Reynolds, and A.Muthitacharoen, "Performance debugging for distributed systems of black boxes," in *SOSP 2003*, 2003.

[14] M.L.Massie, B.N.Chun, and D.E.Culler, "The ganglia distributed monitoring system: Design, implementation, and experience." *Parallel Computing*, vol. 30, no. 7, July 2004.

[15] P.Barham, A.Donnelly, R.Isaacs, and R.Mortier, "Using magpie for request extraction and workload modeling," in *Proc.OSDI*, San Francisco, Dec. 2004, pp. 259–272.

[16] K. Park and V. S. Pai, "CoMon: A mostly-scalable monitoring system for PlanetLab," *Operating Systems Review*, vol. 40, no. 1, January 2006.

[17] G. Spanoudakis and K. Mahbub, "Non-intrusive monitoring of service-based systems," *International Journal of Cooperative Information Systems*, vol. 15, no. 3, pp. 325–358, 2006.

[18] V. Machiraju, A. Sahai, and A. van Moorsel, "Web services management network: an overlay network for federated service management," in *Eighth International Symposium on Integrated Network Management*, March 2003.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8336
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8336