Monitoring SOA Applications with SOOM Tools: A Competitive Analysis

Ivan Zoraja, Goran Trlin, Marko Matijević
Faculty of Electrical Engineering, Mechanical Engineering, and Naval Architecture, University of Split, Split, Croatia

Abstract

Background: Monitoring systems decouple monitoring functionality from application and infrastructure layers and provide a set of tools that can invoke operations on the application to be monitored. **Objectives:** Our monitoring system is a powerful yet agile solution that is able to online observe and manipulate SOA (Service-oriented Architecture) applications. The basic monitoring functionality is implemented via lightweight components inserted into SOA frameworks thereby keeping the monitoring impact minimal. **Methods/Approach:** Our solution is software that hides the complexity of SOA applications being monitored via an architecture where its designated components deal with specific SOA aspects such as distribution and communication. **Results:** We implement an application-level and end-to-end monitoring with the end user experience in focus. Our tools are connected to a single monitoring system which provides consistent operations, resolves concurrent requests, and abstracts away the underlying mechanisms that cater for the SOA paradigm. **Conclusions:** Due to its flexible architecture and design our monitoring tools are capable of monitoring SOA application in Cloud environments without significant modifications. In comparisons with related systems we proved that our agile approaches are the areas where our monitoring system excels.

Keywords: Monitoring, Real-time, Agile, Lightweight, SOA, Cloud, APM, BTM, Security, BI
JEL classification: C61, C63, C8, M15
Paper type: Research article

Received: 26, September, 2012
Revised: 5, March, 2013
Accepted: 6, April, 2013

Citation: Zoraja, I., Trlin, G., Matijević, M. (2013). “Monitoring SOA Applications with SOOM Tools: A Competitive Analysis”, Business Systems Research, Vol. 4, No.1, pp. 21-35.
DOI: 10.2478/bsrj-2013-0003

Acknowledgements: We would like to thank the Croatian Institute of Technology (HIT) for fully funding the SOOM functional prototype for on-premises applications.

Introduction

The market for monitoring systems and tools that observe and modify the run-time behaviour of enterprise applications is evolving rapidly. For example, the Gartner Research (Kowall, Cappelli, 2012) estimates that global spend for APM (Application Performance Monitoring) software licenses including the first-year maintenance contracts will grow by the end of this year to $2.14 billion which is a 9% increase in comparison to 2011. Similar growths are estimated for related monitoring areas such as BTM (Business Transaction Monitoring) and BPM (Business Process Monitoring).

SOA applications provide their business functionality via services. A service is a unit of business functionality exposed to the world via standard descriptions, standard communication protocols, and standard message formats. The clients consume the services by exchanging messages inSimple Object Access Protocol (SOAP)or Representational State

Transfer(REST) architectural styles. A SOA application can be viewed as a graph with services and clients as vertices (nodes) and communication routes as edges. Services are interoperable if they can work together via standards that are not specific to the particular platform, the programming language, or any of the underlying technologies. Modern enterprise applications based on SOA have grown substantially in size, complexity, diversity, and heterogeneity over the past few years.

With the advent of Cloud computing, interactions with distributed applications have become an everyday activity of typical business users, especially of those who work with Customer Relationship Management (CRM) and Enterprise Resource Planning (ERP) systems since such systems are rapidly moving to Cloud environments. Infrastructure as a service (IaaS) providers, i.e. Amazon Web Services, offer the raw computational power, such as virtual machines, to end users. Platform as a Aervice (PaaS) providers, such as Microsoft Azure, abstract away the operating system details from end users and offer essential services such as Web sites and E-mailing. Software as a Service (SaaS) providers add another level of abstraction, offering end users ready to use software solutions deployed in the Cloud. Google Mail and Sales Force are good examples of the SaaS style Cloud solution. However, migrations to the Cloud bring many risks such as potential performance degradations via virtualizations.

The quality of SOA applications deployed on-premises and in the Cloud, as well as the enterprise overall productivity and revenue is directly affected by issues in the applications. Therefore, IT staff and business people need efficient tools (Zoraja, Zulim, Štula, 2008; Hershey, Runyon, 2007) to monitor and manage the run-time behaviour of SOA applications for rapid finding, predicting, and proactively preventing potential causes of issues. Consequently, IT monitoring and management tools have gained in importance over the past few years since they help organizations to deliver services to work flawlessly and optimally at all times for all users. However, current monitoring systems and tools have some shortcomings which prevent them from being efficiently used for monitoring SOA applications deployed on the premises and in the Cloud.

The shortcomings of current monitoring solutions can be categorized as:

- **General Monitoring**

Most of them are not designed and implemented with SOA applications in focus. Instead, they are developed for general-purpose monitoring such as monitoring of virtual machines or network stacks and then enhanced and integrated to be used in the SOA world.

- **Single Monitoring Aspects**

Most current tools are focused on a particular monitoring aspect such as application performance (APM), network performance (NPM), and business transactions (BTM). They are implemented and managed separately and their tools are usually not connected to a common monitoring system and therefore are not aware of each other.

- **Heavyweight**

Most current tools are integrated product suites which are very complex to manage. Their installations and maintenance process is usually a big ceremony requiring not only additional installations of various software components and servers but also expensive consulting assistances.

- **Platform-dependent**

To implement monitoring operations, they typically utilize the low-level mechanisms available in the underlying operating systems, virtual machines, network stacks and packages, databases, and hardware.

- **CloudUnready**

Most of the current tools are not designed with Cloud in mind and therefore cannot be used in Clouds without modifications. For instance, instrumentation mechanisms that are utilized by many current tools can be disabled in Clouds and Cloud providers may not even allow direct accesses to virtual machines due to their security policies. In addition, system metrics that are measured in the guest are generally not trustworthy.

The shortcomings listed above motivated us to design and implement a functional prototype called SOOM (Service-oriented Online Monitor) for monitoring SOA applications. In this paper, we present our solution and compare it with other solutions used in the realm of SOA. SOOM is a real-time (online) monitoring system focused on multiple monitoring aspects.

For instance, SOOM monitors interoperable services, observes IT transactions, identifies performance bottlenecks, and locates and proactively predicts run-time errors in an agile fashion. To minimize the impact on the SOA graph to be monitored, SOOM inserts its lightweight software components into target SOA frameworks using various instrumentation techniques. SOOM supports both SOA architectural styles: SOAP and REST and is Cloud-ready.

The next section discusses both functional and non-functional requirements for SOOM while section three elaborates its software architecture with an accent on software components, their functionality, and interaction patterns. Section four provides an insight into visualization approaches while section five deals with errors and alerts. Section six explains performance monitoring while section seven describes support for business and IT transactions. Section eight illustrates the business intelligence aspects of SOOM. Finally, we finish off with section nine which concludes the paper, discusses our findings and results, and gives suggestions for future work.

Requirements

This section discusses the positioning of SOOM, the operations that it will provide and execute on behalf of its users, and the quality factors it must adhere at.

Positioning

Software developers in SOA environments can generally be classified as infrastructure and application developers. The former deal with development of software frameworks (e.g. SOA) while the latter use that infrastructure to build applications with functionality from a particular business domain. Infrastructure developers often use rudimentary software tools, such as activity tracing and debugging, which are tightly bound to the infrastructure and are, in most cases, inapplicable at the application level. Application developers, on the other hand, are often constrained by financial and timing factors and are devoted to the implementation of the contracted business functionality with practically no time left for the development of advanced monitoring tools.

The main idea behind SOOM is to (1) decouple the monitoring functionality from both the application and the infrastructure by implementing a common monitoring system and (2) to provide a set of tools that can, via that monitoring system, invoke operations on the application to be monitored. The capabilities and conditions to which SOOM must conform are driven by the perceived needs of the main SOA actors: operators, business managers, programmers, and testers. The SOOM objects to be monitored include clients, services, operations, parameters, return values, messages, contracts, endpoints, bindings, addresses, communication paths, and software infrastructure mechanisms such as security, transactions, and reliable messaging.

Functional Requirements

The functionality of SOOM can be viewed as a collection of operations it performs on the objects to be monitored where each operation yields an observable result of value to an actor. The SOOM monitoring operations can be invoked unconditionally and conditionally. Unconditional requests are immediately executed after they have been received while the conditional ones wait for events to happen. An example of an unconditional operation is a request for measuring performance of a particular service. Alerting on a security violation is an example of a conditional request. Applying various synchronization techniques SOOM is able to resolve conflicts when operations are executed concurrently.

SOOM is a complete system since it implements operations for all the tools and observes and manipulates all elements of the SOA graph. All monitoring operations are designed to be executed without races, deadlocks, and falsifications. Our current monitoring system (Version 1.0) provides the full support for WCF (Windows Communication Foundation) (Lowy 2010) and the basic support for Java WS services. SOOM is ready for SOA applications deployed in Cloud. We have also implemented some deep-diving techniques to inspect infrastructure such as operating systems and virtual machines but these techniques are out of scope of this paper.

MainFeatures

Nonfunctional requirements (NFRs) specify the quality attributes of the system. The design and architecture of SOOM are centred around the following main features:

- **Real-time and Lightweight**

All SOOM operations are executed while SOA applications being monitored are running and all collected data reflect the runtime behaviour and the execution context. To achieve this SOOM inserts own lightweight components into SOA frameworks (WCF, Java WS) using various inspection mechanisms. The monitoring impact (intrusiveness) on the application being monitored should be kept minimal. SOOM is an easy to manage solution.

- **Hiding complexity of SOA Applications**

SOA applications are very complex in nature since they are composed of heterogeneous software components connected through a variety of transport protocols which can reside at various Cloud and on-premises environments. SOOM hides the complexity of SOA applications implementing an architecture where its designated components deal with specific application aspects and provide end-to-end monitoring. For example, the SOOM server deals with distribution and parallelism providing operations on services as if they were deployed locally.

- **Application-level and Agile**

Monitoring aspects are integrated in an agile fashion, providing data that can be related to constructs that the end user understands. SOOM performs application-level monitoring but it is independent of application domains (domain agnostic) and can monitor applications in any business area. In contrast, component based monitoring reports averages of individual components (e.g. database) and low-level monitoring (e.g. network) collects data that cannot be easily related to the application constructs.

- **Consistency and Transparency**

All SOOM tools are connected to a single monitoring system which provides consistent operations to all the tools. This means that tools are aware of each other via the monitoring system (Zoraja, 2000). When one tool performs an operation on a service all interested tools are notified about that event. The underling SOA mechanisms using the monitored objects are transparently handled and SOOM tools can be built without taking care of a particular implementation details of the underlying SOA framework.

- **Cloud Ready**

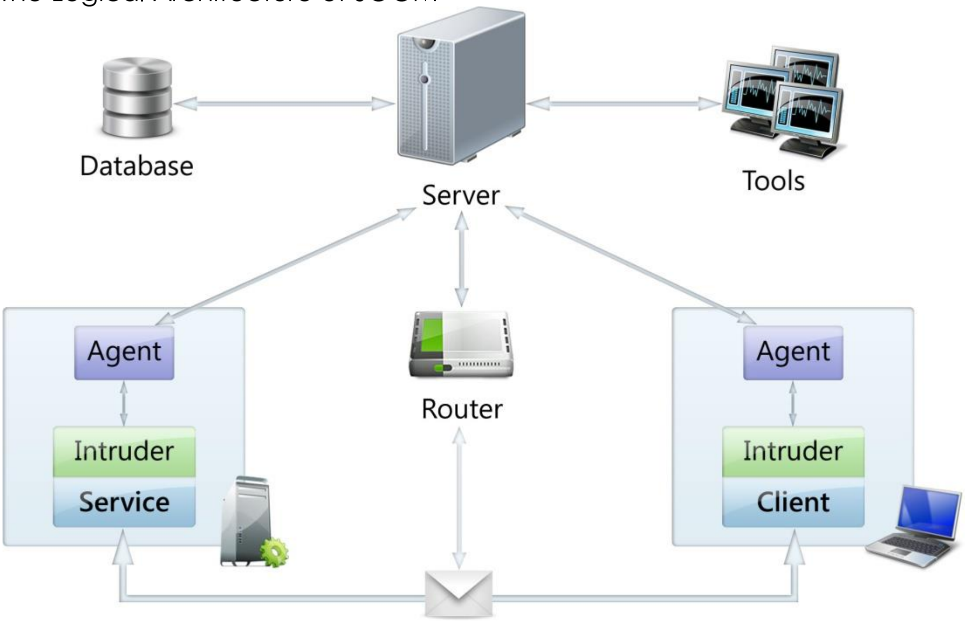
Due to its flexible and application centric design and architecture where all instrumentations and measurements are implemented at the application level, SOOM is fully capable of monitoring SOA application in Cloud, without modifications. SOOM is successfully tested in various Cloud environments, such as Amazon Web Services (IaaS) and Windows Azure (PaaS).

Architecture and Workflows

A real-time monitoring system can be implemented at various software and hardware levels. Only hardware monitors are not intrusive. Software monitors (Zoraja, Zulim, Štula, 2008) are more flexible for modifications but have an impact on the system to be monitored. Monitoring systems integrate the common functionality used by multiple tools in a reusable manner, hide platform idiosyncrasies from tools, and preserve the consistency of monitoring operation issued by concurrent tools. The software architecture of SOOM is based on the CORAL system (Zoraja, 2000). It specifies key decisions mostly regarding decompositions into components and integrations via connectors thereby applying various designs and architectural patterns.

In order to provide contracts to the tools, reduce the system complexity as well to ensure reusability, scalability, programmability and maintainability, the logical architecture of SOOM is based on SOA, the event-action paradigm, and layering. SOOM is designed and implemented having in mind the efficiency of the monitoring system and the impact of monitoring on target business applications. SOOM is designed to be easily enhanced by new functionality and features.

Figure 1
The Logical Architecture of SOOM



Source: Author's illustration

SOA systems make use of the proxy pattern to provide the same programming model to the clients regardless of the service location. The proxy is also used to hide the complexity of the underlying transport protocols and message formats from the client. On the service side, SOA frameworks implement dispatchers which, on behalf of the proxies, perform operations on services. With reference to Figure 1, SOOM intercepts proxies and dispatchers and injects monitoring functionality into the client and service code. SOOM agents control multiple intruders and send monitoring data to the server which provides monitoring operations to versatile tools.

Components and Connectors

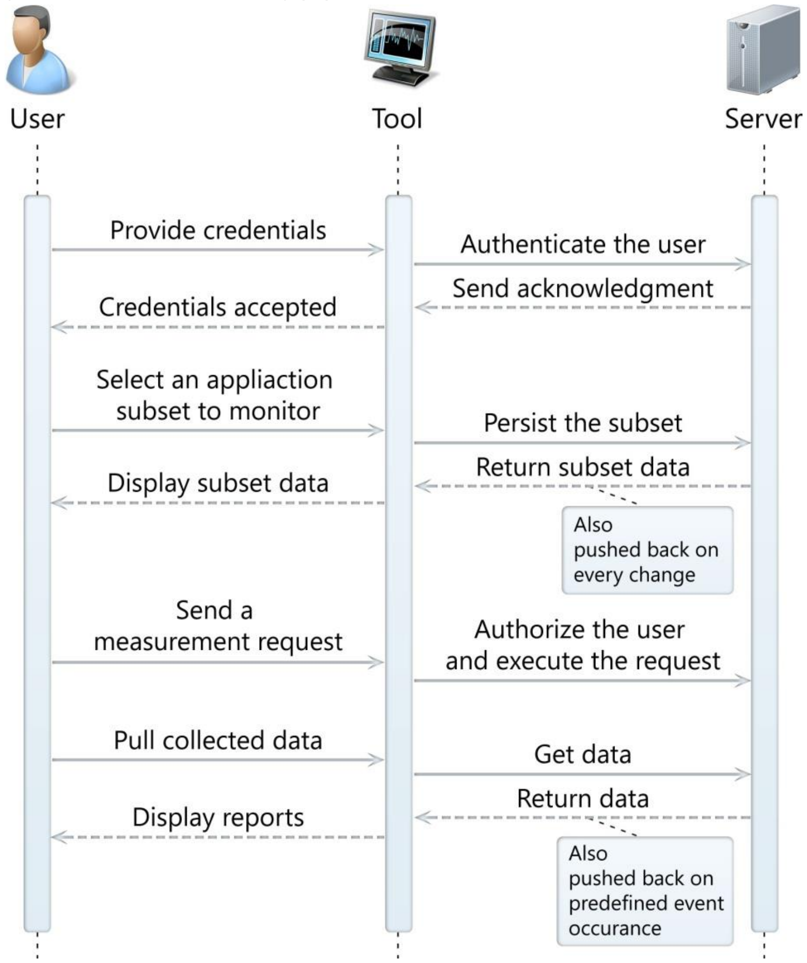
SOOM intruders are inserted into the selected SOA framework in an aspect fashion without the necessity to modify the source code of the application to be monitored. They collect useful activity information and appropriate interaction data records from various client/service interactions and return data to the corresponding agent. Intruders can also modify objects being monitored such as parameters passed by the client and return values. Most SOA frameworks provide extension points that serve for extensibility purposes. SOOM intruders for WCF make use of the predefined extension interfaces exposed by WCF proxies and dispatchers. Java WS intruders use dynamic proxies and can be inserted into Java SOA applications without restarting the services being monitored.

SOA applications can be deployed in public Cloud, private Cloud, or on the premises. This heterogeneity adds another level of complexity to SOA monitoring, especially on interception techniques and intruder actions. Most current approaches rely on a virtual machine or operating system instrumentations. For example Microsoft SCOM (Price, Mueller, Fenstermacher, 2007) employs Windows performance counters for gathering system state information. This type of instrumentation can be disabled in Clouds and Cloud providers may not even allow direct accesses to virtual machines due to their security policies. SOOM deals with this issues using its highly flexible and cloud-ready architecture. SOOM intruders act as integral parts of the monitored application since they are deployed along with the application to be monitored.

SOOM agents are representatives of the SOOM server on local machines. They control multiple running clients and services. Agents forward requests issued by the server to multiple intruders, handle their responses, and return data back to the server. An agent can also control intruders on remote machines. The communication between SOOM components is implemented via asynchronous messaging.

The SOOM server is a single process responsible for distribution and parallelism since it splits requests from the tools and sends them to agents for further processing. Other main tasks of the SOOM server include enforcing consistency of issues requests, binding events to actions, gathering results from agents, and sending replies to the tools. The main monitoring loop waits for requests that can come up from two sources: tools and agents. The SOOM server maintains a database of all data collected from agents and intruders. For agile SOA scenarios which are based on the REST architectural style we integrate the agent functionality with the server code and therefore reduce the number of SOOM components to be deployed in production environments.

Figure 2
User interactions with SOOM via tools

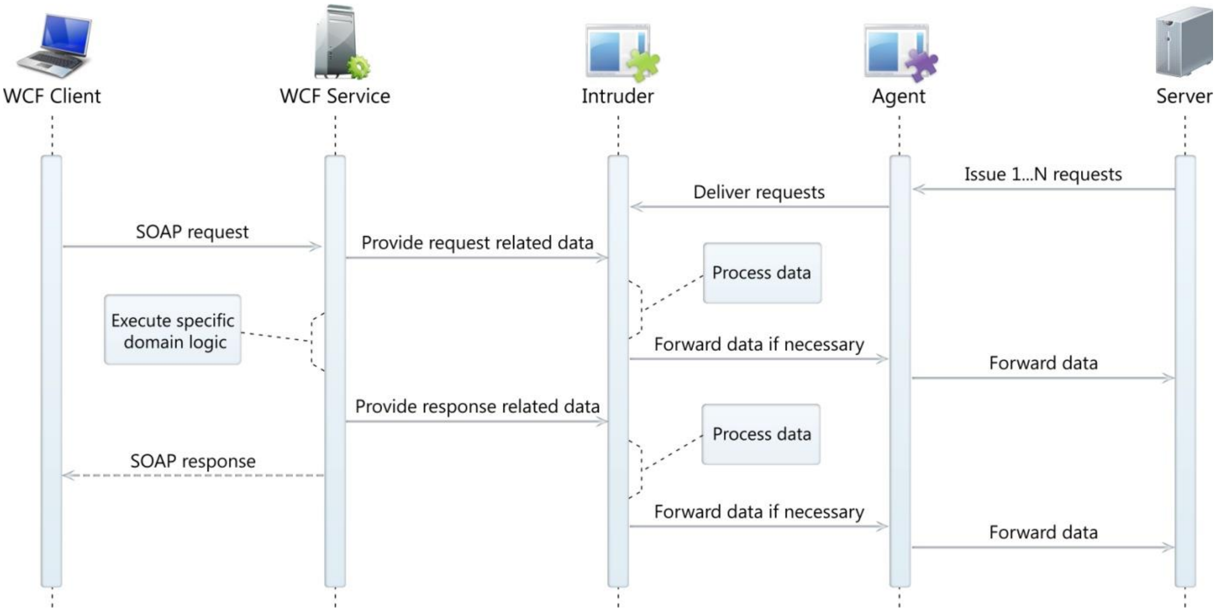


Source: Author's illustration

The SOOM router redirects SOAP messages depending on a set of predefined rules. That way, it can perform load balancing tasks in systems under heavy load and therefore help achieve stability and scalability (Macias, Sanchez, Suarez, Sunderam, 2009) in such systems. The criteria for balancing can be either performance data or the semantic of the service call being invoked.

SOOM users do not directly interact with the SOOM server. Instead, they use GUI components called tools to issue requests and presently received data in meaningful and intuitive ways. Tools are classified according monitoring aspects such as performance and errors, and are placed in the SOOM Dashboard. A SOOM tool forwards user requests to the SOOM server and shows the results obtained from the SOOM server. Current tools are implemented using the WPF (Windows Presentation Foundation) technology. A typical user interaction with the SOOM system is depicted in Figure 2. In this scenario, s/he must provide credentials to be authenticated by the server. S/he can then select SOA applications to be monitored and issue conditional and/or unconditional requests using GUI components from the particular tool.

Figure 3
Exchanging System Messages



Source: Author's illustration

Internal Messages

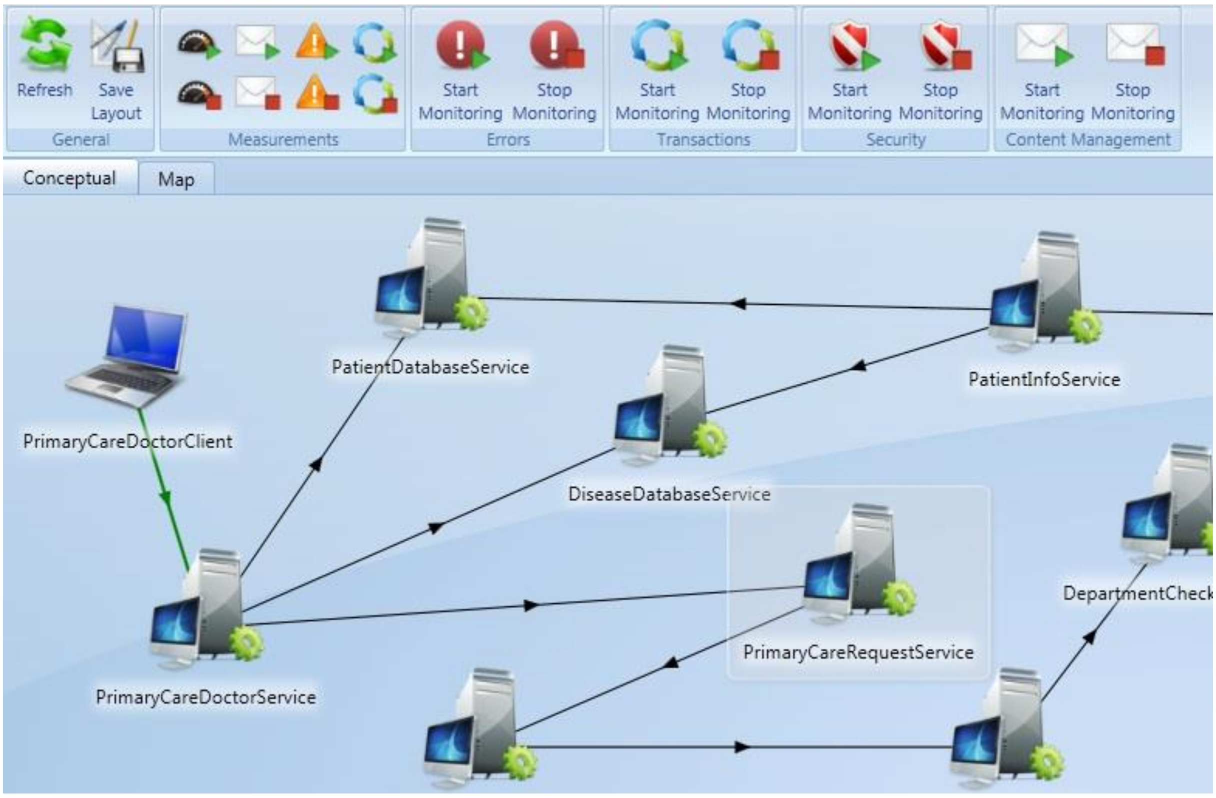
SOA applications utilize messages (SOAP and REST) for communication among different nodes in the system. SOOM is able to intercept and monitor each message sent to or received from a node in the SOA graph being monitored. A typical system workflow describing the interception and handling of SOAP messages is depicted in Figure 3. After the SOOM intruder detects and extracts the SOAP message, message details are sent to the SOOM agent responsible for data aggregation from one or multiple intruders. The final step in this system workflow is data transfer from the SOOM agent to the server.

The SOOM Dashboard

SOA applications often contain dozens or even thousands of nodes and, in order to gain complete insight into their static and dynamic behaviour, all nodes must be monitored. Monitoring such a large set of nodes via textual representations could be a very complex and cumbersome task. For this reason, modern tools use graphical components to represent SOA graphs and provide graphical animations for executions of SOA chains at an abstract level which can be used to categorize current visualization tools in the realm of SOA as (1) application-level and (2) low-level tools. The former makes use of terms and objects found in the business domain while the latter utilizes terms and notions from the underlying technologies such as network stacks, operating systems, and physical hardware components. The main advantage of the tools in the first category is delivering information that can be related to the application constructs such as operations, messages, and parameters and therefore those tools are best suited for application developers and operators. In addition to SOOM, to application-level tools belong tools such as dynaTrace (Ballou, 2008) and AVICode.NET which is now part of SCOM (Price at al.). Low-level tools provide information that can be useful for systems administrators and cannot be easily related to application constructs. Some vendors provide custom visualization plug-ins for their products to be used with low-level tools. For low-level tools, for instance, belong Quest Foglight Network

Management Systems for Virtualized Environments (Quest Software, 2011) and BMC Atrium Discovery and Dependency Mapping (BMC Software, 2009).

Figure 4
The SOOM Dashboard



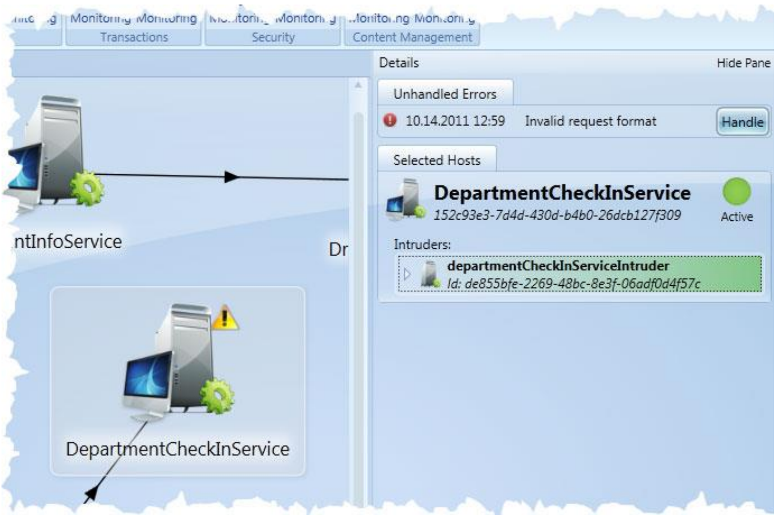
Source: Author's software

With reference to Figure 4, SOOM can discover the services involved in the execution of the SOA application and the communication paths across which those services exchange messages, and show them in the SOOM Dashboard. It represents an integral "dash-board" for all types of SOA monitoring in a form of a directed graph and provides both intuitive and accurate graphical representation of the SOA graph to be monitored, its healthy state, and behaviour. All nodes are represented using their metadata, such as node name, node type, and node identifier. This metadata information is then used to render appropriate icons for different type of nodes, meaning that client nodes can easily be distinguished from service ones. The SOOM Dashboard provides direct lines to visually connect nodes. The lines change colours depending on the state of the connection.

To get a better insight into monitoring of a large number of nodes the dashboard provides controls to zoom in, zoom out, and to save the current configuration. The details of the selected client, service, and connection channel can be shown in the Details Pane (not shown in Figure 4). Via that pane the user can preview the service's metadata by clicking the corresponding button. The Metadata Viewer is a tool that visualizes the metadata (WSDL) using metadata exchange mechanisms available in the underlying SOA infrastructure. Metadata is a platform-independent description of a service including ABC (addresses, bindings, contracts) and information about security, transactions, reliability, and faults.

In addition to the conceptual view, the SOOM Dashboard provides a map view which shows SOA components on their geographical locations. This type of view is a unique feature of SOOM. SOOM also implements a version of the SOOM Dashboard that can be added to the MS Visual Studio 2010 as a plug-in to support the process of testing WCF services.

Figure 5
Detecting Errors



10.14.2011 12:59 Invalid request format

DepartmentCheckInService
152c93e3-7d4d-430d-b4b0-26dcb127f309

departmentCheckInServiceIntruder
Id: de855bfe-2269-48bc-8e3f-06adf0d4f57c

Source: Author's software

Errors and Alerts

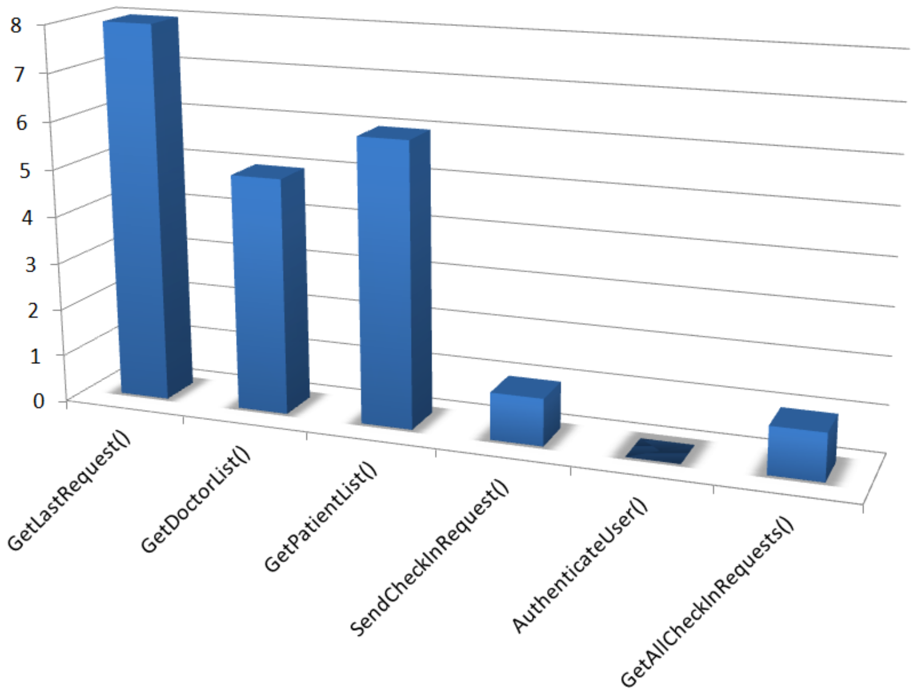
A large variety of runtime errors can arise in SOA applications. For example, connection failure between two nodes may jeopardize data transfer. Error detection and reporting are often referred to as deep diving, since they provide the programmers with low level details about the application misbehaviour. Traditional monitoring systems implement post-mortem error analysis using a collection of application log data long after the errors have happened. Examples of this type of monitoring include various tracing tools, such as Service Trace Viewer from Microsoft (Lowy, 2010). This type of error analysis, although useful in local environments, is not suitable for distributed application since it could be very difficult to correlate errors that span multiple distributed nodes using row tracing data. Tools that integrate error and performance monitoring are discussed in section six.

Runtime exceptions and security violations are the two basic types of errors in SOA applications to be dealt with. SOOM has successfully dealt with both types of errors by detecting errors at run-time at the exact location of the issue using inspection mechanisms available in the underlying SOA infrastructure. As soon as a problem has arisen, SOOM catches the exception from the SOA infrastructure, marks the faulting node problematic, and stores the complete information about the problem in the SOOM database at the SOOM server.

With reference to Figure 5, the erroneous node is marked by the exclamation point in the SOOM Dashboard. Using the Details Pane, the user can discover the root cause by tracing an error back to the issuing client through all intermediary services if exist. By analysing the entire call tree invocations the user can deduce how the particular error was propagated and what was the real cause of the problem. SOOM can also proactively detect services that are not working. When an error occurs or when the system detects the faulting node it can automatically notify the interested parties via E-Mail or SMS messages.

False credentials, replay attacks, eavesdropping, and etc. are common security threats in SOA applications. Some vendors focus their security related efforts on securing network resources and devices applying the process of continuous scanning the whole network for security vulnerabilities. Tools like ManageEngine's Security Manager Plus (ManageEngine, 2009) are able to monitor network routers, computers, and firewalls in order to mitigate security threats. SOOM utilizes the security and auditing mechanisms available in the underlying SOA infrastructure (WCF and Java WS) to gather security related data. Security threats are identified in real-time and acted upon immediately via the SOOM alerting mechanisms.

Figure 6
Presenting the Number of Calls per Operation



Source: Author's software

Performance Monitoring

Functional requirements for an application can be implemented on various architectures. It is the task of the architect to set an appropriate architecture (software and hardware) on which the application will work flawlessly and efficiently for all users at all times. To determine the run-time behaviour of the software and to verify the architecture, the operators use various tools to measure (Milanović Glavan, 2011), analyse (Wismüller, Bubak, Funika, 2008), and visualizes (Bode, 1994) the application performance and availability.

SOOM is focused on SOA. Business SOA applications require agreements and expectations at service levels (SLAs, SLEs) that reflect performance goals for individual and composite services. The SOOM intruders implement a large variety of light (minimal intrusive) and precise sensors to measure SOA performance at the application level while SOOM tools provide graphical components to show performance from the perspective of end users focusing on End User Experience (EUE). In Figure 6 we present a SOOM GUI component that shows the number of calls (frequency) for each individual operation of the service being monitored. For example, operation GetDoctorList has been invoked five times since the performance monitoring process started. Other SOOM GUI components present measured and aggregated data for operations in a single service or for operations invoked in a chain of services where services delegates tasks to other services. The services to be monitored by SOOM can be located on the premises and in the private and public Clouds (Trlin, Zoraja, 2012).

To perform application performance in real time, SOOM injects tokens into SOA messages and makes use of a modified version of the Lamport timestamp algorithm for global clock synchronization (Lamport, 1978). This algorithm is used to establish temporal order between operations invoked in distributed services. After the temporal order has been established, SOOM employs local time differences to measure time slices spent at each individual operation. This technique enables end-to-end performance monitoring which can be used to easily detect and pinpoint bottlenecks in the SOA application being monitored.

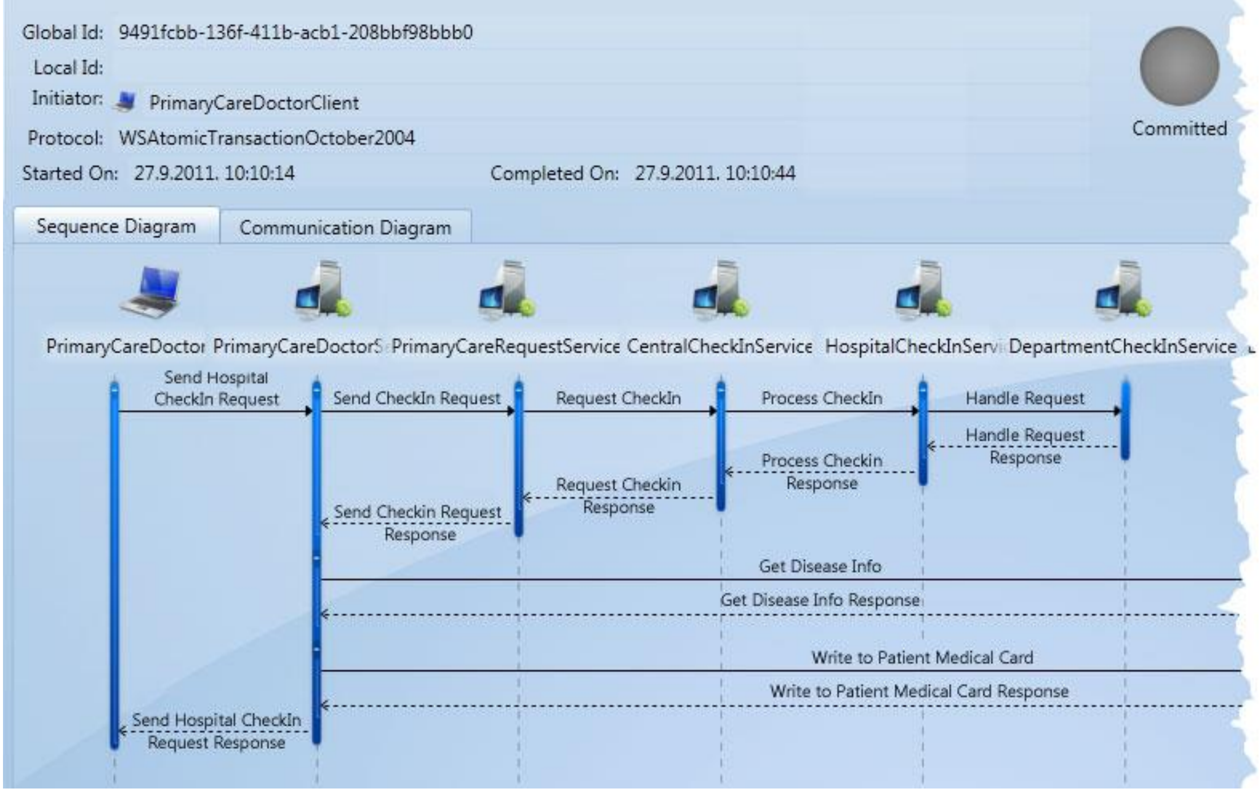
Transaction Monitoring

An SOA IT transaction represents a group of operations invoked in a chain of distributed services, in which all the operations must succeed or fail as a group. It exhibits well-known ACID properties and can end in two ways: with a commit or a rollback. When a transaction commits, the modifications made by its operations are performed. If only one operation within a transaction fails, the transaction must be rolled back, undoing the effects of all operations in the transaction. In addition to IT (technological) transactions some tools (AppDynamics, 2011) monitor business transactions which also represent a group of operations in a set of distributed services but do not exhibit ACID properties.

Most of the current monitoring tools focus on the custom business transactions that are tracked using "dope-and-trace" approaches which insert custom stamps into communication protocols such as HTTP. For example, this approach is utilized by the IBM's Tivoli (IBM 2012) family of tools. There is also an evident intention of combining APM and BPM tools. A representative example of this approach is found in Responsive Process Management Suite, a monitoring software developed by Progress Software (Hailstone, 2010). APM in conjunction with BPM brings focus to the business side of software, therefore making APM software more useful for typical business users.

Performance tools utilize various techniques and mechanisms to measure times (latencies) and frequencies (throughput) of individual and composite (aggregate) components. The measurement techniques should be precise and have minimal impact (overhead) on the application being measured. Visualization techniques use the collected and calculated data and present application performance to the users for analysis (Forrester Consulting, 2010) (CA Technologies, 2012). Most performance tools are targeted towards specific technologies and platforms such as networking (Quest Software, 2011) and database-centric environments (Oracle, 2011).

Figure 7
An Operation Chain in a Transaction



Global Id: 9491fcbb-136f-411b-acb1-208bbf98bbb0
Local Id:
Initiator: PrimaryCareDoctorClient
Protocol: WSAtomicTransactionOctober2004
Started On: 27.9.2011. 10:10:14
Completed On: 27.9.2011. 10:10:44
Committed

Source: Author's software

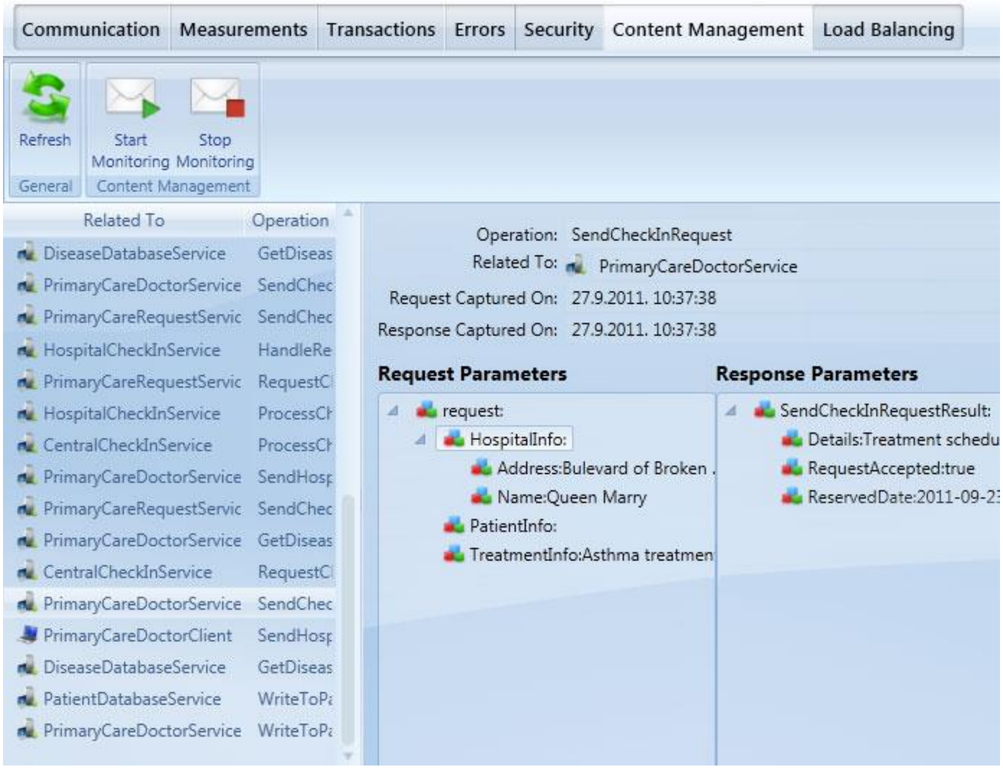
SOOM implements graphical components to visualize WCF IT transactions. All started transactions are listed in the Transaction Navigation Pane while the details of an individual transaction are shown in the Details Pane that provides two diagrams in the UML style: the sequence and communication diagrams. A transaction listed in the Transaction Pane can be in four states: active, committed, aborted, and in-doubt. The active transactions are in progress, the committed transactions have succeeded, the aborted transactions have failed, and the in-doubt transactions are not resolved yet. The in-doubt transactions can happen when a component of the distributed transaction infrastructure including multiple coordinators like MS DTC (Microsoft Distributed Transaction Coordinator) and resource managers fails.

Figure 7 depicts a committed transaction (denoted with the grey circle) in the SOOM Transaction Sequence Diagram. It shows a series of operations invoked on a set of WCF services with an emphasis on the chronological order of invocations. Each transaction has assigned the global and local identifiers, the client node that initiated the transaction, the start and completion times, and the transaction protocol used to exchange messages of the two-phase commit protocol (2PC). The Communication Diagram shows the same information but multiple operations invoked on the same service are depicted using the same communication link along with sequence numbers to denote the chronological order of events. To reconstruct transactions, SOOM makes use of identifiers generated by the underlying transaction infrastructure combined with timestamps inserted into SOA messages to determine the temporal order of events.

Content Management

Modern business applications usually deal with the huge amounts of raw business data and software solutions that collect (Pejić Bach, Šarlija and Jaković, 2009), store, and analyse (Bosilj Vukšić, Indihar Štemberger, Kovačič, 2008; Hernaus, Pejić Bach and Bosilj Vukšić, 2012) information from business applications have gained substantial importance, especially because such software can be used to improve business processes.

Figure 8
Request and Response Parameters



Related To	Operation
DiseaseDatabaseService	GetDiseas
PrimaryCareDoctorService	SendChec
PrimaryCareRequestServic	SendChec
HospitalCheckInService	HandleRe
PrimaryCareRequestServic	RequestCl
HospitalCheckInService	ProcessCh
CentralCheckInService	ProcessCh
PrimaryCareDoctorService	SendHosp
PrimaryCareRequestServic	SendChec
PrimaryCareDoctorService	GetDiseas
CentralCheckInService	RequestCl
PrimaryCareDoctorService	SendChec
PrimaryCareDoctorClient	SendHosp
DiseaseDatabaseService	GetDiseas
PatientDatabaseService	WriteToPa
PrimaryCareDoctorService	WriteToPa

Operation: SendCheckInRequest
Related To: PrimaryCareDoctorService
Request Captured On: 27.9.2011. 10:37:38
Response Captured On: 27.9.2011. 10:37:38

Request Parameters

- request:
  - HospitalInfo:
    - Address:Bulevard of Broken .
    - Name:Queen Marry
  - PatientInfo:
  - TreatmentInfo:Asthma treatmen

Response Parameters

- SendCheckInRequestResult:
  - Details:Treatment schedu
  - RequestAccepted:true
  - ReservedDate:2011-09-2

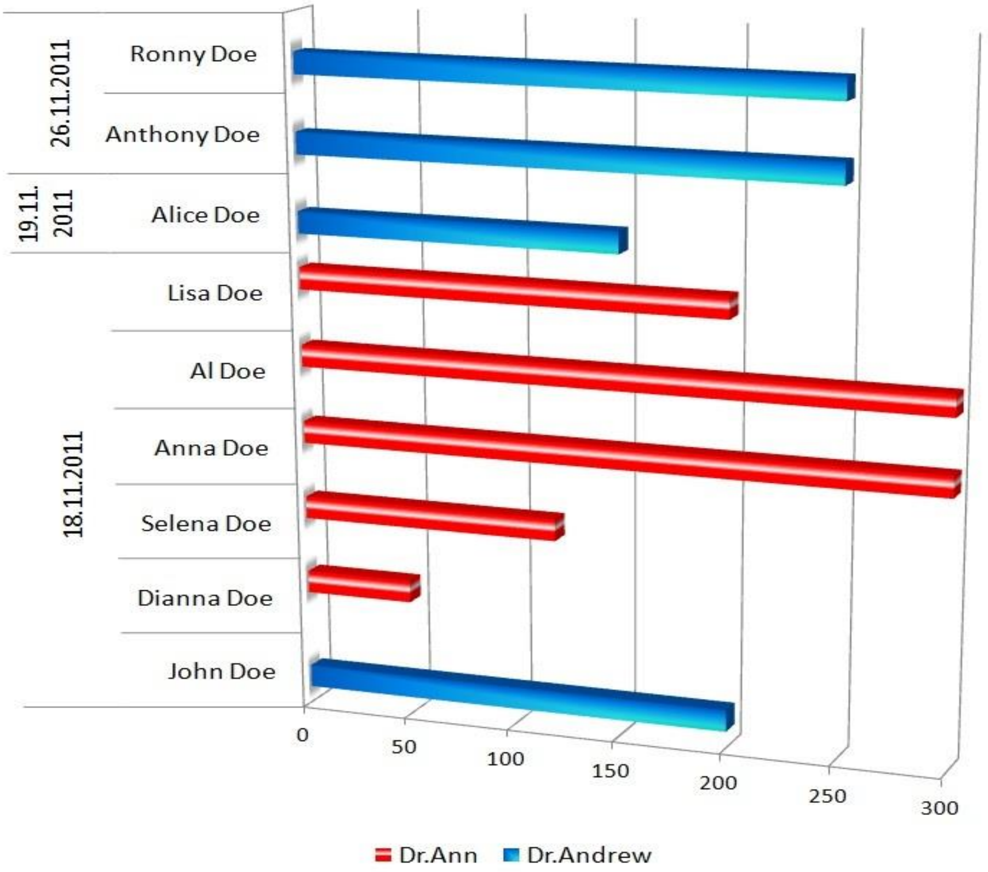
Source: Author's software

Business information in a SOA application is transferred from service to service in SOA messages. SOOM is able to intercept SOA calls, extract business information from request and response messages, and store content data into the SOOM Database or CSV (Comma Separated Value) files. SOOM Intruders are components that gather and store content data. The data can then be analysed using the SOOM Content Viewer, as depicted in Figure 8, or processed as an OLAP Cube using a data mining tool such as Microsoft Excel, as shown in Figure 9, to perform multidimensional database analysis.

With reference to Figure 8, SOOM acts as a tool for gathering, storing, and viewing the content data in SOA applications via the Navigation and Details Pane. Since each service call is recorded (persisted), SOOM can present both request parameters and response values to the user. Complex data types which are passed and returned are displayed in a hierarchical fashion using the name/value semantics. By browsing through the Navigation Pane it is possible to view details about each recorded call in the Details Pane and explore request and response values.

With reference to Figure 9, content data collected by SOOM can be analysed using MS Excel. For example, on the x-axis we have expenses and on the z-axis we have patients. The added dimension that creates a multidimensional table is a doctor, a patient is associated with.

Figure 9
Analytical Data



Source: Author's software

Conclusion and Future Work

Strong trends in IT industry toward software architectures based on SOA for both on-premises and Cloud enterprise applications have highlighted the need for powerful monitoring systems and tools that can support those applications in both development and production scenarios. In this paper, we have discussed the core issues and solutions that come up in the architecture, design, implementation, and usage of an agile, lightweight, cloud-ready, and real-time monitoring solution for SOA applications. We presented the main features of our monitoring system called SOOM and its tools, compared them with other solutions in the realm of SOA monitoring, and proved that our agile monitoring approaches are the areas where SOOM exhibits many advantages over related systems.

SOOM is all about the non-functional requirements (quality factors) for SOA applications. It is able to visualize the complete SOA graph including clients, services, and communication channels. SOOM detects and locates root causes of issues including runtime errors and security violations, measures and analyses end-to-end performance, visualizes IT transactions, and collects and analyses content data. These features make SOOM an effective and innovative monitoring software due to its ability to significantly reduce the problems associated with the development and usage of enterprise SOA applications.

Since SOOM has proven to be a very successful and beneficial asset for any IT based or IT dependent companies we are encouraged and strongly motivated to continue developing SOOM along several lines. One of the first enhancements to be undertaken is to investigate the usage of SOOM to support other programming languages such as Java, Python, and Ruby and to work on other platforms such as Linux and Android. We will further develop SOOM tools based on Web technologies that support rich GUIs since this will lead to a more flexible solution for monitoring SOA applications deployed in the Clouds. A third modification involves a full implementation for the AWS and Windows Azure Cloud platforms. Finally, some long term works will be undertaken to implement (1) routing strategies with various load balancing techniques and (2) methods for analysing and presenting collected business data.

References

1. AppDynamics (2011), "The Power of the Business Transaction: The New Way to Manage Application Performance", Business Whitepaper, available at http://www.appdynamics.com/(October 2011).
2. Ballou M. (2008), "Leveraging Performance Optimization for Business Advantage", Whitepaper, sponsored by dynaTrace software Inc., available at: http://www.compuware.com/(May 2011).
3. BMC Software (2009), "BMC Atrium Discovery and Dependency Mapping", available at http://www.bmc.com/(Jun 2011).
4. Bode, A. (1994), "Parallel Program Performance Analysis and Visualization," Proceedings of Second Workshop on Environments And Tools For Parallel Scientific Computing, pp. 246-253. Townsend, Tenessee, May, 25-27 , 1994.
5. Bosilj Vukšić, V., Indihar Štemberger, M., Kovačič A. (2008), "Business Process Management and Business Intelligence as Performance Measurement Drivers", The Business Review, Cambridge, Vol. 10, No. 1, pp. 338-343.
6. CA Technologies (2012), "CA Application Performance Management", Solution Brief, available at: http://www.ca.com/us/default.aspx /(September 2012).
7. Forrester Consulting (2010), "Managing Performance of Critical Applications", White Paper, available at: http://www.opnet.com/whitepapers/index_apm.html /(June 2012).
8. Hailstone R. (2010), "Progress Responsive Process Management", Progress Software, Technology audit, available at: http://www.progress.com/en/business-need/responsive-process-management.html / (June 2011).
9. Hernaus, T., Pejić Bach, M., Bosilj Vukšić, V., (2012), "Influence of strategic approach to BPM on financial and non-financial performance", Baltic Journal of Management, Vol. 7 No. 4; 376-396.
10. Hershey, P., Runyon, D. (2007), "SOA Monitoring for Enterprise Computing Systems", 11th IEEE International Enterprise Distributed Object Computing Conference, pp. 443-450. Annapolis, Maryland.
11. IBM (2012), "IBM Tivoli Composite Application Manager for Transactions - Guide to agentless transaction tracking", Datasheet, available at: http://www-01.ibm.com/software/tivoli/products/composite-application-mgr-transactions/(August 2012).
12. Kowall, J., Cappelli, W. (2012), "Magic Quadrant for Application Performance Monitoring", available at http://www.gartner.com/DisplayDocument?ref=clientFriendlyUrl&id=2125315
13. Lamport L. (1978), "Time, Clocks, and the Ordering of Events in a Distributed System", Communications of the ACM 21, Vol. 21, No., pp. 558-565.

14. Lowy L. (2010), "Programming WCF Services: Building SOAs with Windows Communication Foundation, 3rd Edition", O'Reilly Media, Sebastopol CA.
15. Macias, E. M., Sanchez, D., Suarez, A. Sunderam, V. (2009), "Optimization of Execution Time Inspired Cross Layer Design Using Effective Load Balancing in a LAN-WLAN Environment", International Journal of Computational Science and Engineering, Vol.4, No. 3, pp. 182-194.
16. ManageEngine (2009), "Security Manager Plus - User Guide", Whitepaper, availabale at: http://www.manageengine.com/products/security-manager/security-management-help.pdf.
17. Milanović Glavan, Lj. (2011), "Understanding Process Performance Measurement Systems", Business Systems Research, Vol. 2, No. 2, pp. 25-38.
18. Oracle (2011), "Introducing Oracle Enterprise Manager 12c ", Oracle Press Release, available at: http://www.oracle.com/us/products/enterprise-manager/index.html / (February 2012).
19. Pejić Bach M., Šarlija N., Jaković B. (2009), "Business Intelligence and Risk Management: Case Study of Croatian Banking Sector", International Journal of Data Analysis and Information Systems, Vol. 1, pp. 1-9.
20. Price B., Mueller, J. P., Fenstermacher, S. (2007), "Mastering System Center Operations Manager 2007", Wiley Publishing, Indianapolis. Indiana.
21. Quest Software (2011), "Simplified Network Management For Virtualization Administrators", available at: http://www.quest.com/ (February 2012).
22. Trlin, G., Zoraja, I. (2012), "Online SOA Chain Performance Monitoring In Mixed Environments", The 20th International Conference on Software, Telecommunications and Computer Networks, Split, 11-13 Sept. 2012. pp.1-5.
23. Wismüller R., Bubak, M., Funika, W. (2008), "High-Level Application Specific Performance Analysis using the G-PM Tool", Future Generation Computer Systems, Vol. 24, No. 2, pp.121-132.
24. Zoraja I., (2000), "Online Monitoring in Software DSM Systems", Shaker Verlag. Aeachen.
25. Zoraja I., Zulim I., Štula, M. (2008), "CORAL - Online Monitoring in Distributed Applications: Issues and Solutions", WSEAS Transactions on Computers, Vol. 7, pp. 113-118.

About the authors

Ivan Zoraja is a professor of computer science at the University of Split, Croatia and the director of company Zoraja Consulting. He received his PhD degree in computer science from the Technical University of Munich in collaboration with the Emory University in Atlanta. His primary research interests are software engineering, parallel computing, distributed systems, and 3D algorithms and simulations in medicine. Author can be contacted at ***zoraja@fesb.hr***

Goran Trlin is a PhD student of computer science at the University of Split, Croatia. He received his master degree in computer engineering from the University of Split, Croatia. His primary research interests are software engineering, parallel computing, distributed systems, and business intelligence systems. Author can be contacted at ***Goran.Trlin@fesb.hr***

Marko Matijević is a PhD student of computer science at the University of Split, Croatia. He received his master degree in computer engineering from the University of Split, Croatia. His primary areas of interest are software engineering, distributed systems and user interface design. Author can be contacted at ***Marko.Matijevic@fesb.hr***