Usage Tracking Components for Service-Oriented Middleware Systems

Daniel Skrobo, Klemo Vladimir, and Sinisa Srbljic
School of Electrical Engineering and Computing, University of Zagreb, Croatia
{daniel.skrobo, klemo.vladimir, sinisa.srbljic}@fer.hr

Abstract

Collecting data on user activities is one of the fundamental middleware services in web-enabled systems. The collected data is analyzed and used by various high-level services, like user profiling, accounting, security auditing, and system health monitoring. In this paper, we present architecture and performance evaluation of usage tracking components for service-oriented middleware systems. Presented middleware components are designed as loosely-coupled usage tracking services, which brings two important benefits. Usage tracking services can be seamlessly integrated with various service-oriented systems without disturbing their operation. Since usage tracking services are loosely-coupled, system users can dynamically deploy and manage multiple usage tracking configurations.

1. Introduction

Service-Oriented Computing (SOC) [1], supported by technologies based on WS-* standard set [2], promises to transform Internet into a flexible and dynamic computing environment. In this new service-oriented environment, distributed applications are developed by integrating autonomous and loosely-coupled services using service composition middleware systems. Typical service composition systems provide service deployment, execution, and coordination facilities. These facilities enable rapid development of business applications using EAI (Enterprise Application Integration) concepts [3], and enforcement of B2B (Business to Business) interactions in open service marketplaces [4]. Successful adoption of SOC-enabled technologies in middleware systems brought new research challenges, like service discovery, selection, coordination, security, and usage tracking [5].

Usage tracking middleware components enable monitoring of user activities in service-oriented systems and form the basis for high-level services, such as user profiling, accounting, security auditing, and system health monitoring. However, enforcement of usage tracking is a complex task, due to dynamic nature of service interaction patterns and distribution of services across organizational boundaries. Thus, design of middleware components for effective usage tracking must address several important issues, like distributed data harvesting, storage space management, data aggregation, usage tracking configuration management, and service scalability.

In this paper, we present architecture and performance evaluation of middleware components for usage tracking in service-oriented environments. Usage tracking components are designed to meet the following two challenges: adaptability to various distributed environments and simplified configuration management. In order to make them adaptable to various distributed environments, the presented middleware components for usage tracking are designed as loosely-coupled services. Usage tracking services are deployed on nodes of host service-oriented system where they record invocation events for application services and collaboratively manage the recorded usage tracking data. Since usage tracking services are loosely-coupled, system users can dynamically change usage tracking configurations without disturbing host system’s overall operation. Usage tracking configurations are specified using simple XML-based documents. Furthermore, users can access usage tracking data by issuing queries written in a simple XML-based query language.

Section 2 presents related work on usage tracking systems. Section 3 presents a middleware system used for integration and testing of the developed usage tracking services. Architecture of usage tracking services is presented in Section 4, while usage records management protocols are described in Section 5. Section 6 presents languages for describing usage tracking configurations and querying the system for usage records. Section 7 presents system performance tests and results. Section 8 concludes the paper and outlines future work.

2. Related Work

Web usage monitoring has been originally enforced at Web servers for storing client’s HTTP protocol-level interactions with the system. However, web server logs
are maintained as flat textual files describing low-level events inappropriate for most application-specific purposes. Thus, research has been focused on specialized solutions for usage tracking in Grid and cluster systems, e-business information systems, and open service-oriented systems.

Ganglia [6] and MonALISA [7] are the most prominent monitoring systems for high performance computing environments [8]. With Ganglia monitoring system, clusters across the Internet can be federated and consistently managed using a hierarchical logical structure. The system has been successfully ported to several UNIX-based platforms and it is currently in use for over 500 clusters around the world. MonALISA consists of a collection of autonomous agent-based subsystems that cooperate in performing monitoring tasks. The system can be extended with additional data processing modules and libraries that are dynamically loaded. The described systems are primary used for monitoring performance-related parameters, like CPU load, disk and network utilization. Therefore, they are inappropriate for service-oriented environments, where the goal is to record and correlate messaging interactions among services.

Dynasoar [9] and MidArc [10] monitoring frameworks include e-business usage tracking middleware components based on SOAP-intermediaries [11]. They are both designed for service-oriented systems where they provide facilities for SOAP message interception, filtering, and data extraction. However, both of these frameworks are not based on a service-oriented architecture and they have been custom developed for specific middleware platforms. Thus, they do not provide a usage tracking solution that can be seamlessly integrated with various service-oriented systems.

View-based approach to tracking services [12] enables users to refine composite service specifications according to changes of context in their environment. The use of context-awareness at service level enables deployment and execution of highly personalized and flexible composite services. Although very effective for process-centric adaptations, this specialized tracking method may not be seamlessly used for enforcement of service-level accounting, security auditing, and health monitoring.

Significant research on usage tracking has been done. However, usage tracking in service-oriented environments imposes some new challenges which have not yet been addressed. Since services are dynamically deployed, usage tracking configuration management must be simple and flexible. Usage tracking must be enforced unobtrusively and without altering the logic of tracked services. The collected data must be persistent and available even after the tracked services are not longer available. To that end, we have designed a usage tracking architecture that provides a holistic solution aimed to meet these requirements.

3. Programmable Internet Environment

Rapidly expanding population of Internet users brings diversity in online activities. Due to great variety in web usage patterns, it is becoming impossible to foresee what new innovative services would best match individual needs of each user. However, if users are empowered with ability to construct services by themselves, they could adjust and personalize their everyday web-related activities.

As part of our previous work, we have developed a service-oriented environment PIE (Programmable Internet Environment), which enables web users to compose services by assembling highly personalized web-processes. PIE is a distributed and dynamic middleware system that brings together service composers, users, and administrators. Service composers join PIE with a set of Internet hosts, deploy their application services, compose services into new value-added services, and offer services to users residing outside PIE. System administrators are responsible for maintenance and upgrades of PIE system middleware components.

Figure 1 presents a sample configuration of PIE with 3 nodes deployed on separate Internet hosts (a,b,c). PIE
nodes include subsystems: *Overlay*, *OpenResources*, *OpenCollectives*, and *Access*. *Overlay* [13] is an application-level virtual network communication service, *OpenResources* (OR) [14] is a service deployment and discovery subsystem, *OpenCollectives* (OC) [15] is subsystem responsible for execution of composite services, while *Access* [16] subsystem enforces access control for services within PIE.

A sample composite service shown by Figure 1 consist of application services (AS1, AS2), distributed programs (DP1, DP2), and a coordination mechanism (CM) deployed on three PIE nodes (1). Application services implement arbitrary application-specific functionalities. Distributed programs implement logic for coordination of application services into composite service workflow (2, 3). The logic of distributed programs is written in script-like language SSCL [17] (Simple Service Composition Language). Coordination mechanisms [18] are services that enable communication and synchronization of distributed programs (4, 5).

4. Usage tracking services architecture

The developed usage tracking components for middleware systems are based on a service-oriented architecture with two classes of loosely-coupled services: Global Usage Tracking (GUT) and Local Usage Tracking (LUT) services. GUT service enables users to access stored usage tracking records by submitting custom queries. In order to resolve user queries, GUT service uses aggregated usage tracking records and metadata gathered from a set of LUT services. LUT services are deployed on nodes of host distributed system along with other application services. The purpose of each LUT service is to intercept incoming and outgoing SOAP traffic and record service invocation events for application services whose usage is begin tracked. Since host distributed system makes no distinction between usage tracking services and any other application service, GUT and LUT services can be seamlessly integrated with various host service-oriented distributed systems in order to enforce usage tracking.

Figure 2 presents a sample distributed system with a set of GUT and LUT services. The presented system consists of 3 nodes deployed on separate hosts (a, b, c). Each node of host system includes a layer with Service Deployment Middleware (SDM) facilities. SDM provides functionalities for deployment and execution of application services (AS1, AS3) in resource pools of each host system node. The deployed application services cooperate by exchanging SOAP messages in order to accomplish distributed application goals (1-3). In order to enforce usage tracking of application services, system user deploys single GUT service on node 1 (4) and LUT services on nodes 2 and 3 (5). In order to integrate them with the host system, GUT and LUT services are registered with SDM service available on their host nodes (6-8). In addition, LUT services also register with GUT service as sources of aggregated usage tracking records (9-10).

The placement of GUT and LUT services forms a single user-defined configuration for usage tracking. GUT and LUT services are flexible and can be used for multiple concurrent usage tracking configurations. For instance, a different user may deploy an additional GUT service on Node 3 and register LUT service available on node 2 as data source for that new GUT service. In order to prevent collisions, GUT and LUT services store data related to different usage tracking configurations in separate storage space.

![Figure 2. Usage tracking services deployment and management](image-url)
Having the usage tracking middleware components defined as services brings two important benefits. The designed usage tracking services are adaptable to various service-oriented environments where they can be seamlessly deployed as any other application service. Thus, GUT and LUT services can be dynamically deployed on system nodes without disturbing the host system’s operation. The second benefit is simplified management of usage tracking configurations. Since LUT and GUT services are loosely-coupled, system users may have different concurrent usage tracking configurations that can be changed at runtime. Users can setup and manage independent usage tracking configurations for a subset of application services that they own within a service-oriented environment.

Once GUT and a set of LUT services are deployed, they collaboratively enforce usage tracking. Figure 3 describes interactions between usage tracking services during system operation. After deployment on Node 2, LUT service is registered with SOAP Engine of local SDM for reception of all incoming and outgoing SOAP traffic on host node. Furthermore, system user registers a usage tracking filter with LUT service (1,2). Usage tracking filter specifies which application services on the host node will be tracked. SOAP Engine mediates all request and response messages exchanged between each local application service and remote clients (3-6). Since LUT service is registered for reception of all SOAP traffic of the local node, SOAP Engine forwards a copy of each SOAP message it receives to LUT service (7). Furthermore, LUT service analyzes the received SOAP messages, builds the corresponding usage records, and exchanges usage tracking records and metadata with GUT service (8). During system operation, system user may issue custom queries to GUT service (9) in order to obtain specific usage records. In order to resolve user queries, GUT service analyzes aggregated usage tracking metadata, obtains the resulting usage records and forwards them to the user (10).

The described usage tracking middleware services have been implemented using languages and tools available in .NET development framework. Furthermore, PIE middleware system has been used as integration and test platform for the developed usage tracking services.

4.1. Global usage tracking service

Global Usage Tracking (GUT) service enables users to inspect collected usage tracking records. As shown by Figure 4, GUT service consists of Query Resolver, Usage Records Manager, Directory, and Local Storage modules.

Usage Records Manager is responsible for collecting two types of data: usage tracking records obtained from remote LUT services and addresses of remote LUT services where usage records are stored (1). Received usage records are stored in Local Storage (2), while addresses of LUT services are stored in Directory (3). Query Resolver provides interface through which system users can issue queries for obtaining usage records that are gathered during system operation. On reception of a user query (4), Query Resolver uses Directory (5) to resolve the location where relevant usage records are stored. Usage records are obtained either from Local Storage (6) or pulled from remote LUT services (7), and forwarded to the user (8).

4.2. Local usage tracking service

Local Usage Tracking (LUT) service analyzes SOAP traffic on host node, detects service invocation events for application services, and creates the corresponding usage records. LUT service consists of Message Queue,
Message Analyzer, Configuration Manager, Usage Records Manager, and Local Storage modules shown by Figure 5.

Message Queue implements a FIFO (First-In-First-Out) data structure for storing messages received from SOAP Router (1). Message Analyzer pulls SOAP messages stored in the queue (2), obtains usage tracking filters from Configuration Manager (3), and matches the filters against received messages in order to create the corresponding usage records. System users register their usage tracking filters through Configuration Manager access interface (4,5). Registered filters are stored to the Local Storage module (6). On reception of usage records (7), Usage Records Manager may take one of the following two actions: (i) store the received usage records to Local Storage (8) and forward the host’s node address to GUT service (9), or (ii) forward the received usage records to GUT service (9) without storing it to the Local Storage. Which action will be taken, depends on the type of usage records management protocol that is being used by the LUT service.

5. Usage records management protocols

Usage records management protocols are used by GUT and LUT services to exchange usage tracking records and metadata. GUT and LUT services use custom-designed versions of push and pull protocols for dissemination of usage records, which are typically used in web-enabled systems [19].

5.1. Push protocol

LUT services that use push protocol forward each generated usage record to GUT service without storing it to Local Storage. GUT service stores each received usage record to its Local Storage module and each usage record is registered with the Directory module. Therefore, all usage records generated by registered LUT services are centrally stored in GUT service Local Storage module. This enables GUT service to resolve all user queries locally without issuing requests to remote LUT services.

5.2. Pull protocol

LUT services that use pull protocol store usage records to Local Storage module and send address of the host node and other metadata describing usage record attributes to GUT service. GUT service receives the metadata and stores it to its Directory module. When GUT service Query Resolver receives a user query, it performs a lookup to Directory module in order to determine the address of LUT services that store usage records relevant to the given query. Afterwards, each LUT service that may contain relevant records receives user query from GUT service and matches the query against locally stored usage records. Matched usage records are forwarded to GUT service. GUT service integrates the results from LUT services and forwards the resulting usage records to the user.

6. Languages for usage tracking

Users of GUT and LUT service use two XML-based languages for enforcement of usage tracking: Usage Tracking Filtering Language (UTFL) and Usage Tracking Query Language (UTQL). UTFL enables users to specify their usage tracking requirements for each application service. UTQL enables users to specify which type of usage records they wish to obtain from GUT service.

Figure 6 presents samples of a UTFL, usage record, and UTQL documents. UTFL document contains

Figure 6. Samples of usage tracking filter, record, and query
<UsageTrackingFilter> root element. Root element encapsulates <Service> element with attribute that specifies the name of service for which usage tracking is enforced. In addition, <Service> element may contain a set of <Operation> elements. Each <Operation> element contains an attribute with the name of service operation for which invocations are being tracked. <Operation> elements also contain <ReqHeader>, <ReqBody>, <ResHeader>, and <ResBody> standard elements. Elements <ReqHeader> and <ResHeader> include XPath language queries that specify XML segments in SOAP request and response headers that will be recorded on service invocation. Furthermore, elements <ReqBody> and <ResBody> include XPath language queries that specify XML segments in body of SOAP request and response messages that will be recorded on service invocation. Once a filter document is registered with a LUT service, LUT service uses XPath queries to extract the data from received SOAP messages and generate the corresponding usage records. Each usage record contains a set of standardized elements described in Table 1. Element <content> includes application-specific XML data extracted from SOAP messages using XPath queries specified in the filter document (a).

### Table 1. Descriptions of usage record elements

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ClientID&gt;</td>
<td>Name of the remote client that invoked the service</td>
</tr>
<tr>
<td>&lt;ServiceID&gt;</td>
<td>Name of invoked service</td>
</tr>
<tr>
<td>&lt;ServiceAddr&gt;</td>
<td>Address of invoked service</td>
</tr>
<tr>
<td>&lt;OperationID&gt;</td>
<td>Invoked operation name</td>
</tr>
<tr>
<td>&lt;Time&gt;</td>
<td>Time of service invocation</td>
</tr>
<tr>
<td>&lt;Content&gt;</td>
<td>XML data extracted from SOAP request and response messages</td>
</tr>
</tbody>
</table>

As shown by Figure 6, user queries include most of the elements contained in usage records. Each element may contain concrete value or a wildcard * symbol. In order to resolve a query, query element values are matched against the values of the corresponding usage record elements (b).

### 7. Performance evaluation

A generic distributed test environment for service-oriented systems has been used in order to measure the efficiency of usage records dissemination using push and pull protocols.

Figure 7 presents test environment configuration including Testing System and Tested System. Testing System consists of a Controller and a set of Load Generator components deployed on separate nodes connected in a LAN. Test administrator uses the Controller to define test parameters (1), and initiate the test by forwarding test parameters to Load Generators (2). During test period, Load Generators invoke Test Service (3) according to the specified parameters. Parameters include test duration, Test Service address, average pause interval between each consecutive service invocation ($T_P$), and XML payload sent to the Test Service.

Tested System consists of a GUT service, LUT service, and Test Service. Test Service is a simple service that receives SOAP requests of 1000 kB in size, spends 0.5 seconds in an active loop generating CPU load, and returns the received SOAP message intact to the remote client. During the test, LUT and GUT services exchange usage records using either push or pull protocol depending on the chosen test configuration (4). LUT services are extended with performance counters for measuring local network traffic utilization and CPU load, while GUT service includes counters for measuring delay for resolving user queries. The counters are used to gather the statistics on resource utilization during system operation. Upon test end, Controller issues sample queries to GUT service in order to let GUT service measure delay for resolving user queries (5). Finally, Controller gathers statistics on CPU load, network utilization, and query resolution delays from LUT and GUT services (6).

Test results are presented in Figure 8. Figure 8.a) shows average CPU load generated on the node with LUT service in dependency of service invocation pause interval $T_P$. If LUT is using pull protocol, with increase in
service invocation period $T_P$, the generated CPU load is at first slightly raised and then steadily decreased. However, if LUT is using push protocol, CPU load is rapidly decreasing as invocation period $T_P$ becomes greater than 5 seconds. These results indicate that under light service invocation loads (larger $T_P$) push protocol generates less CPU load than pull protocol. When service invocation load is light, the local network bandwidth is not heavily utilized. Thus, the overhead of forwarding complete usage records to GUT service is less CPU-intensive than storing usage records to host node file system and forwarding address reference of the host node to GUT service. However, with heavier service invocation loads (smaller $T_P$) more local network bandwidth is utilized and host node’s network connection becomes a bottleneck. Thus, it becomes more CPU-intense to forward the entire usage record to GUT than to store the usage record locally and forward only a small packet of data containing address of the local node to GUT service.

Figure 8.b) shows the level of utilized network bandwidth in dependency of service invocation pause interval $T_P$. The utilized network bandwidth consists of test service invocation traffic and traffic generated by usage records management protocols. The results indicate that pull protocol uses about 15% less of network bandwidth compared to push protocol. Furthermore, increase in service invocation period $T_P$ rapidly lowers the level of utilized network bandwidth. These results indicate that it is more efficient to use pull protocol than push protocol when network bandwidth is a scarce resource.

The measurements of average time required for resolving user queries show that use of push protocol incurs significantly lower delays than pull protocol. For instance, when Test Service is invoked with service invocation pause interval $T_P=0.1$ seconds and pull usage records management protocol is in use, it takes 0.7 seconds on the average to resolve a user query. However, when pull usage records management protocol is in use, it takes 12 seconds on the average to resolve a user query. The use of pull protocol incurs more time to resolve user queries since for resolving each user query usage records have to be obtained from remote LUT services.

The choice between push and pull protocol should be made depending on the expected load of application service invocation. If application service will be frequently invoked by remote clients, pull protocol should be used to reduce CPU load and network bandwidth utilization. However, the use of pull protocol will increase the delays for resolving user queries. Thus, if application service will not be invoked frequently, push protocol should be chosen in order to minimize the delays for resolving user queries.

8. Conclusion

Usage tracking is the basis for various high-level business services, like user profiling, security auditing, and decision support making. Thus, it has been actively researched for various web-enabled systems. However, service-oriented environments impose some unique usage tracking requirements which have to be addressed. Typical requirements are seamless integration, unobtrusive operation, and simplified configuration management of usage tracking services.

In this paper, we present architecture of service-oriented middleware components for usage tracking. Main contributions of the presented architecture are twofold: (i) adaptability to various service-oriented distributed systems and (ii) simplified usage tracking
configuration management. The architecture is based on lightweight and adaptable usage tracking services that can be integrated with various service-oriented middleware systems in order to enforce usage tracking. With usage tracking services, users can dynamically deploy and manage multiple usage tracking configurations without disturbing operation of host distributed system. To that end, simple XML-based languages are used to describe usage tracking requirements and query the system for recorded usage records. Usage tracking services exchange usage tracking metadata using specialized push and pull protocols. In order to determine the efficiency of push and pull protocols, we have measured their performance in typical usage tracking configurations. The results indicate that the protocol choice should be made depending on the expected application service invocation load.

As part of the future work, the described architecture will be extended with support for usage tracking of composite services. To that end, we plan to research methods for aggregation and clustering of records related to services forming composite services. Furthermore, we will design an adaptive performance-aware dissemination protocol that will dynamically switch between push and pull mode of operation.

Acknowledgements
The authors wish to thank Ivan Zuzak, Dejan Skvorc, Miroslav Popovic, Ivan Benc, and Ivan Skuliber for their valuable comments, and Vedran Najrajter for conducting the performance tests.

9. References