SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid

Hong-Linh Truong[‡][*] Thomas Fahringer [+]

[‡]Institute for Software Science, University of Vienna
Nordbergstrasse 15/C/3, A-1090 Vienna, Austria
truong@par.univie.ac.at

[+]Institute for Computer Science, University of Innsbruck
Technikerstrasse 13, A-6020 Innsbruck, Austria
Thomas.Fahringer@uibk.ac.at

Abstract

This paper describes SCALEA-G, a unified monitoring and performance analysis system for the Grid. SCALEA-G is implemented as a set of grid services based on the Open Grid Services Architecture (OGSA). SCALEA-G provides an infrastructure for conducting online monitoring and performance analysis of a variety of Grid services including computational and network resources, and Grid applications. Both push and pull models are supported, providing flexible and scalable monitoring and performance analysis. Source code and dynamic instrumentation are implemented to perform profiling and monitoring of Grid applications. A novel instrumentation request language for dynamic instrumentation and a standardized intermediate representation for binary code have been developed to facilitate the interaction between client and instrumentation services.

1 Introduction

Grid Monitoring is crucial task that provides useful information for several purposes such as performance analysis and tuning, performance prediction, fault detection, resource brokering and scheduling. Most existing Grid monitoring tools are separated into two distinct domains: *Grid infrastructure monitoring* and *Grid application monitoring*. The lack of an integrated system has hindered the detection of correlations among measurement metrics of various sources at different levels of details. Grid monitoring tools that combine application and system monitoring and performance analysis are crucial as these tools will provide the user a unified view that uncovers the correlations among performance metrics from various sources. In addition, many existing Grid monitoring tools focus on the monitoring and analysis for Grid infrastructure; yet little effort has been done to examine Grid applications. To date, application performance analysis tools are mostly targeted to conventional parallel and distributed systems (e.g. clusters, SMP machines). As a result, these tools are not well adjusted to the scalability, diversity, dynamics and security of the Grid.

To tackle the above-mentioned challenges, we are developing a new system named SCALEA-G which is a unified system for monitoring and performance analysis in the Grid. SCALEA-G is implemented as a set of OGSA-based services [15]. It provides an infrastructure of OGSA-compliant grid services for online monitoring and performance analysis of a variety of Grid services including computational resources,

[*] Corresponding author

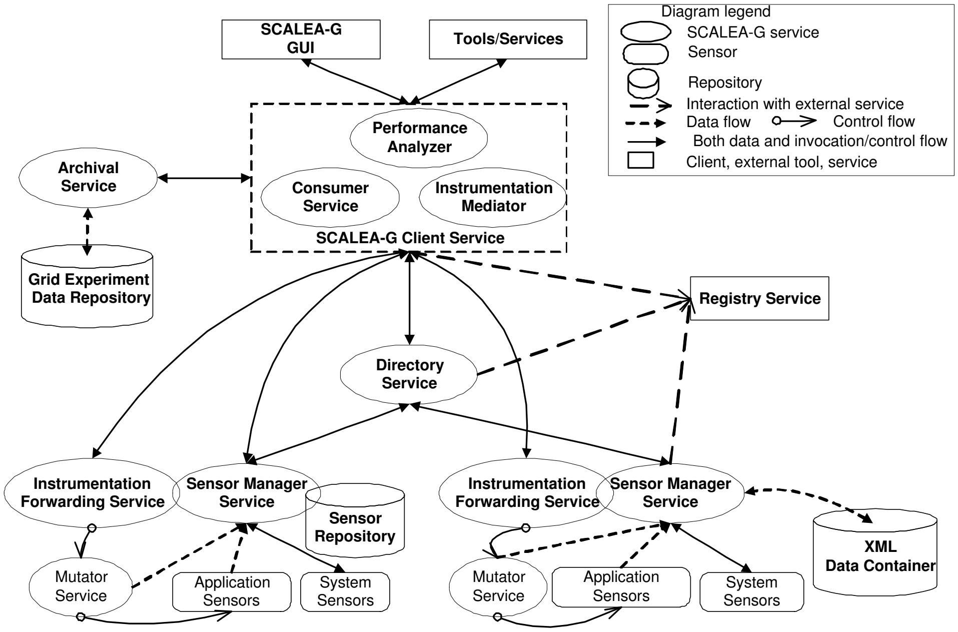


Figure 1: High-level view of SCALEA-G Architecture.

networks, and applications. Both push and pull models proposed in GMA (Grid Monitoring Architecture) [3] are supported, providing a flexible and scalable mechanism for Grid monitoring and performance analysis. In SCALEA-G, each type of monitoring data is described by an XML schema [29], allowing any client to easily access the data via XPath [28]. SCALEA-G supports both source code and dynamic instrumentation for profiling and monitoring events of Grid applications. A novel instrumentation request language has been devised to facilitate the interaction between client and instrumentation services. System and application specific metrics are related to each other in a single system, thus increasing the chance to uncover Grid performance problems and their dependences.

This paper focuses on the design of the SCALEA-G system. The rest of this paper is organized as follows: Section 2 presents the architecture of SCALEA-G. In Section 3, we describe SCALEA-G sensors and Sensor Manager Service. Section 4 describes the dynamic instrumentation service for Grid applications. We then discuss the data delivery, caching and filtering mechanism in Section 5. Security issues in SCALEA-G are outlined in Section 6. Section 7 illustrates first experiments and examples of the current prototype. We present some related work in Section 8 before concluding with an outlook to future work in Section 9.

2 SCALEA-G Architecture

SCALEA-G is an open architecture based on OGSA [15], combined with GMA [3]. Figure 1 depicts the architecture of SCALEA-G which consists of a set of OGSA-based services and clients.

The *SCALEA-G Directory Service* (DS) is used for publishing and searching information about producers and consumers that produce and consume performance data, and information about types and characteristics of that data. The *Archival Service* (AS) is a data repository which is used to store monitoring data and performance results collected and analyzed by other components. The *Sensor Manager Service* (SM) is used to manage sensors that gather and/or measure a variety of types of data for monitoring and performance analysis, to store monitoring data and to provide that data to consumers.

Application instrumentation can be done at source code level manually, automatically or dynamically at the runtime. The *Instrumentation Forwarding Service* (IFS) receives instrumentation requests from clients and forwards the requests to the *Mutator Service* (MS) which conducts the dynamic instrumentation. The *Client Service* (CS) provides interfaces for administrating other SCALEA-G services and accessing data in these services. In addition, it provides features for analyzing performance data. Any external tools and services can access SCALEA-G by using CS. A *GUI* - supported by CS - enables the user to graphically examine monitoring and performance analysis results. SCALEA-G services register and search information about their service instances in *Registry Services*.

Interactions among SCALEA-G services are divided into *Grid service-based operations* and *TCP-based stream data delivery*. Grid service operations are used to perform tasks which include controlling activities of services and sensors, subscribing and querying performance data, registering, querying and receiving information from DS. In stream data delivery mode, a TCP channel is used to transfer monitoring data, performance data and results among producers (e.g. sensors, SMs) and consumers (e.g. SMs, clients). Grid service operations incorporate transport-level and message-level security whereas data channel is based on a secure connection; security in SCALEA-G relies on Grid Security Infrastructure (GSI) [26].

When deploying SCALEA-G, instances of sensors and MSs are executed in compute nodes being monitored. An instance of SM can be deployed to manage multiple sensors and MSs in a node or a set of nodes, depending on the real system and workload. Similarly, SMs in different administrative domains can publish information with multiple DS instances. The client discovers SCALEA-G services through Registry Services which can be deployed in different domains.

3 Sensors and Sensor Manager Service

3.1 System Sensors and Application Sensors

SCALEA-G distinguishes two kinds of sensors: *system sensors* and *application sensors*. System sensors are used to monitor and measure the performance of Grid infrastructure (e.g. compute hosts, network connections) whereas application sensors are used to measure execution behavior of code regions and to monitor user-defined events in Grid applications. Sensors freely customize their collected data which is expressed in XML. SCALEA-G services may not be aware of monitoring data structures, supposing the data is in XML representation. All sensors are associated with some common properties such as sensor identifier, data schema, parameters and interact with SMs by exchanging XML messages.

Most system sensors provide dynamic information, e.g. the available bandwidth of a network path, CPU usage of a compute node, but few can provide static information, e.g hardware information about compute hosts. System sensors can query or collect data from other providers. Figure 2 presents an excerpt of XML schema of the data provided by a sensor named `path.bandwidth.capacity.TCP` which is used to measure the bandwidth capacity of a network path (between two compute nodes) in the Grid. SCALEA-G provides a variety of system sensors for monitoring the most commonly needed types of performance data on the Grid investigated by GGF DAMED-WG [11] and NMWG [18].

Application sensors collect profiling and event data of Grid applications. Figure 3 shows the top-level XML schema for profiling data provided by application sensors. The `name` attribute corresponds to the identifier of profiling sensor. The `exp` element specifies a unique identifier determining the experiment. This identifier is used to distinguish data between different experiments. The `cr` element refers to source information of the code region (e.g. line, column, function name). The `pu` element describes the context in which the code region is executed; the context includes information about Grid site, computational node, process, thread. The `metrics` element refers to performance metrics, each metric is represented in a (name, value) tuple. A similar sensor is defined for describing event data in applications.

Both sensors types are treated basically the same. They, however, differ in their control (e.g. activation, instrumentation) and security model (e.g. permission in accessing data). This distinction allows us to simplify the management of two different types of sensors.

```
<xsd:complexType name="SensorData">
 <xsd:sequence>
  <xsd:element name="source"
                         type="xsd:string"/>
  <xsd:element name="destination"
                         type="xsd:string"/>
  <xsd:element name="eventtime"
                           type="xsd:long"/>
  <xsd:element name="bandwidth"
                         type="xsd:double"/>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:NMTOKEN"
       fixed="path.bandwidth.capacity.TCP"/>
 <xsd:attribute name="resourceID"
                          type="xsd:string"/>
</xsd:complexType>
```

Figure 2: XML schema of data provided by path.bandwidth.capacity.TCP sensor.

```
<xsd:complexType name="SensorData">
  <xsd:sequence>
  <xsd:element name="exp" type="xsd:string"/>
  <xsd:element name="pu" type="PU"
                minOccurs="0" maxOccurs="1"/>
   <xsd:element name="cr" type="CR"
                minOccurs="0" maxOccurs="1"/>
   <xsd:element name="metrics"
                type="MetricList"
                minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="name"
        type="xsd:NMTOKEN" fixed="app.prof"/>
  <xsd:attribute name="resourceID"
                           type="xsd:string"/>
</xsd:complexType>
```

Figure 3: Top-level XML schema of application profiling data.

3.2 Sensor Repository

To simplify the management and deployment of system sensors and the processing of sensor data, a *sensor repository* is used to hold (i) information about available system sensors, (ii) information about sensors which will be activated at start-up time, and (iii) registered XML schemas. Each sensor repository is managed by an SM which can activate available sensors in the repository when requested.

```
<xsd:element name="sensor" type="SensorDescription"
               minOccurs="0" maxOccurs="unbounded"/>
<xsd:complexType name="SensorDescription">
 <xsd:sequence>
        <xsd:element name="measureclass"
                                type="xsd:string"/>
        <xsd:element name="desc"
    type="xsd:string" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="schemafile"
                                type="xsd:string"/>
        <xsd:element name="params"
  type="ParamsEntry"  minOccurs="0" maxOccurs="1"/>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:NMTOKEN"/>
 <xsd:attribute name="ondemand" type="xsd:boolean"/>
</xsd:complexType>
```

```
<xsd:complexType name="ParamsEntry">
 <xsd:sequence>
   <xsd:element name="param"
                   type="ParamEntry"
 minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ParamEntry">
    <xsd:attribute name="name"
                type="xsd:string"/>
    <xsd:attribute name="desc"
                type="xsd:string"/>
    <xsd:attribute name="dataType"
                type="xsd:string"/>
</xsd:complexType>
```

Figure 4: XML schema used to describe sensors in the sensor repository.

Figure 4 displays an excerpt of the XML schema used to describe sensors in the sensor repository. The XML schema is used to specify sensor-related information such as `name` (a unique name of the sensor), `measureclass` (implementation class), `schemafile` (XML schema of data produced by the sensor), `params` (parameters required for invoking the sensor), etc. Although not specified in the repository, by default the lifetime of a sensor instance will optionally be specified when the sensor instance is created.

3.3 Sensor Manager Service

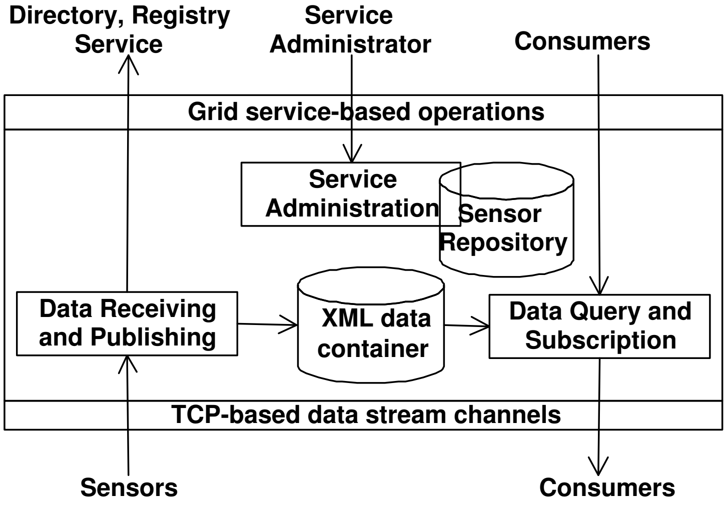

Directory, Registry Service
Service Administrator
Consumers
Grid service-based operations
Service Administration
Sensor Repository
Data Receiving and Publishing
XML data container
Data Query and Subscription
TCP-based data stream channels
Sensors
Consumers

Figure 5: Sensor Manager Service Implementation

The main tasks of Sensor Manager Service (SM) are to control and manage activities of sensors in the sensor repository, to publish information about data collected by sensors to DSs, to receive and buffer monitoring data sensors produce, to provide monitoring data to consumers via data query and subscription (DQS). SM includes the following components: Service Administration, Data Query and Subscription, Data Receiving and Publishing as shown in Figure 5. SM provides control and request tasks, data receiving and delivery over both Grid-based service operations and TCP-based data streams.

The *Service Administration* component receives requests from the SCALEA-G administrator and controls activities of SMs and their sensors in the sensor repository. The administrator can activate any sensor (thus making a new sensor instance) or deactivate an existing sensor instance. When a new sensor instance is activated, it will be added into the list of sensor instances. Similarly, a sensor instance will be removed from this list when it is deactivated. The administrator can perform the registration (adding, updating or removing) information about the SM, properties of data provided by sensor instances with selected DSs. All the interactions in this component are carried out through invocations of Grid-based service operations.

The *Data Query and Subscription* component is responsible for processing DQS requests from consumers. DQS tasks are implemented as service operations but monitoring data is delivered via TCP-based data streams. Based on the information published in the DS, consumers can subscribe monitoring data provided by sensor instances that is archived in SMs. The resulting data will be *pushed* to consumers via TCP-based data channels. When a consumer unsubscribes a data type, the DQS component will stop sending that data to the consumer. To support the *pull* mode, this component processes queries with constraints (produced by consumers) to filter data of interest. The resulting data satisfying the requested constraints will be sent back to the consumer. DQS requests can be represented in XPath based on the published XML schema.

The *Data Receiving and Publishing* component conducts two tasks. Firstly, it receives data collected by sensor instances. A data receiver is used to receive data from sensors. It then processes and stores the received data into data buffers. The data receiver is a thread binding a well-known port and interacts with sensors via TCP connections. Data is stored in buffers in XML data containers. Secondly, it implements functionality that publishes (adding, updating, removing) of information about SM and properties of monitoring data to DS.

3.4 Interactions between Sensors and Sensor Manager Services

The interactions among sensors and SMs involve the exchange of three XML messages. In the *initialization phase*, the sensor instance sends a `sensorinit` XML message which contains sensor name, optionally an XML schema of monitoring data, lifetime and description information about the sensor instance to the SM which then makes these information available for consumers via DS. In the *measurement phase*, the sensor instance repeatedly performs measurement, encapsulates its measurement data into a `sensordataentry` XML message, and pushes the message to the SM. The measurement data is enclosed by `<![CDATA[ ... ]]>` tags thus sensors can customize the structure of their collected data. SM gets measurement data and stores the data into XML containers in native format. In the *finalization phase*, before stopping sending collected data, the sensor instance sends a `sensorfinal` XML message to notify the SM.

Each message in the above protocol is self-explained. Therefore, multiple types of monitoring data can be delivered via a connection which can be transient, not necessarily persistent.

4 Instrumentation Service

We support two approaches: source code and dynamic instrumentation. In the first approach, source code can be automatically instrumented by a service that is based on the SCALEA Instrumentation System [25]. This approach, however, simply instruments input source files (for Fortran), not addressing compilation issue (e.g. to compile instrumented code in different architectures). Thus, the client has to compile and link the instrumented files with the measurement library containing application sensors. Moreover, source code instrumentation can be done manually by the end-user.

In the second approach, we exploit the dynamic instrumentation mechanism based on Dyninst [8]. With dynamic instrumentation, applications remain unchanged. A *Mutator Service* (MS) is implemented as a GSI-based SOAP C++ Web service based on gSOAP toolkit [21] that controls the instrumentation of application processes on the host where the processes are running. We develop an XML-based instrumentation request language (IRL) to allow the client to specify code regions of which performance metrics should be determined and to control the instrumentation process. The client controls the instrumentation by sending IRL requests to MSs which in turn perform the instrumentation, e.g. inserting application sensors into application processes. MS hides all the low level detail of the instrumentation process.

Whereas IRL allows to specify instrumentation requests containing information about code regions that should be instrumented, the question is how clients understand application structures in order to specify selected code regions and by what means MS describes the application structures. In our framework, MS provides the application structure to the client in SIRBC (Standardized Intermediate Representation for Binary Code) format which is based on XML. Based on SIRBC, the client can decide which code regions should be instrumented. IRL, the language for the conversation between the instrumentation requester and engine, and SIRBC, the language for describing the structure of the application being instrumented, are central elements of the dynamic instrumentation service. Providing interfaces via service operations, together with IRL and SIRBC, the instrumentation service is highly interoperable and it can easily be integrated and used by other services. In the following section, we discuss SIRBC and IRL.

4.1 SIR for Binary Code

Normally the performance tool developers have to build separate instrumentation engines for different programming languages and instrumentation strategies such as dynamic and static instrumentation. This work is a time consuming effort. The APART working group has proposed a Standardized Intermediate Representation (SIR) as an abstract representation for procedural and object-oriented programs [14]. Basically a SIR contains information about statements and directive types with very little details on the structure of individual statements and directives. The idea is that high-level tools must only be aware of the type of a statement in order to make a decision about what code regions should be instrumented.

SIR is an XML-based representation that includes information about program units (e.g functions, methods), code regions (e.g. function calls, loops, statements,), etc. SIR is designed for describing Fortran, Java, C and C++ programs. Therefore, it supports a rich set of information about programs having different data types and following different programming paradigms. However, with dynamic instrumentation, in which the intended instrumented program is available in binary code only, the information obtained is substantially reduced. With binary code, we mostly can obtain information and instrument at the level of program units, function calls and loops. Thus, a simplified version of SIR would be more suitable as it does not require the instrumentation service to implement all features proposed by SIR. We develop SIRBC for that purpose.

```
<xsd:element name="sirbc"   type="SIR"/>
<xsd:complexType name="SIR">
  <xsd:sequence>
    <xsd:element name="unit" type="SIRUnit" minOccurs="0"  maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SIRUnit">
  <xsd:sequence>
    <xsd:element name="coderegion" type="SIRCodeRegion" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="type" type="xsd:string"  minOccurs="0"  maxOccurs="1"/>
  <xsd:attribute name="name" type="xsd:string"  minOccurs="0"  maxOccurs="1"/>
  <xsd:attribute name="id" type="xsd:string"  minOccurs="0"  maxOccurs="1"/>
</xsd:complexType>
<xsd:complexType name="SIRCodeRegion">
 <xsd:sequence>
  <xsd:element name="callee" type="SIRCallee"  minOccurs="0"  maxOccurs="1"/>
 </xsd:sequence>
 <xsd:attribute name="type" type="xsd:string"/>
  <xsd:attribute name="id" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="SIRCallee">
  <xsd:sequence>
      <xsd:element name="linestart" type="xsd:integer"  minOccurs="0"  maxOccurs="1"/>
      <xsd:element name="lineend" type="xsd:integer"  minOccurs="0"  maxOccurs="1"/>
      <xsd:element name="sourcefile" type="xsd:string"  minOccurs="0"  maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
```

Figure 6: SIR for Binary Code

Figure 6 presents the XML schema of SIRBC which is based on the idea of SIR. Currently SIRBC supports only program unit and function call representation. An application process is represented as a set of program units (element `SIRUnit`). Each program unit contains a set of code regions (element `SIRCodeRegion`). A code region is a function call that contains information (e.g. name, source line) about the callee function (element `SIRCallee`). Each program unit, code region is associated with a unique identifier (attribute `id`). The client uses that identifier to refer to the program unit or the code region. By using SIRBC, the instrumentation requester can understand the application structure and specify instrumentation requests containing code regions that should be instrumented.

4.2 Instrumentation Request Language (IRL)

The IRL is provided in order to facilitate the interaction between instrumentation requester (e.g. users, tools) and instrumentation services. IRL which is an XML-based language consists of instrumentation messages: request and response. Clients send requests to MSs and receive responses that describe the status of the requests.

```
<xsd:element name="irl"   type="IRL"/>

<xsd:complexType name="IRL">
 <xsd:sequence>
  <xsd:element name="experiment"
       type="IRLExperiment"
                      minOccurs="0" maxOccurs="1"/>
  <xsd:element name="request"  type="IRLRequest"
             minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="response" type="IRLResponse"
              minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="IRLRequest">
  <xsd:sequence>
    <xsd:element name="experiment"
   type="IRLExperiment" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="task" type="IRLTask"
                minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:NMTOKEN"/>
</xsd:complexType>
<xsd:complexType name="IRLExperiment">
 <xsd:sequence>
 <xsd:element name="applicationName"
                                  type="xsd:string"/>
 <xsd:element name="jobID"        type="xsd:string"/>
 <xsd:element name="experimentID" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="IRLTask">
  <xsd:sequence>
    <xsd:element name="coderegion"
           type="CodeRegion"
minOccurs="1" maxOccurs="unbounded"/>
    <xsd:element name="metrics"
                 type="MetricList"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CodeRegion">
  <xsd:attribute name="unit"
               type="xsd:string"/>
  <xsd:attribute name="name"
                 type="xsd:string"/>
  <xsd:attribute name="id"
                 type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="IRLResponse">
 <xsd:sequence>
  <xsd:element name="detail"
                  type="xsd:string"/>
 </xsd:sequence>
 <xsd:attribute name="name"
                 type="xsd:NMTOKEN"/>
 <xsd:attribute name="status"
                 type="xsd:NMTOKEN"/>
</xsd:complexType>
<xsd:simpleType name="MetricList">
  <xsd:list itemType="xsd:string"/>
</xsd:simpleType>
```

Figure 7: Excerpt of XML schema of Instrumentation Request Language.

Figure 7 outlines the XML schema of IRL. The job to be instrumented is specified by `experiment` element. Current implementation of IRL supports four requests including `attach`, `getsir`, `instrument`, `finalize`:

- `attach`: requests MS to attach the application and to prepare to perform other tasks on that application.
- `getsir`: requests the MS to return SIRBC of a given application.
- `instrument`: specifies code regions (based on SIRBC) and performance metrics should be instrumented and measured.
- `finalize`: notifies MS that client will not perform any request on the given application.

In responding to a request from a client, MS will reply to the client by sending an instrumentation response which contains the name of the request, the status of the request (e.g. OK, FAIL) and possibly a detailed responding information encoded in `<![CDATA[ ...  ]]>` tags.

5 Data Delivery, Caching and Filtering

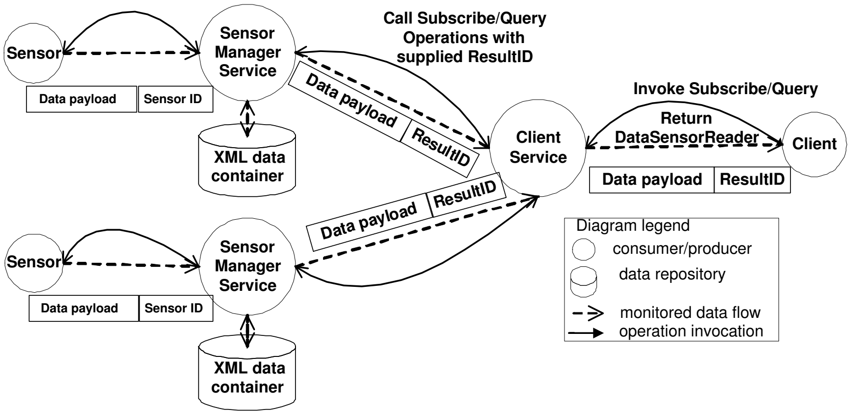


Figure 8: Data Delivery and Aggregation

Messages containing performance and monitoring data are delivered via separate data channel. Message propagation uses a simple tunnel protocol as presented in Figure 8. In this protocol, each sensor builds its XML data messages and sends the messages to an SM which stores the messages into appropriate buffers. When a client subscribes and/or queries data by invoking operations of Client Service (CS), CS calls corresponding operations of SM and passes a `ResultID` to the SM. CS returns to the client a `DataSensorReader`. The SM builds XML messages by tagging the `ResultID` to the data which satisfies the subscribed/queried constraints and sends these messages to CS. At CS side, based on `ResultID`, the messages are filtered and stored into `DataSensorReader`. The `ResultID` is used to aggregate resulting data of the same request delivered from multiple SMs. Clients can call APIs (blocking and non-blocking) or set up a call-back on `DataSensorReader` to get the resulting data.

Data produced by system sensors will be cached in circular bounded buffers at SM. In the current implementation, for each type of system sensor, a separate data buffer is allocated for holding data produced by all instances of that type of sensor. Data buffers keep monitoring data in its native format in XML data containers implemented atop Berkeley DB XML[2]. In the push mode, any new data entry satisfying the subscribed constraints will always be sent to the subscribed consumers. In the pull mode, SM only searches current available entries in the data buffer and returns entries satisfying constraints of consumer query to the requested consumers. Buffering data produced by application sensors is similar to that for system sensors. However, we assume that there is only one user to perform the monitoring and analysis for each application and the size of the data buffer is unbounded.

6 Security Issues

The security in SCALEA-G is based on GSI [26] facilities provided by Globus Toolkit (GT). Each service is identified by a certificate.

SCALEA-G imposes control on clients in accessing its services and data provided by system sensors through an Access Control List (ACL). The ACL maps client's information to sensors and services which can be accessed by the client. ACL contains information about service name, sensor identifier, subject of user certificate, tasks and permissions. The client information obtained from client's certificate when the certificate is used in authentication will be compared with entries in the ACL in the authorization process. For each user, an ACL entry specifies services, sensors, tasks (control and access the data) and permissions associated with these sensors in the services.

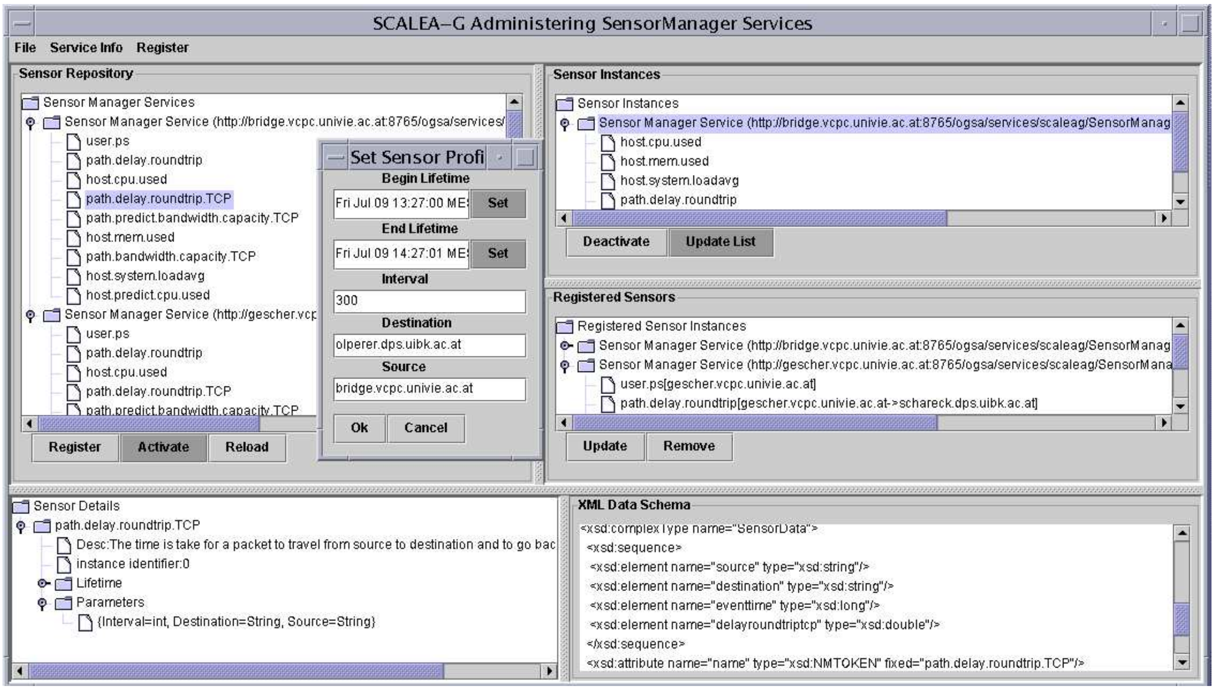
The security model for MS is a simplified version of that for GT3 GRAM [26] in which IFS (Instrumentation Forwarding Service) can forward instrumentation requests of clients to MS. MS runs in a non-privilege account. However, if MS is deployed to be used by multiple users, it must be able to create its instances running in the account of calling users. By doing so, the instances have permission to attach to user application processes and they can perform dynamic instrumentation.

In the case of monitoring and analyzing applications, when subscribing and/or querying data provided by application sensors, client's information will be recorded. Similarly, before application sensor instances start sending data to the SM, the SM obtains information about the client who executed the application. Both sources of information will be used for authorizing the client in receiving data collected by application sensors.

7 Experiments and Examples

We have prototyped SCALEA-G SM, DS, MS, a set of system and application sensors. Globus Toolkit [20] and JavaCog [22] are used for implementing OGSA-enabled services, communication and security in SCALEA-G. Application sensors are based on Globus GSS C APIs library. PostgreSQL [1] has been used for archiving information in DS. To buffer the data at SM, we use Berkeley DB XML provided by Sleepycat [2]. In this section we illustrate experiments.

7.1 Administrating Sensor Manager Services and Sensors



```
Sensor Manager Services
  Sensor Manager Service (http://bridge.vcpc.univie.ac.at:8765/ogsa/services/
    user.ps
    path.delay.roundtrip
    host.cpu.used
    path.delay.roundtrip.TCP
    path.predict.bandwidth.capacity.TCP
    host.mem.used
    path.bandwidth.capacity.TCP
    host.system.loadavg
    host.predict.cpu.used
  Sensor Manager Service (http://gescher.vcp
    user.ps
    path.delay.roundtrip
    host.cpu.used
    path.delay.roundtrip.TCP
    path.predict.bandwidth.capacity.TCP
```

Begin Lifetime: Fri Jul 09 13:27:00 ME

End Lifetime: Fri Jul 09 14:27:01 ME

Interval: 300

Destination: olperer.dps.uibk.ac.at

Source: bridge.vcpc.univie.ac.at

```
Sensor Instances
  Sensor Manager Service (http://bridge.vcpc.univie.ac.at:8765/ogsa/services/scaleag/SensorManag
    host.cpu.used
    host.mem.used
    host.system.loadavg
    path.delay.roundtrip
```

```
Registered Sensor Instances
  Sensor Manager Service (http://bridge.vcpc.univie.ac.at:8765/ogsa/services/scaleag/SensorManag
  Sensor Manager Service (http://gescher.vcpc.univie.ac.at:8765/ogsa/services/scaleag/SensorMana
    user.ps[gescher.vcpc.univie.ac.at]
    path.delay.roundtrip[gescher.vcpc.univie.ac.at->schareck.dps.uibk.ac.at]
```

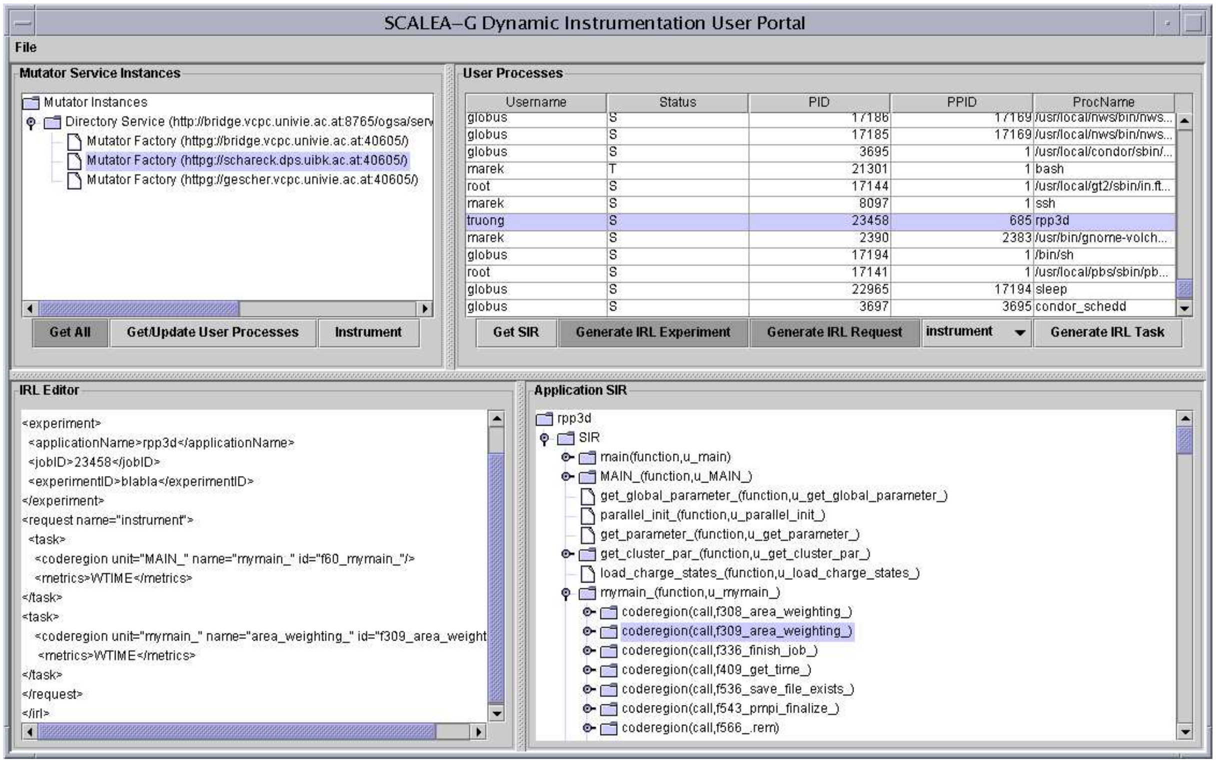
```
Sensor Details
  path.delay.roundtrip.TCP
    Desc:The time is take for a packet to travel from source to destination and to go bac
    instance identifier:0
    Lifetime
    Parameters
      {Interval=int, Destination=String, Source=String}
```

```
<xsd:complexType name="SensorData">
 <xsd:sequence>
  <xsd:element name="source" type="xsd:string"/>
  <xsd:element name="destination" type="xsd:string"/>
  <xsd:element name="eventtime" type="xsd:long"/>
  <xsd:element name="delayroundtriptcp" type="xsd:double"/>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:NMTOKEN" fixed="path.delay.roundtrip.TCP"/>
```

Figure 9: SCALEA-G Administration GUI.

Figure 9 displays the GUI used to manage activities of SMs. By selecting an SM, a list of available sensors in sensor repository and a list of sensor instances managed by that SM will be shown in the top-left window (titled *Sensor Repository*) and top-right window (titled *Sensor Instances*) of Figure 9, respectively. A user (with permission, controlled by ACL of SM) can make a request creating a new sensor instance by selecting a sensor in the list of available sensors, clicking the `Activate` button and specifying input parameters and lifetime. For example, Figure 9 shows the dialog for setting input parameters for `path.delay.roundtrip.TCP` sensor. An existing sensor instance can be deactivated by selecting `Deactivate` button. By choosing a sensor, detailed information of that sensor (e.g. parameters, XML schema) will be shown in the two bottom windows. In the middle-right window (titled *Registered Sensors*), we can examine which sensor instances that register to a SM.

7.2 Dynamic Instrumentation



```
Mutator Instances
  Directory Service (http://bridge.vcpc.univie.ac.at:8765/ogsa/serv
    Mutator Factory (httpg://bridge.vcpc.univie.ac.at:40605/)
    Mutator Factory (httpg://schareck.dps.uibk.ac.at:40605/)
    Mutator Factory (httpg://gescher.vcpc.univie.ac.at:40605/)
```

Username	Status	PID	PPID	ProcName
globus	S	17186	17169	/usr/local/nws/bin/nws...
globus	S	17185	17169	/usr/local/nws/bin/nws...
globus	S	3695	1	/usr/local/condor/sbin/...
marek	T	21301	1	bash
root	S	17144	1	/usr/local/gt2/sbin/in.ft...
marek	S	8097	1	ssh
truong	S	23458	685	rpp3d
marek	S	2390	2383	/usr/bin/gnome-volch...
globus	S	17194	1	/bin/sh
root	S	17141	1	/usr/local/pbs/sbin/pb...
globus	S	22965	17194	sleep
globus	S	3697	3695	condor_schedd

```
<experiment>
 <applicationName>rpp3d</applicationName>
 <jobID>23458</jobID>
 <experimentID>blabla</experimentID>
</experiment>
<request name="instrument">
 <task>
  <coderegion unit="MAIN_" name="mymain_" id="f60_mymain_"/>
  <metrics>WTIME</metrics>
</task>
<task>
  <coderegion unit="mymain_" name="area_weighting_" id="f309_area_weight
   <metrics>WTIME</metrics>
</task>
</request>
</irl>
```

```
rpp3d
  SIR
    main(function,u_main)
    MAIN_(function,u_MAIN_)
    get_global_parameter_(function,u_get_global_parameter_)
    parallel_init_(function,u_parallel_init_)
    get_parameter_(function,u_get_parameter_)
    get_cluster_par_(function,u_get_cluster_par_)
    load_charge_states_(function,u_load_charge_states_)
    mymain_(function,u_mymain_)
      coderegion(call,f308_area_weighting_)
      coderegion(call,f309_area_weighting_)
      coderegion(call,f336_finish_job_)
      coderegion(call,f409_get_time_)
      coderegion(call,f536_save_file_exists_)
      coderegion(call,f543_pmpi_finalize_)
      coderegion(call,f566_.rem)
```

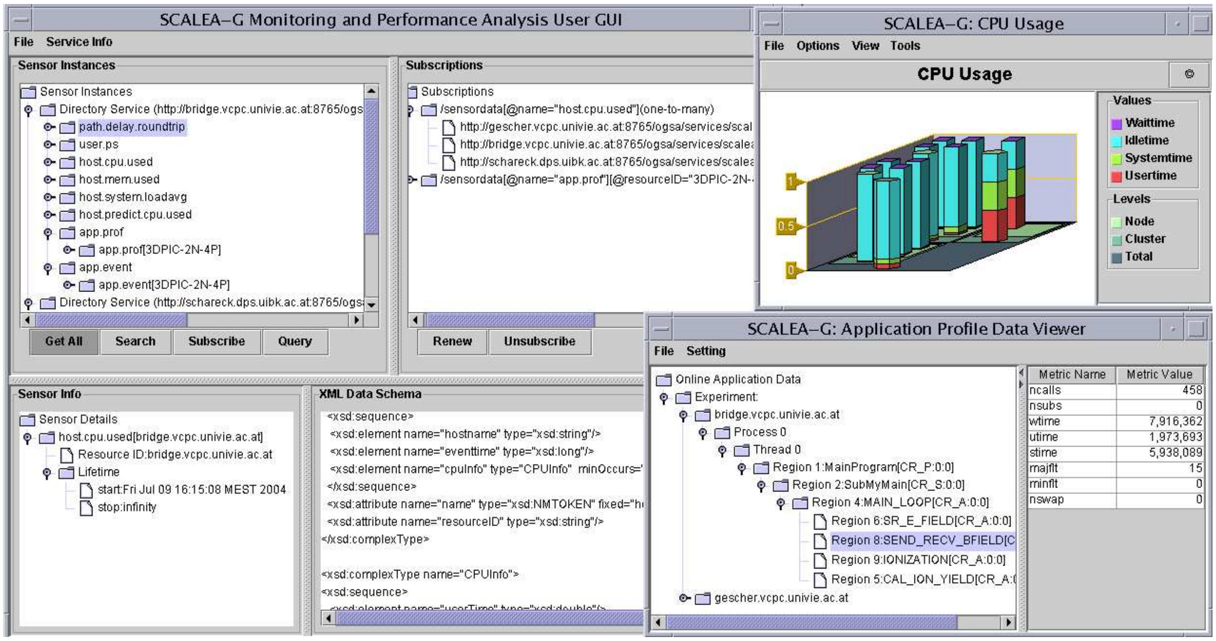
Figure 10: SCALEA-G Dynamic Instrumentation GUI.

Figure 10 displays the GUI for conducting the dynamic instrumentation in SCALEA-G. On the top-left window, the user can choose a DS and retrieve a list of instances of MS registered to that DS. The user can monitor processes running on compute nodes where instances of MS execute by calling `Get/Update User Processes` operation of MS (e.g. see the top-right window of Figure 10). For a given application process, its SIRBC can be obtained via *Get SIR* operation, e.g. the SIRBC of *rpp3d* process is visualized in the bottom-right window. In the bottom-left window, the user can edit IRL requests and send these requests to selected instances of MS.

7.3 Performance Monitoring and Analysis GUI

The user can monitor and analyze online performance behavior of Grid systems and applications at the same time. In the top-left window of Figure 11, the user can examine DSs and information about sensor instances registered with these DSs. Under the DS tree, sensor instances are grouped into categories based on the sensor identifier; each category provides the same data type but of different resources. For each sensor instance, the user can examine its properties (e.g. lifetime and XML schema) by choosing the instance from the list of sensor instances.

The user can perform one-to-one or one-to-many DQS by selecting sensor instance (for one-to-one mode) or sensor category (for one-to-many mode), choosing `Subscribe` or `Query` (see Figure 11) and then editing the request (e.g. subscription time, XML data filter) if needed. We can subscribe, query and then examine multiple types of monitoring data at the same time. For example, we subscribed application data of experiment `3DPIC-2N-4P` and data provided by `host.cpu.used` sensor in compute nodes on which the application processes execute, and then examined that data online (see *Application Profile Data Viewer* and *CPU Usage* window in Figure 11). List of existing subscriptions is shown in the `Subscriptions` tree. A subscription can be canceled by selecting the subscription and clicking `Unsubscribe`. Similarly, an existing subscription can be renewed by clicking `Renew`.



```
Sensor Instances
  Directory Service (http://bridge.vcpc.univie.ac.at:8765/ogs
    path.delay.roundtrip
    user.ps
    host.cpu.used
    host.mem.used
    host.system.loadavg
    host.predict.cpu.used
    app.prof
      app.prof[3DPIC-2N-4P]
    app.event
      app.event[3DPIC-2N-4P]
  Directory Service (http://schareck.dps.uibk.ac.at:8765/ogs
```

```
Subscriptions
  /sensordata[@name="host.cpu.used"](one-to-many)
    http://gescher.vcpc.univie.ac.at:8765/ogsa/services/scal
    http://bridge.vcpc.univie.ac.at:8765/ogsa/services/scalea
    http://schareck.dps.uibk.ac.at:8765/ogsa/services/scalea
  /sensordata[@name="app.prof"][@resourceID="3DPIC-2N-
```

```
Sensor Details
  host.cpu.used[bridge.vcpc.univie.ac.at]
    Resource ID:bridge.vcpc.univie.ac.at
    Lifetime
      start:Fri Jul 09 16:15:08 MEST 2004
      stop:infinity
```

```
  <xsd:sequence>
   <xsd:element name="hostname" type="xsd:string"/>
   <xsd:element name="eventtime" type="xsd:long"/>
   <xsd:element name="cpuInfo" type="CPUInfo"  minOccurs='
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:NMTOKEN" fixed="h
  <xsd:attribute name="resourceID" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="CPUInfo">
<xsd:sequence>
```

```
Online Application Data
  Experiment:
    bridge.vcpc.univie.ac.at
      Process 0
        Thread 0
          Region 1:MainProgram[CR_P:0:0]
            Region 2:SubMyMain[CR_S:0:0]
              Region 4:MAIN_LOOP[CR_A:0:0]
                Region 6:SR_E_FIELD[CR_A:0:0]
                Region 8:SEND_RECV_BFIELD[C
                Region 9:IONIZATION[CR_A:0:0]
                Region 5:CAL_ION_YIELD[CR_A:(
    gescher.vcpc.univie.ac.at
```

Metric Name	Metric Value
ncalls	458
nsubs	0
wtime	7,916,362
utime	1,973,693
stime	5,938,089
majflt	15
minflt	0
nswap	0

Figure 11: Performance Monitoring and Analysis GUI.

8 Related Work

Over the past few years, many Grid performance tools have been developed as cataloged in [17]. To assist developers in building monitoring tools for Grid system, Global Grid Forum (GGF) [19] has proposed the GMA [3] model which describes the major components of a Grid monitoring tool and their essential interactions.

Several existing tools are available for monitoring Grid computing resources and networks, such as MDS (a.k.a GRIS) [9], R-GMA (Relational GMA) [12], NWS [27], GridRM [4], Ganglia [23]. However, few monitoring and performance analysis tools for Grid applications have been introduced. GRM [5] is a semi-on-line monitor that collects information about an application running in a distributed heterogeneous system. In GRM, however, the instrumentation has to be done manually. OCM-G [6] is an monitoring infrastructure targeting to interactive Grid applications. Atop OCM-G, G-PM [7] is used to conduct the performance analysis. OCM-G combines dynamic and source instrumentation, however, currently limited to MPI functions.

None of aforementioned systems, except MDS, is an OGSA-based Grid service. Thus, it is rather difficult to deploy these tools in OGSA-based framework. Differing from most existing Grid monitoring tools, SCALEA-G is based on OGSA and supports both Grid infrastructure and application monitoring and performance analysis in a unified system, thus increasing the chance to correlate measurement metrics from various sources at different levels. Furthermore, existing tools employ a non-standard representation for monitored data. SCALEA-G, in contrast, uses widely-accepted XML for representing customizable performance data, and provides DQS mechanism with XPath-based requests.

Although there are well-known tools supporting dynamic instrumentation, e.g. Paradyn [24], DPCL [10], these tools are designed for conventional parallel systems rather than Grids. The requests for conducting the dynamic instrumentation in these tools are not widely accessible and highly interoperable. The lack of a well-defined, general purpose protocol like IRL has hindered other services from using these tools to conduct the dynamic instrumentation. Our work on developing IRL and general-purposed instrumentation service intendedly provides a generic means for a large number of other services to conduct dynamic instrumentation of interesting applications while hiding all low level details of instrumentation mechanism.

Recent work in APART has also proposed MIR (Monitoring and Instrumentation Request) [13] which supports to control instrumentation and request measurement data. MIR approach, however, it is not suitable for querying multiple types of data as we have to implement several wrappers to convert MIR to specific languages (e.g XPath) used to query against measurement data. Our IRL is employed only for instrumentation purpose. By using XPath to express requests for measurement data we can easily customize the request based on a data representation and directly use these requests to get measurement data. However instrumentation requests in MIR can be incorporated into IRL.

9 Conclusion and Future Work

In this paper we presented the architecture of SCALEA-G, a unified monitoring and performance analysis system for the Grid, based on the OGSA and GMA concept. We have described the architecture and presented some experiments. The main contributions of this paper center on the unique, monitoring and performance analysis system based on OGSA, the dynamic instrumentation service for Grid applications with the novel instrumentation request language (IRL) and standardized intermediate representation for binary code (SIRBC).

We plan to exploit peer-to-peer features for DQS in our services. The set of sensors will be extended to support monitoring based on resource model and rules, monitoring more resources and services, and providing more diverse types of data. We are currently working on an extension of SIRBC to cover loops and to describe workflow-based Grid applications. IRL will be extended to allow the specification of more complex instrumentation requests, e.g. deactivating and removing instrumentation. With the recently-announced support of WSRF [16] under Globus, we will port SCALEA-G to WSRF environment.

Acknowledgments

This research is partly supported by the Austrian Science Fund as part of the Aurora Project under contract SFBF1104 and by the IST APART Working Group on Automatic Performance Analysis under contract EP 29488.

References

[1] Postgresql 7.1.2. http://www.postgresql.org/docs/.

[2] Sleepcat Berkeley DB, http://www.sleepycat.com.

[3] B. Tierney et. al. A Grid Monitoring Architecture. Technical report, Performance Working Group, Grid Forum, January 2002. http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-2.pdf.

[4] Mark Baker and Garry Smith. GridRM: An Extensible Resource Management System. In *Proceedings of IEEE International Conference on Cluster Computing (CLUSTER'03)*, pages 207–215, Hong Kong, December 01-04 2003. IEEE Computer Society Press.

[5] Zoltan Balaton, Peter Kacsuk, Norbert Podhorszki, and Ferenc Vajda. From Cluster Monitoring to Grid Monitoring Based on GRM. In *Proceedings. 7th EuroPar'2001 Parallel Processings*, pages 874–881, Manchester, UK, 2001.

[6] Bartosz Balis, Marian Bubak, Włodzimierz Funika, Tomasz Szepieniec, and Roland Wismüller. An infrastructure for Grid application monitoring. *LNCS*, 2474:41–49, 2002.

[7] Marian Bubak, Włodzimierz Funika, and Roland Wismüller. The CrossGrid performance analysis tool for interactive Grid applications. *LNCS*, 2474:50–60, 2002.

[8] Bryan Buck and Jeffrey K. Hollingsworth. An API for Runtime Code Patching. *The International Journal of High Performance Computing Applications*, 14(4):317–329, Winter 2000.

[9] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*. IEEE Press, August 2001.

[10] L. DeRose, T. Hoover Jr., and J. Hollingsworth. The dynamic probe class library: An infrastucture for developing instrumentation for performance tools. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS-01)*, pages 66–66, Los Alamitos, CA, April 23–27 2001. IEEE Computer Society.

[11] Discovery and Monitoring Event Description (DAMED) Working Group. http://www-didc.lbl.gov/damed/.

[12] A. Cooke et al. R-GMA An Information Integration System for Grid Monitoring. In *Proceedings of 11th International Conference on Cooperative Information Systems (CoopIS 2003)*, Sicily,Italy, 3-7 November 2003.

[13] T. Fahringer, M. Gerndt, Bernd Mohr, Martin Schulz, Clovis Seragiotto, and Hong-Linh Truong. Monitoring and Instrumentation Requests for Fortran, Java, C and C++ Programs. APART Working group (http://www.kfa-juelich.de/apart/), Work in progress, August 2004.

[14] T. Fahringer, M. Gerndt, Bernd Mohr, Martin Schulz, Clovis Seragiotto, and Hong-Linh Truong. Standardized Intermediate Representation for Fortran, Java, C and C++ Programs. APART Working group (http://www.kfa-juelich.de/apart/), Work in progress, August 2004.

[15] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *IEEE Computer*, pages 37–46, June 2002.

[16] Ian Foster, Jeffrey Frey, Steve Graham, Steve Tuecke, Karl Czajkowski, Don Ferguson, Frank Leymann, Martin Nally, Tony Storey, William Vambenepe, and Sanjiva Weerawarana. Modeling Stateful Resources with Web Services. Specification, Globus Alliance, Argonne National Laboratory, IBM, USC ISI, Hewle tt-Packard, January 2004.

[17] M. Gerndt, R. Wismueller, Z. Balaton, G. Gombas, P. Kacsuk, Z. Nemeth, N. Podhorszki, H.-L. Truong, T. Fahringer, M. Bubak, E. Laure, and T. Margalef. Performance Tools for the Grid: State of the Art and Future. Technical report, Research Report Series, Lehrstuhl fuer Rechnertechnik und Rechnerorganisation (LRR-TUM) Technische Universitaet Muenchen, Vol. 30, Shaker Verlag, ISBN 3-8322-2413-0, 2004.

[18] GGF Network Measurements Working Group. http://forge.gridforum.org/projects/nm-wg/.

[19] Global Grid Forum. http://www.gridforum.org/.

[20] Globus Project. http://www.globus.org.

[21] gSOAP: C/C++ Web Services and Clients. http://www.cs.fsu.edu/˜engelen/soap.html.

[22] G. Laszewski, I. Foster, J. Gawor, and P. Lane. A java commodity grid kit. *Concurrency and Computation: Practice and Experience*, 13(643-662), 2001.

[23] Matthew L. Massie, Brent N. Chun, and David E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, May 2004.

[24] Paradyn Parallel Performance Tools. http://www.cs.wisc.edu/paradyn/.

[25] Hong-Linh Truong and Thomas Fahringer. SCALEA: A Performance Analysis Tool for Parallel Programs. *Concurrency and Computation: Practice and Experience*, 15(11-12):1001–1025, 2003.

[26] Von Welch, Frank Siebenlist, Ian Foster, John Bresnahan, Karl Czajkowski, Jarek Gawor, Carl Kesselman, Sam Meder, Laura Pearlman, and Steven Tuecke. Security for Grid Services. In *12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, pages 48–57, Seattle, Washington, June 22 - 24 2003.

[27] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computing Systems*, 15:757–768, 1999.

[28] XML Path Language. http://www.w3.org/tr/xpath.html.

[29] XML Schema. http://www.w3.org/xml/schema.