

REPORTING GRID SERVICES (REGS) SPECIFICATION

draft-ggf-ogsa-regs-0.3.1

Yariv Aridor, Dean Lorenz, Benny Rochwerger
IBM Haifa Research Lab.

Bill Horn
IBM T. J. Watson Research Center

Hany Salem
IBM Software Group

Abstract

The *Reporting Grid Services (ReGS)* system is designed to be a set of core OGSA services for logging, tracing and monitoring applications in a distributed, heterogeneous computing environment. It provides OGSA style logging interfaces for use by other grid services and applications. It is also capable of virtualizing existing logging systems including zOS logging, NT events and syslog.

Keywords

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Contents

1	Introduction	3
1.1	An Architecture for Distributed Reporting	3
1.2	Definitions	7
1.2.1	Acronyms	7
1.2.2	Terms	7
1.2.3	Namespaces	9
1.3	Document Organization	9
2	Filtering Grid Service	11
2.1	Information Flow	11
2.2	<i>NotificationSink</i> portType	11
2.2.1	<i>RegsReport</i>	11
2.3	<i>NotificationSource</i> portType	13
2.3.1	<i>NotificationSource</i> Notification Message	13
2.3.2	<i>NotificationSource</i> Subscription Expression	14
2.3.3	<i>NotificationSource</i> Subscription Expression Types	15
2.4	<i>Factory</i> portType	16
2.4.1	<i>Factory</i> Service Parameters	16
2.4.2	<i>Factory</i> Creation Input Types	16
2.5	<i>Registration</i> portType	17
2.6	Basket Sharing (TBD)	17
2.7	Non-native Baskets	18
2.8	Interaction with Producers	18
2.8.1	Producer Discovery	19
2.8.2	Changing a Producer's Reporting Granularity	19
3	Basket Grid Service	20
3.1	Basket as a Multi-Consumer Service	20
3.2	Lifetime of the Basket Instance	20
3.3	ServiceData Elements	20
3.3.1	<i>BasketServiceReports</i>	20
3.3.2	<i>queryBySimpleFilter</i>	21
3.3.3	<i>BasketProfile</i>	22
3.3.4	<i>BasketCurrentState</i>	23
3.3.5	<i>subscribeByReports</i>	24
3.4	<i>subscribeByBasketStatus</i>	24
3.5	Basket PortType: Operations and Messages	25
3.5.1	<i>Basket::performActionsOnReports</i>	25
4	Acknowledgments	26
A	Schemas	29
A.1	<i>regs:report</i> Schema	29
A.2	<i>regs:filter</i> Schema	32
A.3	<i>regs:basketFactory</i> Schema	34
A.4	<i>regs:BasketServiceReports</i> Schema	35

A.5	regs:BasketProfile Schema	36
A.6	regs:BasketCurrentState Schema	37
A.7	regs:BasketActions Schema	38
A.8	regs:ReportSet Schema	39
A.9	regs:SubscribeByReports Schema	40
A.10	regs:SubscribeByBasketStatus Schema	41
A.11	regs:RepositoryFullMessage Schema	42
A.12	regs:ReportDiscardedMessage Schema	43

1 Introduction

The *Open Grid Services Architecture (OGSA)* [7, 6, 24] is an attempt to bring together the worlds of Web services [9] and grid computing [5]. The Web services architecture for distributed computing is based on using platform independent, widely adopted open standards such as XML[26] and SOAP[27]. This architecture addresses some of the basic issues of heterogenous distributed computing, including standard interface definition mechanism, multiple protocol bindings, and local/remote transparency. Grid computing encompasses the set of protocols and tools needed for large scale resource sharing in dynamic, multi-institutional *virtual organizations*¹ [8]. The grid protocols provide good technical solutions for key problems in distributed computing (e.g., authentication, delegation, resource discovery, reliable invocation, etc.); however, the grid tools currently available (in the Globus Toolkit [3, 4]) are not easy to incorporate into other systems and there is little reuse of components across projects.

OGSA is a service-oriented approach to virtualize and share resources. It starts with the Web services architecture and adds to the model concepts taken from grid computing such as lifecycle managements, discovery and the grid security model. A *grid service* is basically a Web service that supports a minimal set of required interfaces and well-defined behaviors.

The *Reporting Grid Services (ReGS)* system is designed to be a set of core OGSA services for logging, tracing, and monitoring applications in OGSA-based grid environments. It provides OGSA style logging interfaces for use by other grid services and applications. It is also capable of virtualizing existing logging systems, including syslog [14, 16], NT events [15], and zOS logging [13].

1.1 An Architecture for Distributed Reporting

The *Grid Monitoring Architecture (GMA)* specification [22] identifies the three basic components of a distributed monitoring system: producers, consumers, and a directory where consumers can find information about producers. In addition, the GMA proposes the use of intermediaries or compound components, which function both as producers and consumers, for building advanced services. Popular logging packages such as Apache's `log4j` [10] and the proposed Java Logging APIs (JSR-047) [12] clearly identified the need for a sophisticated filtering mechanism and for a variety of storage methods. The *Reporting Grid Services (ReGS)* architecture brings ideas from these packages into a GMA-inspired architecture by defining the interfaces and behavior of two types of intermediaries: a *filtering* component and a *storage and delivery* component.

In a typical reporting² scenario, an application (*the message producer*) generates data that may or may not be used at a later time by another application (*the message consumer*). In most cases, the amount of data generated is very large, while the amount of data actually consumed could be relatively small. Therefore, it is desirable to have a mechanism to control the amount of data generated and to filter out data that is actually kept (*the filtering service*). Finally, different types of data may have different durability and consumption characteristics. For example, while some data becomes irrelevant very fast, but is needed as soon as it is generated (e.g., real-time monitoring data), some other data may be needed even months after it was generated (e.g., auditing data). Hence, there is a need for a mechanism to create different repositories of data (*baskets*), each with its own behavioral characteristics (*the storage and delivery service, or basket service*). The components that make up this scenario, as well as the basic interactions between them, are shown in Figure 1. In the ReGS architecture, standard OGSA-like interfaces are defined for each component;

¹Virtual Organizations, or VOs, are defined as a set of resources, possibly spanning across locations and administrative domains, and cooperating in a common computational task.

²In this document we used the term "reporting" to encompass logging, tracing, and monitoring; although the characteristics of the data generated may differ, the basic infrastructure is the same.

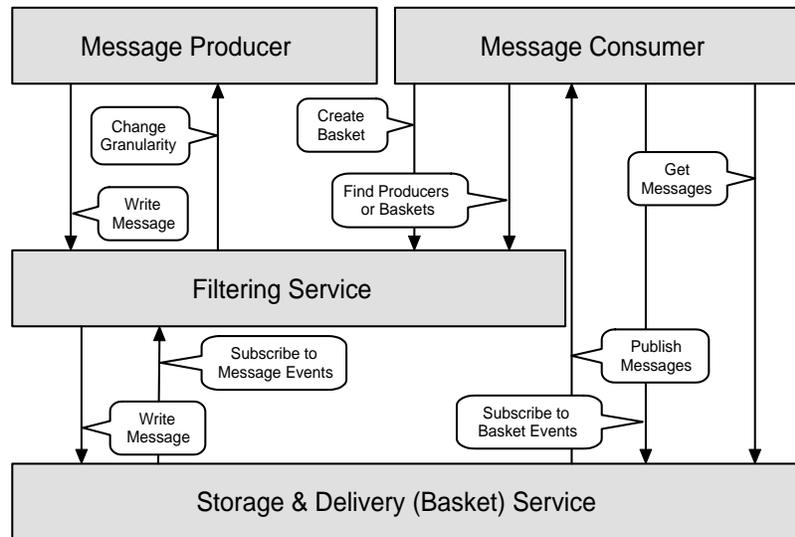


Figure 1: Components of a reporting system and their basic interactions

when appropriate, ReGS maps the different interactions present in the model to the existing operations and portTypes (see Figure 2).

The idea that the functionality of *data baskets* can be easily tailored to meet the needs of consumers is fundamental to the ReGS architecture. What goes into a basket is determined by a *filtering rule*; how and when messages are delivered is determined by a *notification policy*; and how long messages stay in the basket is determined by a *deletion policy* (see Figure 3).

Synchronous message consumption is supported by the standard mechanism for querying service data from a grid service (the `GridService::findServiceData` operation in the `GridService` portType). Asynchronous message consumption is supported by the standard mechanism for notification delivery (the `deliverNotification` operation in the `NotificationSink` portType). Based on these consumption models and the basket policies, all three *interaction models* defined by the GMA are easily supported:

Publish/subscribe The “subscription creation” procedure is achieved in our model by creating a new basket. To achieve the publish/subscribe interaction, the notification policy is set to deliver every message upon arrival to the consumer.

Query/response This is the typical ReGS “polling” interaction, where consumers retrieve from the basket all the messages that fit a particular query. The matching messages are packed together and sent to the consumer in a single response.

Notification The notification policy of the basket is set so when “enough” messages have arrived, they are packed and delivered to the consumer in a single notification.

Although most baskets are created and controlled by consumers, any system should provide at least some basic reporting capabilities. For this purpose, two basic baskets are defined:

1. The *Recycle Basket* is the default repository for all messages generated. It serves as a place holder for all recent messages, allowing consumers to access data that was generated before any customized

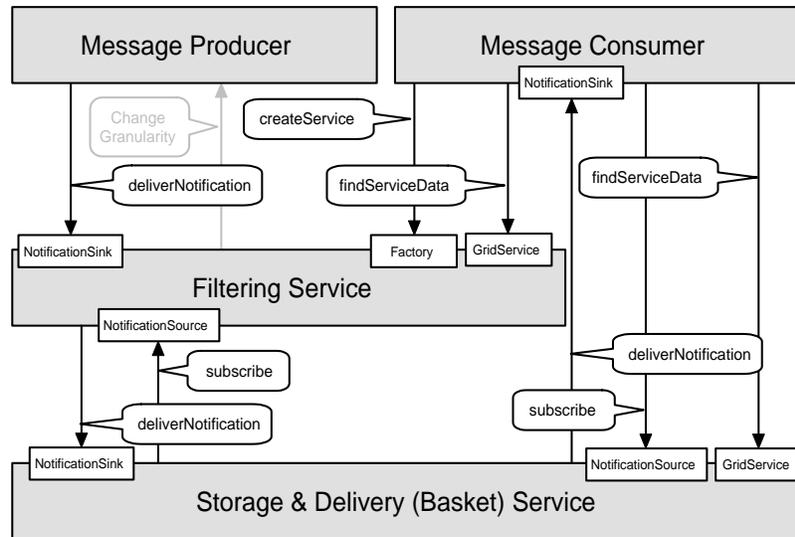


Figure 2: OGSA-based reporting: Most interactions between components can be mapped into the operations defined by the standard OGSI portTypes. New portTypes and operations are limited to those interactions that do not naturally map into the standard (grayed out in this figure).

basket was created. By properly setting the deletion and notification policies on this basket, the `logrotate` [17] functionality³ can be easily replicated.

2. The *System Basket* wraps platform specific logging systems and functions as a basket for system-related events. This serves two purposes: first, it provides a “standard” mechanism to expose event data generated by legacy producers; second, new producers that follow the ReGS interfaces can put data in the platform-specific repository for system events (see Figure 4).

In a heterogeneous distributed environment there are many different deployment possibilities for the different components described above. In the simplest case, all components are co-located in the same machine, while in the most general case, each component is located on a different machine. By defining the interactions between components in terms of grid services, these cases, and all the ones in between, can be easily supported and leave the details of how data is transferred to the binding stage at the Web services layer. The ReGS architecture is flexible enough to allow all possible combinations; however, the scenario that we think will be the most common is where each hosting environment, or container, will house a filtering service that will receive data from local applications and distribute it both to local and possible remote baskets. Consumers (most likely remote) will use the filtering service to create custom baskets that capture the data that is of interest to them (see Figure 5). The consumer that creates the basket will decide the basket’s location; in addition to creating baskets, consumers can register existing basket with remote filtering services.

The ReGS architecture differs from existing distributed logging mechanisms in that it focuses on the definition of the interfaces between the different components. Each of these systems can be easily mapped into the ReGS model by having their components (or at least some of them) wrapped with the ReGS OGSA-based interfaces. For example, in a system using distributed `syslog`, at each node the local `syslog` daemon can be wrapped with the filtering service interfaces. This will give OGSA-based consumers the

³Automatic rotation, compression, removal, and mailing of log files. Each log file may be handled daily, weekly, monthly, or when it grows too large.

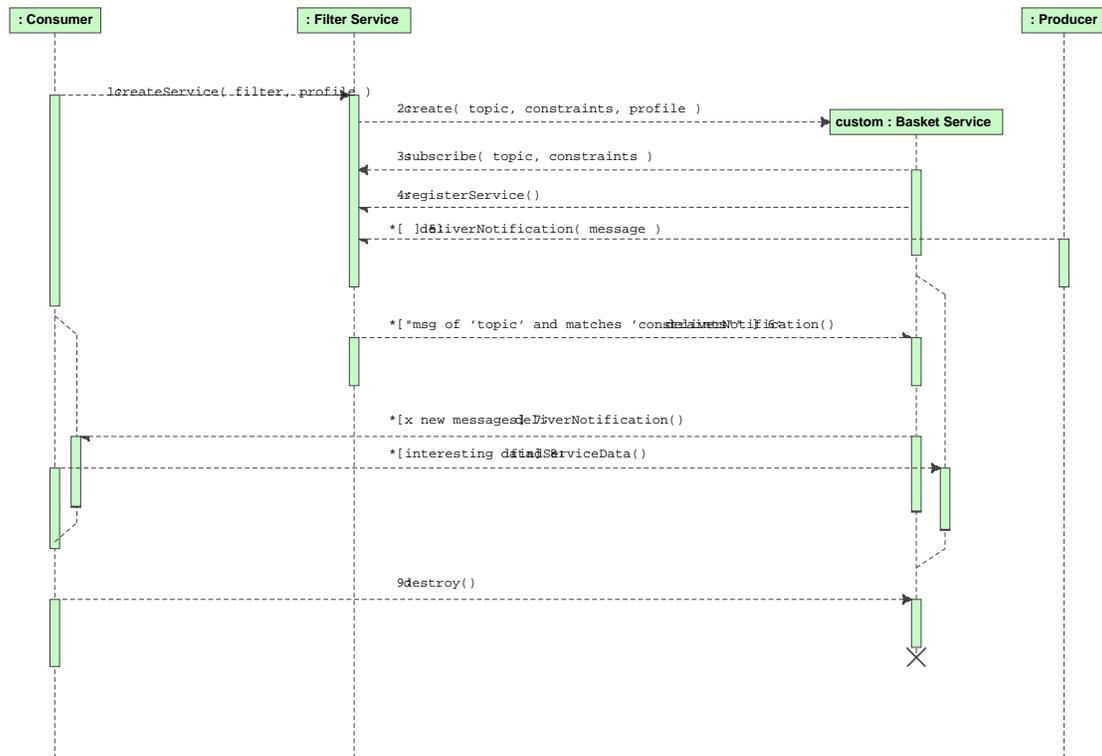


Figure 3: The lifecycle of a custom basket: A consumer requests the creation of a custom basket from the filtering service by specifying the characteristics of the basket (1). In response, the filtering service creates a new instance of the basket service (2,3,4). When messages arrive at the filtering service from producers, they are compared with the filtering rule of the basket. If there is a match, the message is delivered to the basket service (5,6). The basket service can send digests of messages back to the consumer (7), or wait for the consumer to retrieve the desired messages (8). When the consumer no longer needs the basket, it disposes of it (9).

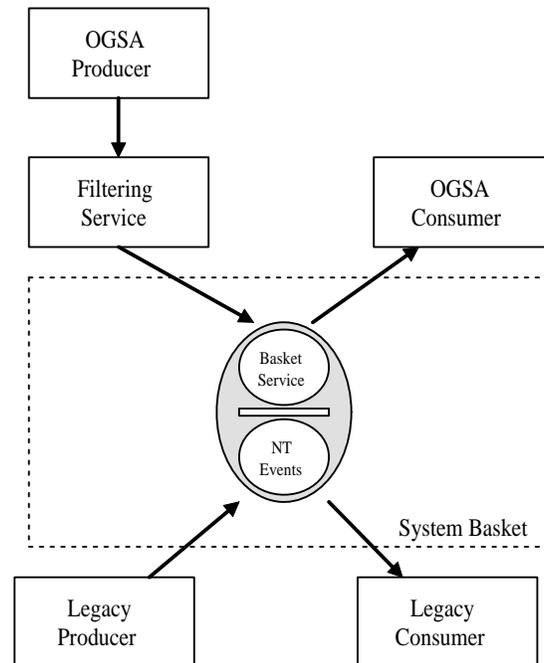


Figure 4: The System Basket virtualizes the platform-specific logging mechanism (e.g., NT Events): using the system basket OGSA-enabled consumers can get data generated by legacy consumers, and OGSA enable producers can write data into the platform-specific repository of logging information.

ability to control what gets sent to the `syslog` message collector. The collector itself can be wrapped with the basket service interfaces, so the OGSA-based consumer can query the data and/or get notifications (see Figure 6). Note, in this example, the flow of data from the original source to the message collector is not “OGSified”. The ReGS model can coexist with other systems by wrapping only the parts that need to be exposed. Similarly, popular grid monitoring systems such as JAMM [23], MDS-2 [2], NWS [29], and NetLogger [11] can be fully or partially mapped to the ReGS model.

1.2 Definitions

1.2.1 Acronyms

The following is a list of the acronyms used in this document:

GMA Grid Monitoring Architecture

OGSA Open Grid Services Architecture

OGSI Open Grid Services Infrastructure

WSDL Web Services Description Language

SDE Service Data Element

1.2.2 Terms

The following is a list of the acronyms used in this document, along with their definitions:

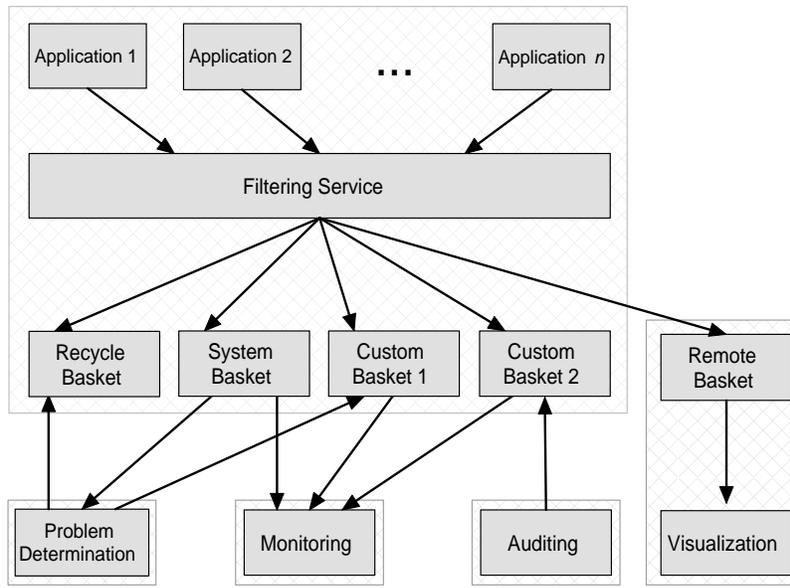


Figure 5: Putting it all together: Applications running within a single container deliver their data to the local filtering service, which distributes it to the relevant local and remote baskets (co-located processes are enclosed by the shaded boxes). In this example, producers running on the same host produce data that is filtered by the local filtering service; each consumer runs on a different machine and one of them (the Visualization consumer) has a basket co-located with the consumer application. The arrows point from service requestors to providers; consumers might pull data (e.g., the Auditing consumer), wait for data to be pushed to them (e.g., the Monitoring consumer), or do both (e.g., the Problem Determination consumer).

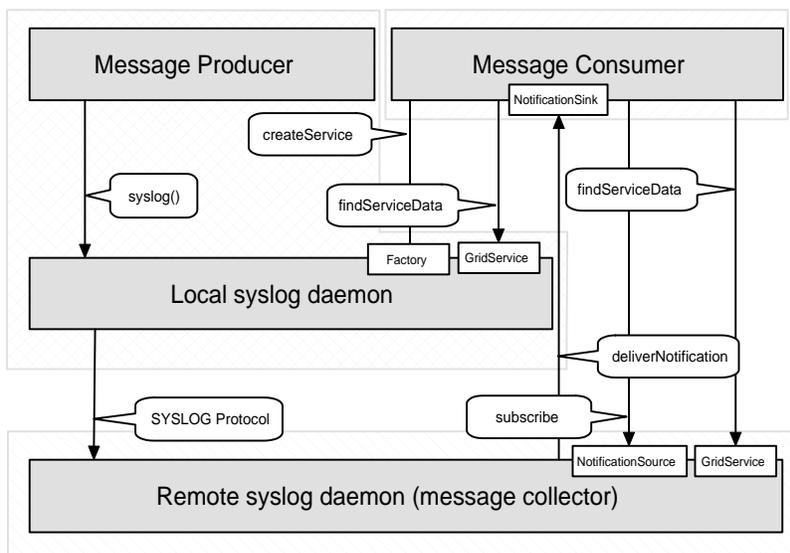


Figure 6: Wrapping syslog: The syslog daemon on each node is wrapped with the filtering service interfaces so OGSA-based consumers have control over what is delivered to the syslog message collector. The message collector is wrapped with the basket service interfaces, so the OGSA-based consumers can query the data and/or get notifications. In this example, the flow of data from the original source to the message collector is not OGSified

portType defines an interface as a set of related operations

serviceType list of portTypes; enables aggregation

serviceImplementation representation of the actual code

service instanceOf extension. Maps description to instance

1.2.3 Namespaces

The following Namespaces are used throughout this document:

Prefix	Namespace
regs:regs	regs:http://hrl.ibm.com/ogsa/schema/regs
regs:gsdl	regs:http://www.gridforum.org/namespaces/2002/07/gridServices
regs:xsd	regs:http://www.w3.org/2001/XMLSchema
regs:xsi	regs:http://www.w3.org/2001/XMLSchema-instance

1.3 Document Organization

The rest of this document is organized as follows: Section 2 presents details for the filtering service, and Section 3 presents details for the Basket Service. The definitions of the different *service data elements* [24] are given by the XML schemas for the corresponding extensibility elements section. To simplify reading, the schemas are represented with diagrams following the notation used by the TurboXML schema editor (see Table 1 for an overview of the notation). The complete schemas are included in Appendix A at the end of the document. Throughout the document, we use the `regs:` prefix to indicate XML type and elements taken from the `regs:http://hrl.ibm.com/ogsa/schema/regs` namespace, which is the target namespace for all our schemas.

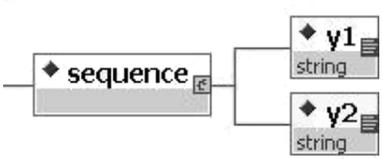
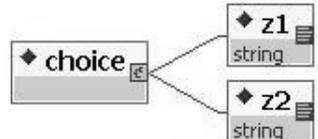
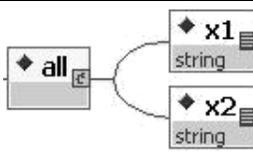
	xsd:element
	minOccurs="0"
	maxOccurs="unbounded"
	minOccurs="1" maxOccurs="5"
	xsd:sequence
	xsd:choice
	xsd:all
	xsd:attribute

Table 1: TurboXML graphical representation of XML schemas

2 Filtering Grid Service

The main function of the filtering grid service is to filter and distribute report messages. The filtering service receives messages from report message producers, examines the message content, and “puts” the message in the appropriate baskets.

The filtering service acts as a single access point by which consumers can discover which producers exist and which types of messages are produced and are available for consumption. It also allows consumers to discover existing baskets and to create new baskets when needed.

The filtering grid service does *not* implement any new portTypes. It implements the *GridService*, *NotificationSink*, *NotificationSource*, *Factory*, and *Registration* portTypes.

2.1 Information Flow

Producers send report messages to the filtering service. The filtering service acts as a *notification intermediary* [24] and republishes the messages. Each basket is subscribed to the filtering service using a filter rule. Only messages that match the basket’s rule are forwarded to the basket.

Consumers use the filtering service to discover which producers publish report messages and what is the format of those messages. They use this information to define new (potentially producer-specific) filter rules.

The filtering service acts as a basket *factory* that can be used by consumers to create new baskets with specific filter rules and specific profiles (see Section 3).

The filtering service is a *registry* for baskets. Consumers can find out information about existing baskets, and may choose either to use an existing basket or create a new one.

2.2 NotificationSink portType

Producers send *report* messages to the filtering service. The filtering service **MUST** be a *NotificationSink* and is able to receive report messages through the *NotificationSink::deliverNotification* operation. Each message includes a single report record, which **MUST** be sent as a serviceData element named *RegsReport*. The RegsReport SDE is described in the following section.

2.2.1 RegsReport

```
<gsdl:serviceDataDescription name="RegsReport"
  type="regs:report"
  minOccurs="1"          maxOccurs="Exactly one"
  xmlns:regs="http://hrl.ibm.com/ogsa/schema/regs/"
  xmlns:gsdl="http://schemas.gridforum.org/gridServices/">
  <wsdl:documentation>
```

An XML `regs:report` element that contains the report data.

```
</wsdl:documentation>
</gsdl:serviceDataDescription>
```

The schema is shown in Figure 7

Producers may include any data in a report message, however, the message must be an XML document that is a valid instance of the `regs:report` type (see Appendix A.1 for the complete schema).

Each report is divided into sections. Each section has a set of properties and a body. The attributes of the section provide meta information for the section as follows.

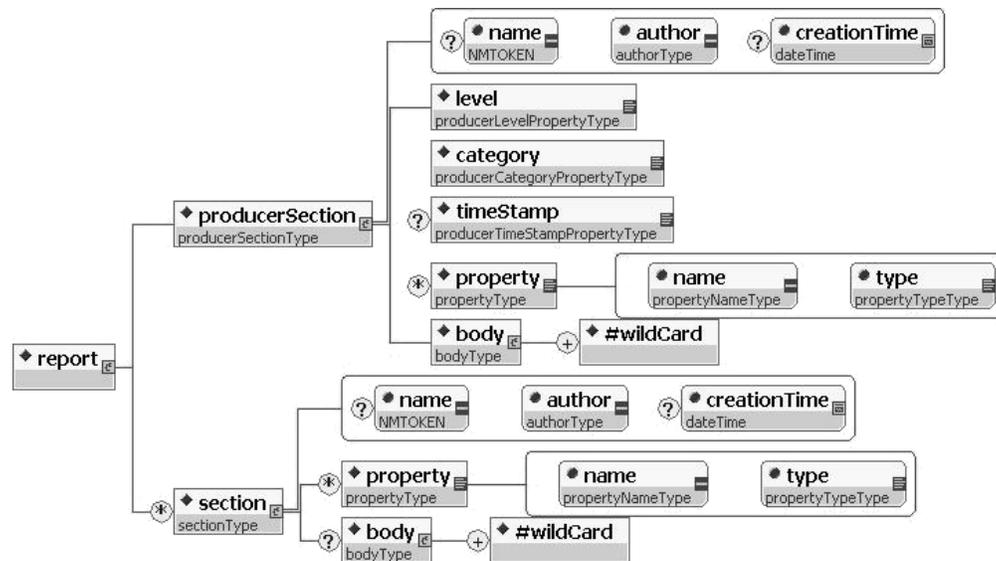


Figure 7: Schema for SDE RegsReport

author section creator; it MAY be a GSH.

name unique identifier; each section must be uniquely identifiable via author and name.

creationTime time at which the section created.

Issue 1 *Should we have std. lifetime attributes as well? That is, some producers may want to include lifetime declaration properties for each report they create. Should we add these properties as optional attributes to each section (as recommended in the GS specification section 4.4.4)? If we do, should we define the meaning of each attribute in our context (e.g., what does good until mean for a report)?*

The properties in each report section provide formatted fields by which a message may be filtered. Each property is a simple-content XML element with a name, type, and value as defined by the `regs:-propertyType`. The name and type attributes of the property element describe its value (the XML contents). The author of the section may choose any name for the property and MUST provide a type that matches the property value. The type is a string that should be a name of any simple type, as defined by XML schema specification [28].

The body of the section may include any well-formed XML data. The filtering service MAY treat this as opaque data, but it is RECOMMENDED that filtering services support application-specific filtering based on body content.

There is a special *producerSection* that MUST appear in every message. This section is similar to any other section, but includes predefined properties, which MUST be set by the producer. All the predefined properties of this header section have a name with a prefix “REGS_”. This prefix may not appear as the name of a property in any other section. The author property of the producerSection must be set to a QName for the producer. There are three properties that MUST be supported:

level a positive integer that defines the severity level of the message. The constants ALL, DEBUG, INFO, WARN, ERROR, and FATAL may be used to indicate severity levels of 0, 10, 20, 30, 40, and 50, respectively.

category this indicates a producer-specific, directory-like named hierarchy for the message (e.g., PACKAGE/MODULE/SUBMODULE). The category attribute is similar to a log4j or a JSR47 logger name, and allows the selection of messages on their location in the hierarchy.

timeStamp this is an optional property that provides a global timeStamp for the message.

The XML elements for these properties are explicitly named `level`, `category`, and `timeStamp`. However, to simplify message selection, their element type is the same as any other property, namely `regs:propertyType`. The name attribute of these properties are the same as their element name with an additional “REGS-” prefix (i.e., `REGS_level`, `REGS_category` `REGS_timeStamp`).

Issue 2 *The new spec implies that NotificationSinks can only receive messages from NotificationSources to which they are subscribed. Here we have an argument for allowing message reception via NotificationSinks without subscription. Do we need any modifications to the current spec of NotificationSink portType (add msg type, sender, etc.)? The alternative is to define our own mechanism (a new portType).*

Issue 3 *According to the new spec, a single SDE may contain multiple XML elements as values. This means that the same SDE may be used to send a single report or multiple reports. The basket service stores all reports in a single SDE, and sends query answers of multiple reports. Should we use just one SDE type for all (producer reports, basket storage, basket answers)? Should we allow producers to send more than one report in a message?*

2.3 *NotificationSource portType*

The filtering service acts as a *notification intermediary* and MUST implement the *NotificationSource portType*.

The filtering service is an intermediary for all the reports created by the producers. Subscribers may subscribe to receive notifications on the *RegsReport* SDE which, at any given moment, stores the most recent report received by the filtering service. The *NotificationSource portType* provides a mechanism to select reports based on their content. The baskets are subscribed to receive notification on all reports that match their filter rule.

2.3.1 *NotificationSource Notification Message*

The outgoing XML messages have the same format as the incoming reports. That is, they are a *RegsReport* serviceData elements as described above (Section 2.2).

Although the format of the messages is the same, the message content is not. The filtering service SHOULD add at least one section to the report, which would include properties that are added to every message. It should identify itself in the `author` attribute and indicate when the report was processed in the `creationTime` attribute. It is RECOMMENDED that the filtering service add *ID* and *ReportSize* attributes.

The filtering is performed on the *outgoing* message content. That is, subscribers can select messages based on properties (or any other content) that are added by the filtering service.

Issue 4 *Where and how does the filtering service describe the properties it adds?*

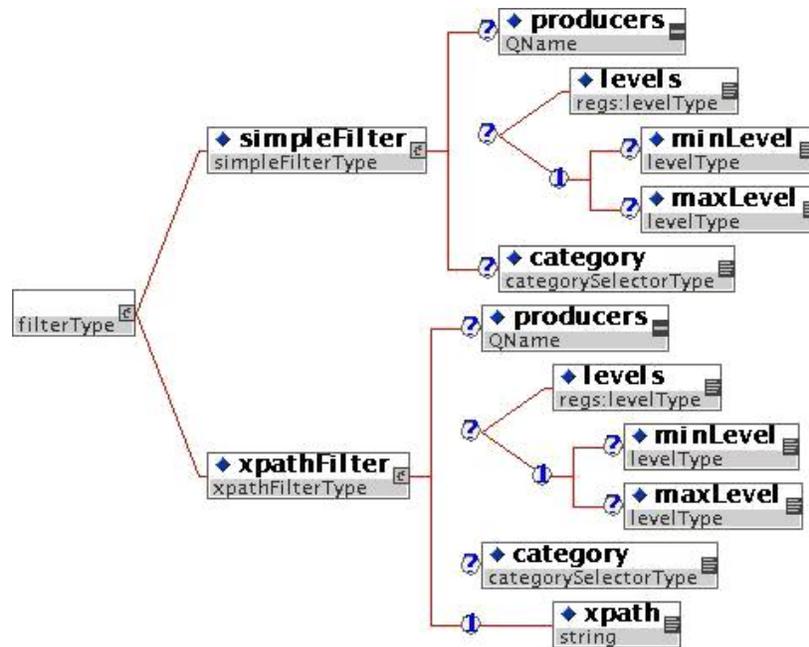


Figure 8: XML type `regs::filterType` is used to define the *SubscriptionExpression* for the *NotificationSource* portType

2.3.2 NotificationSource Subscription Expression

The subscription expression allows content-based report message selection. It defines a filter rule for the subscriber. All reports that match the specified rule are immediately forwarded to the subscriber.

The filtering service **MUST** support filtering based on the property fields of the report message. It is **RECOMMENDED** that it also support filtering based on application-specific fields that are included in the body of the report message.

There are two *SubscriptionExpressionTypes* that **MUST** be supported by the filtering service. Both are described using an XML element with type `regs:filterType` (Figure 8).⁴ The `regs:filterType` (see Appendix A.2 for the full schema) includes either a `regs:simpleFilter` element or a `regs:xpathFilter` element, and may be extended to support more filters.

regs:simpleFilter this filter supports the selection of messages based on their producer, level, and category.

It **MUST** be supported by the filtering service. This filter is focused on the producer section and examines the `author` attribute, and the `level` and `category` properties.

Producer selection is specified using the `producers` XML element, which contains a list of producer QNames.

Levels can be selected either by giving a list of levels using the `levels` element, or by a min–max range using the `minLevel` and `maxLevel` elements.

The category field is treated as a directory hierarchy with the producer’s QName added as the parent of the branch. For example a report from a producer named “P” with a category A/B will match the branch P/A/B. This allows for the hierarchical selection of messages from several producers. Subscribers may select any (sub-)branches of the hierarchy using a restricted XPath-like expression.

⁴The same filter XML syntax is used to perform queries on the basket service (see Section 3.3.2).

This works similar to the `selector` field of the XML schema unique element [28]. The `category` element may contain a list of one or more category selectors separated by a vertical bar (“|”). The category selector can be specified using an asterisk (“*”) to mean any name and “//” to indicate any descendant.

For example `*//` matches all categories from every producer; this has the same effect as not including the `category` element in the expression. The category `P//REPORT/*/LOG` matches `P/REPORT/C/LOG`, `P/A/REPORT/B/LOG`, and `P/A/B/REPORT/C/LOG`, but does not match `OTHER-P/REPORT/A/LOG`, `P/REPORT/A/B/LOG`, `P/REPORT/A/LOG/B`, and `P/A/B/REPORT/C/LOGGER`.

regs:xpathFilter this filter is an extension of the `regs:simpleFilter` and MUST be supported by the filtering service. It allows for selection using an XPath expression on the entire report. The XPath is used as a predicate (qualifier); namely, the message is filtered out if the expression evaluates to FALSE or is empty. The root of the XPath expression is the `regs:report` element. The filtering service MAY limit the XPath expression only to properties and SHOULD have higher performance for such queries. It is RECOMMENDED to also allow more general XPath expressions that examine the body elements.

The expression may include all the XML elements of the `regs:simpleFilter` plus an `xpath` element.

Issue 5 *An alternative to filtering by using SubscriptionExpression is to create a new SDE for each filter rule and to subscribe to all updates of this SDE. For example, when a basket is created, the filtering service can create an SDE that contains the most recent report that matches the basket’s filter rule. The basket can then be subscribed to using the default subscribeByServiceDataName SubscriptionExpression-Type.⁵ The advantage of this approach is that it makes it easy for new subscribers to discover and use existing filtering rules. The advantage of using a SubscriptionExpression is that it specifies the filter rule in the “natural” place, per subscription. Also, it provides the mechanisms by which subscribers can discover how a filter rule should be specified and the semantics for defining it during the subscription. In this case, the SubscriptionExpression is made available through the subscription instance serviceData.*

Issue 6 *Should we RECOMMEND a mapping of these selection rules onto existing messaging intermediaries (e.g., JMS)? For instance, we could recommend that the producer+category field become the topic of the message.*

2.3.3 NotificationSource Subscription Expression Types

The filtering service MUST include the following initial serviceData Value elements:

```
<gsdl:serviceData
  name="gsdl:SubscriptionExpressionTypes">
  <xsd:anyURI>
    http://hrl.ibm.com/ogsa/schema/regs/queryBySimpleFilter
    http://hrl.ibm.com/ogsa/schema/regs/queryByXPathFilter
  </xsd:anyURI>
</gsdl:serviceData>
```

These correspond to the `regs:simpleFilter` and `regs:xpathFilter` elements described above.

Issue 7 *These SubscriptionExpressionTypes are only allowed for the RegsReport SDE. The new spec does not provide an easy way to define this restriction.*

⁵This is equivalent to creating a topic for each basket and using the old `NotificationSourceTopic` portType.

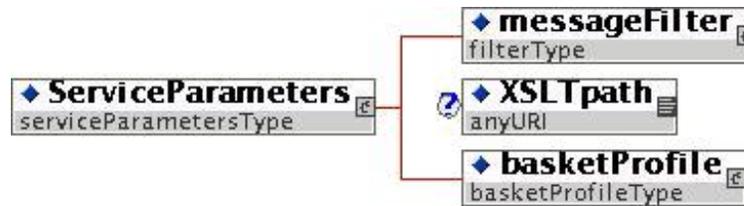


Figure 9: XML type `regs:serviceParameters` is used to define the *ServiceParameters* of the *Factory* portType

2.4 Factory portType

The filtering service acts as a basket factory. Baskets are created by invoking the *Factory::createService* operation at the *Factory* portType. As described in Section 3, at creation each basket is associated with a filter rule and a basket profile. Consumers provide this association through the `serviceParameters` input argument of the *createService* operation. The created basket is automatically subscribed to the filtering service using the *NotificationSource* portType (see Section 2.3), and its profile is set.

2.4.1 Factory Service Parameters

As described above, the *ServiceParameters* argument of of the *Factory::createService* operation defines the filtering rule and profile of the created basket. The *ServiceParameters* is an XML document with type `regs:serviceParameters` as shown in Figure 8 (see Appendix A.3 for the full schema).

The `regs:serviceParameters` XML element includes a `regs:messageFilter` element with a `regs:filterType` type, and a `regs:basketProfile` element with a `regs:basketProfile` type. The created basket is automatically subscribed to the filtering service via the *NotificationSource* portType (see Section 2.3) with the `regs:messageFilter` used as *SubscriptionExpression*. The format of the filter rule is as described above (Section 2.3.2). The basket is created with the specified profile (see Section 3 for details) and is automatically registered (i.e., is available through the *Registration* portType, see Section 2.5).

The `regs:serviceParameters` XML element MAY include an optional `XSLTpath` element, which provides a URI for an XSLT [25] document to be applied on any report that passes the filter *before* it is forwarded to the basket. This allows a consumer to store only relevant information in the basket, instead of the entire original message. If the filtering service supports this option, it acts as a standard XSLT engine. There is no restriction on what manipulation may be performed, except that the resulting document must be a valid XML instance of the `regs:report` schema. This mechanism may also be used to add consumer-specific data to the report (e.g., correlation data). The data SHOULD be added in a separate `section` element with proper attribute values.

2.4.2 Factory Creation Input Types

The filtering service MUST include the following initial `serviceData` Value elements:

```

<gsdl:serviceData
  name="gsdl:CreationInputTypes">
  <xsd:qname>
    http://hrl.ibm.com/ogsa/schema/regs/ServiceParameters
  </xsd:qname>
</gsdl:serviceData>
  
```

Issue 8 *What type of lifetime management do we have for baskets? Typical basket usage is longterm during which the creator (consumer) may be disconnected. Does it make sense to require the consumer to periodically refresh the basket or else risk losing the log data when the basket service expires? This issue is general and should be dealt within the GS spec.*

TBD Describe the container baskets and their lifetime management.

Issue 9 *Who creates the container basket?*

2.5 **Registration portType**

The filtering service is also a registry for baskets and **MUST** implement the *Registration portType* as defined in the grid service Specification. Each basket that is created by the filtering service is automatically registered. The registry provides a handle to each basket and the `regs:serviceParameters` XML element used for its creation. As for any Registration grid service, it can be searched for baskets with specific filtering rules or with other desired characteristics.

Issue 10 *Do we need to define a WS-Inspection document format for the baskets? Is this the right format to describe several instances of the same service? What is the minimal WS-Inspection document?*

Issue 11 *Are all baskets public? Can any consumer discover any basket? If not, should consumers specify at creation time if a basket is public or private?*

2.6 **Basket Sharing (TBD)**

Different consumers may request baskets with identical filter rules. It is desirable that the filtering service does not create several identical baskets, but rather that consumers share a basket.

There are several levels of basket sharing:

1. A consumer uses an existing basket in a “passive” mode, with no control over the basket’s content, and without the basket being aware of this consumer. The creator of the basket defines the filter rule and the policy for the basket and is the only one allowed to perform actions such as deleting messages. The passive user may only issue queries to retrieve reports from the basket.
2. A consumer may have no control over the content, but may be allowed to subscribe to basket notifications. This is the same as the previous mode, but the consumer may subscribe to receive notifications from the basket. There may be a single notification policy, namely all subscribers of the basket get exactly the same notifications, or independent subscriptions may be allowed.
3. A basket may share only the filtering rule, but give each consumer full management of the reports. In this case, the basket service gives each consumer the appearance of a separate basket instance (and may share data between those instances), while the filtering service forwards reports to a single instance.

Depending on the implementation, some or none of these different sharing modes may be supported. If they are supported, several types of interfaces could be used:

1. The consumer may explicitly search (via a *findServiceData* query) for an existing basket that matches its requirements. It would then go directly to the basket instance.

2. The consumer may simply try to create a new basket. The filtering service would identify that the new basket has similar properties to existing baskets and would return a handle to an existing basket rather than to a new one.
3. The consumer may provide an existing basket handle when invoking the *createService* operation. The filtering service would reestablish the subscription to the existing basket and would create the basket if it does not already exist.

The set of basket operations that are available to the consumer depends on the sharing mode and the basket capabilities. These may be controlled by the basket instance itself or by the filtering service. In the latter case, the consumer invokes operations on the filter service, which makes the necessary adjustments to the basket.

Issue 12 *Who should detect identical baskets? Do we REQUIRE that a filtering service be able to identify when a createService operation should return a handle to an existing basket rather than a new one? Is this an implementation issue?*

Issue 13 *Should the basket service include operations for basket sharing or should everything be done through the filtering service?*

2.7 Non-native Baskets

The filtering service can also interact with *non-native* (or *foreign*) baskets that were not created by the service. This is useful when one basket listens to several filtering services (e.g., for collecting logs in a cluster). It is also useful for implementing custom baskets, which have capabilities beyond those created by the filtering service.

In order to receive messages from the filtering service, such baskets must be made known by the filtering service, namely they must be subscribed to the service. This can be done by directly subscribing to the filtering service via the *NotificationSource::subscribe* operation.

Such foreign baskets SHOULD be made available to other consumers by registering them at the filtering service via the *Registration::registerService* operation. In this case, the baskets MUST also provide the *regs:serviceParameters* XML element used for their creation. Custom baskets MUST extend the *regs:serviceParametersType* and the *regs:BasketProfileType* types to expose their unique capabilities.

Issue 14 *Should this be done through the createService operation instead? A non-native basket may give its handle as an argument to the filtering service. The filtering service would then perform all relevant operations, such as registration and subscription.*

Should we prohibit direct subscription to the filtering service by non-native baskets?

2.8 Interaction with Producers

The main interaction of the filtering service with the producers is reception of the report messages. The filtering service does not require any information on the producer in order to be able to filter its messages. However, the filtering service should be able to provide consumers with information on the available producers and their messages. The filtering service may also affect the granularity of messages produced by each producer.

2.8.1 Producer Discovery

Each producer SHOULD provide metadata with additional information on its report messages. The information SHOULD include an XML description of the messages produced.

Categories the structure of the category named hierarchy.

Properties the names and types of all producer-specific properties.

Body schema a schema for the producer-specific body element. This schema allows for selection, based on the producer-specific body extensibility element.

QName a qualified name for the producer, which can be used for message selection.

ServiceLocator if the producer is a grid service, then it MUST provide a *serviceLocator*.

Issue 15 *How should the producer information be made available to consumers? Should the filtering service be a registry for producers? If so, how would it handle producers who are not grid services? Should we define an SDE for producer metadata?*

Issue 16 *What is the mechanism by which the filtering service obtains the producer metadata? One option is that every producer would have an SDE that includes this information and the filtering service would query this SDE. However, that is applicable only to producers who themselves are grid services. Also, the filtering service must be able to obtain a list of all available producers (e.g., through the container registry) before it can query each producer for this information.*

Another option is to explicitly register every producer and provide this data during the registration. We can define a new operation to support this or we can use the optional XML fragment of the Registration::registerService operation. In the latter case the producer must be a grid service. In both cases the producer must know the filtering service to be able to invoke the operation; however, this is not an issue since it must be aware of the service in order to send its report messages.

2.8.2 Changing a Producer's Reporting Granularity

Producing fine-grained reporting messages may consume considerable producer resources. It is desirable to be able to produce only messages for which there is a demand. Producers may allow the filtering service to set their reporting granularity based on the existing baskets and their filtering rules. For example, if there is no basket subscribed for DEBUG messages, the producer's debug flag may be switched off.

In order to support this kind of optimization, the filtering service must be able to analyze and aggregate all filtering rules. When new baskets with new filtering rules are created, the filtering service can determine that a *Producer::changeGranularity* operation must be invoked on some producers.

Issue 17 *How do you invoke this operation on non grid service producers? Do we REQUIRE every filtering service to be able to do this?*

An alternative: An administrator consumer may first directly increase the debug level of relevant producers and only then create a new basket.

3 Basket Grid Service

The basket grid service stores reports generated by producers. A basket consists of a filtering rule, a report repository, and a basket profile

The *filtering rule* determines which reports are stored in the basket. Refer to Section 2.3.2 for the filtering rules definition.

The basket stores all reports in the *report repository*. The report repository presents reports through a service data element, called *basketServiceReports* (3.3.1). This allows consumers to use the standard *GridService::findServiceData* operation in order to access reports.

The *basket profile* specifies different basket characteristics, such as the maximum size of the report repository, policy for when the report repository becomes full, etc. (3.3.3).

The basket service implements the Basket and NotificationSource portTypes.

The basket service MUST support the queries defined in the filter service (see Section 2.3.2). The *simple-Filter* query is described in Section 3.3.2. In addition, basket services MAY support additional queries.

3.1 Basket as a Multi-Consumer Service

The basket service MAY allow multiple consumers to access its services. The creator of the basket service specifies the permissions for other consumers to use the basket.

Issue 18 *What does it mean that the basket keeps user state:*

- *If one user deletes a report, is this report deleted for other users? (yes)*
- *If one user creates a subscription, can another user find this subscription and join it? Does it make sense for a basket service?*
- *What is the other user state that the basic Basket service should address?*

3.2 Lifetime of the Basket Instance

The basket grid service supports the standard soft state lifetime management described in the Grid Services Specification Document [24].

3.3 ServiceData Elements

3.3.1 BasketServiceReports

```
<gsdl:serviceDataDescription name="BasketServiceReports"
  type="regs:basketServiceReportsType"
  minOccurs="1"          maxOccurs="Exactly one"
  xmlns:regs="http://hrl.ibm.com/ogsa/schema/regs/"
  xmlns:gsdl="http://schemas.gridforum.org/gridServices/">
  <wsdl:documentation>
    An XML regs:regs:basketServiceReports element that
    contains all reports in the basket.
  </wsdl:documentation>
</gsdl:serviceDataDescription>
```

The schema is shown in Figure 10



Figure 10: Schema for SDE BasketServiceReports

The *BasketServiceReports* service data element contains all reports in the basket. Prior to storing an incoming report in the repository, the Basket first adds its *section* element to the report (refer to 2.2.1 for the report description). The basket section contains the following information:

name attribute set to “ReportMetaData”.

author attribute set to the grid service reference of the basket.

creationTime attribute set to the time when the report was delivered to the basket.

property element contains the size of the report. The name of the property is “reportSize”.

Issue 19 *Need to specify the format for the size.*

body element does not present.

The rationale for putting all reports in one service data element is to enable an easy, but powerful, query to find interesting reports. Indeed, since all reports are organized in one XML element with a well-defined structure, the entire power of XPath may be used to query reports. For example, using XPath, it is easy to write a query to find all sequences of three WARN reports.

Refer to Appendix A.4 for the *BasketServiceReports* schema defining the report service data element.

3.3.2 queryBySimpleFilter

A *queryBySimpleFilter* results in all reports that match the filter. The simpleFilter QueryExpressionType follows:

```

<gsdl:serviceData
  name="gsdl:QueryExpressionTypes">
  <xsd:anyURI>
    http://hrl.ibm.com/ogsa/schema/regs/queryBySimpleFilter
  </xsd:anyURI>
</gsdl:serviceData>
  
```

Issue 20 *Do we need to define our XPath query when XPath is already defined in OGSA?*

The QueryExpression schema for the simple filter is the same as the schema used for the *simpleFilter* presented in Section 2.3.2.

The resulting format is an XML document that complies with the XML schema diagram shown in Figure 11 (see Appendix A.8 for the complete XML schema).

The meaning of the attributes and elements in the result is as follows:

numberOfReports attribute specifies the number of reports in the result.

timeStamp attribute specifies the basket service timestamp when the result was created.

report element is a report matching the query.



Figure 11: Result format of the simpleFilter query

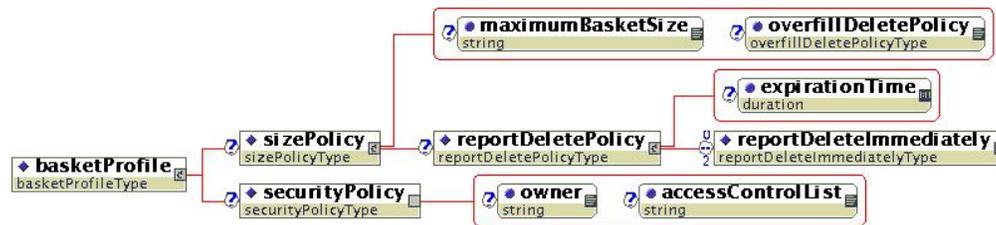


Figure 12: Schema for SDE BasketProfile

3.3.3 BasketProfile

```
<gsdl:serviceDataDescription name="BasketProfile"
  type="regs:basketProfile"
  minOccurs="1"      maxOccurs="Exactly one"
  xmlns:regs="http://hrl.ibm.com/ogsa/schema/regs/"
  xmlns:gsdl="http://schemas.gridforum.org/gridServices/">
  <wsdl:documentation>
    An XML regs:basketProfile element that defines the
    behavior of the basket.
  </wsdl:documentation>
</gsdl:serviceDataDescription>
```

The schema is shown in Figure 12

The basket profile defines the behavior of the basket. All basket services **MUST** support the XML schema shown above. A basket service **MAY** extend the profile in order to support a more sophisticated configuration. The basic basket profile defines the following policies:

- Policy for basket size management.
- Security policy.

Refer to Appendix A.5 for the schema defining the basket profile.

The semantics for the *BasketProfile* are as follows:

sizePolicy element specifies the policy for basket size management.

maximumBasketSize attribute specifies the maximum basket size. The basket service **MUST** provide the default value for the *maximumBasketSize*.

overflowDeletePolicy attribute specifies which reports are deleted when the repository is full. This is equal to one of the following strings:

DeleteOldest the oldest reports are deleted from the repository. The age of the reports is determined based on the *creationTime* attribute of the “ReportMetaData” section (see Section 3.3.1).



Figure 13: Schema for SDE BasketCurrentState

DeleteIncoming incoming reports are not stored in the repository.

reportDeletePolicy element specifies the policy for deleting reports in the repository.

expirationTime attribute specifies the expiration time for attributes stored in the repository.

reportDeleteImmediately element specifies whether reports are deleted after they are read by a consumer. This element contains one of the following string contents:

DeleteAfterReadOnPush reports MAY be deleted after they are delivered to ANY consumer through the notification mechanism.

DeleteAfterReadOnPoll reports MAY be deleted after ANY consumer requested the reports through the *GridService::findServiceData* operation.

securityPolicy element specifies the security policy for the basket.

owner attribute specifies the owner of the basket.

Issue 21 *Need to define the format for the owner. Currently it is just a string.*

accessControlList attribute specifies the permissions of different consumers to query the reports and perform operations on the basket.

Issue 22 *Need to define the format for the accessControlList. Currently it is just a string.*

3.3.4 BasketCurrentState

```
<gsdl:serviceDataDescription name="BasketCurrentState"
  type="regs:basketCurrentState"
  minOccurs="1"          maxOccurs="Exactly one"
  xmlns:regs="http://hrl.ibm.com/ogsa/schema/regs/"
  xmlns:gsdl="http://schemas.gridforum.org/gridServices/">
  <wsdl:documentation>
    An XML regs:basketCurrentState element that reflects the
    current state of the basket.
  </wsdl:documentation>
</gsdl:serviceDataDescription>
```

The schema is shown in Figure 13

The BasketCurrentState service data reflects the current state of the basket. The basketCurrentState reveals the following information:

- The size used by the basket. It is defined by the SizeUsage complex type in the schema. Based on the query results of the SizeUsage, a consumer may take steps to avoid the size overflow of the basket.

Refer to Appendix A.6 for the schema that defines the basket's current state.

The meaning of the attributes in the *basketCurrentState* follows:

usedBasketSize represents the currently used basket size.

freeBasketSize represents the current free basket size.

numberOfReports represents the number of reports currently stored in the basket.



Figure 14: *subscribeByReports* SubscriptionExpressionType. This allows subscription to a basket in order to receive reports

3.3.5 subscribeByReports

A *subscribeByReports* results in notification messages being sent when one or more reports arrive at the basket.

The basket service MUST include the following service data value element:

```
<gsdl:serviceData
  name="gsdl:SubscriptionExpressionTypes">
  <xsd:anyURI>
    http://hrl.ibm.com/ogsa/schema/regs/subscribeByReports
  </xsd:anyURI>
</gsdl:serviceData>
```

The diagram of the *subscribeByReports* SubscriptionExpressionType is shown in Figure 14 (see Appendix A.9 for the complete XML schema):

The meaning of the attributes in the *subscribeByReports* is as follows:

minInterval attribute has exactly the same meaning as in the *subscribeByServiceDataName* [24].

numberOfReportsToAggregate attribute specifies the minimum number of new reports aggregated prior to delivering them to the subscriber.

XSLTpath attribute specifies the URI for a XSLT document. The basket service transforms notification messages according to the specified XSLT document prior to delivering them to subscribers. The schema diagram for a notification message before the transformation is shown in Figure 11.

3.4 subscribeByBasketStatus

A *subscribeByBasketStatus* results in notification messages being sent when the basket status changes.

The basket service MUST include the following service data value element:

```
<gsdl:serviceData
  name="gsdl:SubscriptionExpressionTypes">
  <xsd:anyURI>
    http://hrl.ibm.com/ogsa/schema/regs/subscribeByBasketStatus
  </xsd:anyURI>
</gsdl:serviceData>
```

The diagram of the *subscribeByBasketStatus* SubscriptionExpressionType is shown in Figure 15 (see Appendix A.10 for the complete XML schema).

The meaning of the elements in the *subscribeByReports* is as follows:

fullIndicator element allows specification of the degree of repository fullness at which subscribers are notified.

percentFull attribute specifies the degree of repository fullness in percentages. When the repository reaches the specified degree, the basket sends a notification to subscribers. The delivered message conforms to the schema diagram shown in Figure 16 (see Appendix A.11 for the complete XML schema).



Figure 15: *subscribeByBasketStatus* SubscriptionExpressionType. This allows subscription to a basket in order to receive its status changes

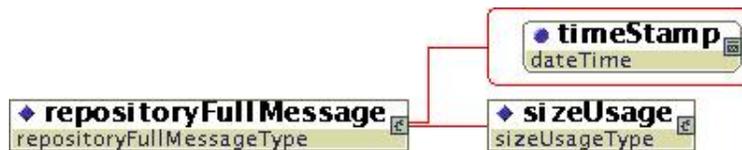


Figure 16: Schema for a message delivered when repository fullness reaches the specified degree

Issue 23 How does the subscriber know the source (service locator) from which it receives the message? This is the OGSA question.

XSLTPath attribute specifies the transformation performed on the message before it is delivered to subscribers. Note, the message, before transformation, conforms to the schema diagram shown in Figure 16.

notifyIfReportDiscarded element allows notification of subscribers when the repository discards incoming reports.

notify attribute: If set to *true*, subscribers are notified when the repository discards incoming reports. The delivered message conforms to the schema diagram shown in Figure 17 (see Appendix A.12 for the complete XML schema).

XSLTPath attribute specifies the transformation performed on the message before it is delivered to subscribers. Note, the message, before transformation, conforms to the schema diagram shown in Figure 17.

3.5 Basket PortType: Operations and Messages

3.5.1 *Basket::performActionsOnReports*

Performs actions on reports that match the query condition. First, the basket applies the *query* on the *reports* service data element. Next, the basket applies *actions*, one after another, on the results of the query.

Input

query Query supported by the basket.

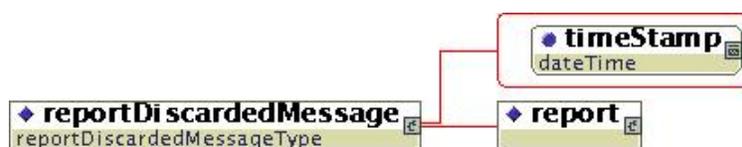


Figure 17: Schema for a message delivered when the repository discards incoming reports



Figure 18: Schema diagram defining the *basketActions*

actions The list of actions that conform to the type *basketActions*. The schema diagram that defines the *basketActions* is shown in Figure 18. Refer to Appendix A.7 for the entire schema.

The meaning of the elements in the *basketActions* is as follows:

delete this action deletes the results. For example, if the results of the query are reports, the selected reports are deleted. If the results of the query are elements of the report, the elements are removed from the report.

addTag this action adds “tags” to selected reports. If this action is specified, the result of the query MUST be reports. The element(s) under the addTag element are added to all selected reports.

Output

Faults

- TBD

4 Acknowledgments

We would like to thank Oleg Frenkel for his contribution to this work; and to Dave Elko and Jim Warnes from the IBM Server Group - their input has been essential to the consolidation of the ideas presented in this document. Also special thanks to Chani Sacharen for her careful review of this document.

References

- [1] CORBA. *Systems Management: Event Management Service*, 1997. X/Open Document Number: P437, <http://www.opengroup.org/onlinepubs/008356299>.
- [2] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proc. 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*. IEEE Press, August 2001. <http://www.globus.org/research/papers/MDS-HPDC.pdf>.
- [3] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. Journal of Super-computer Applications*, 11(2):111–128, 1997.
- [4] I. Foster and C. Kesselman. The globus project: A status report. In *Proc. IPPS/SPDP Heterogeneous Computing Workshop*, pages 4–18, 1998.
- [5] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann, 1999.
- [6] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(6), June 2002.
- [7] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. <http://www.globus.org/research/papers/ogsa.pdf>, June 2002.
- [8] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal Supercomputer Applications*, 15(3), 2001. <http://www.globus.org/research/papers/anatomy.pdf>.
- [9] S. Graham, S. Simeonov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, and R. Neyama. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. SAMS, 1st edition, December 2002.
- [10] C. Gülcü. Short introduction to log4j. <http://jakarta.apache.org/log4j/docs/manual.html>, March 2002.
- [11] D. Gunter, B. Tierney, K. Jackson, J. Lee, and M. Stoufer. Dynamic monitoring of high-performance distributed applications. In *Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC)*, pages 163–170, July 2002.
- [12] G. Hamilton. *Java Logging APIs - Draft 0.55*. Sun Microsystems, October 2000. <http://jcp.org/aboutJava/communityprocess/review/jsr047>.
- [13] IBM. *MVS Programming: Assembler Services Guide*, 3 edition, March 2002. <http://publibfi.boulder.ibm.com/epubs/pdf/iea2a620.pdf>.
- [14] C. Lonvick. The BSD syslog protocol. RFC 3164, IETF, August 2001. <http://www.ietf.org/rfc/rfc3164.txt>.
- [15] Microsoft. *Platform SDK: Debugging and Error Handling*. http://msdn.microsoft.com/library/en-us/debug/eventlog_2tbb.asp.

- [16] D. New and M. Rose. Reliable delivery for syslog. RFC 3195, IETF, November 2001. <http://www.ietf.org/rfc/rfc3195.txt>.
- [17] E. Siever, S. Spainhour, J. Hekman, and S. Figgins. *Linux in a Nutshell*, chapter 3, page 222. O'Reilly, 3rd edition, August 2000.
- [18] A. Slominski, M. Govindaraju, D. Gannon, and R. Bramley. SoapRMI events: Design and implementation. Technical Report TR549, Indiana University, May 2001. <http://www.cs.indiana.edu/Research/techreports/TR549.shtml>.
- [19] W. Smith and D. Gunter. Simple LDAP schemas for grid monitoring. Technical report, Global Grid Forum Performance Working Group, June 2001. <http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-13-1.pdf>.
- [20] W. Smith, D. Gunter, and D. Quesnel. A simple XML producer-consumer protocol. Technical report, Global Grid Forum Performance Working Group, June 2001. <http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-8-2.pdf>.
- [21] Sun Microsystems. *Jini Technology Core Platform Specification - Distributed Events*, 1.2 edition. <http://www.sun.com/software/jini/specs/jini1.2html/event-spec.html>.
- [22] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A grid monitoring architecture. Technical report, Global Grid Forum Performance Working Group, January 2002. <http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-2.pdf>.
- [23] B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, and M. Thompson. A monitoring sensor management system for grid environments. In *Proceedings of the Ninth IEEE International Symposium on High-Performance Distributed Computing (HPDC)*, pages 97–104, August 2000. <http://www-didc.lbl.gov/papers/JAMM.HPDC00.pdf>.
- [24] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, and C. Kesselman. Grid service specification. Draft 3, Global Grid Forum, July 2002. <http://www.globus.org/research/papers/gsspec.pdf>.
- [25] W3C. *XSL Transformations (XSLT)*, November 1999. <http://www.w3.org/TR/xslt>.
- [26] W3C. *Extensible Markup Language (XML) 1.0*, 2nd edition, August 2000. <http://www.w3.org/TR/2000/WD-xml-2e-20000814>.
- [27] W3C. *SOAP Version 1.2 Part 0: Primer*, December 2001. <http://www.w3.org/TR/2001/WD-soap12-part0-20011217>.
- [28] W3C. *XML Schema Part 0: Primer*, May 2001. <http://www.w3.org/TR/xmlschema-0>.
- [29] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, October 1999. <http://www.cs.ucsb.edu/~rich/publications/nws-arch.ps.gz>.

A Schemas

A.1 regs:report Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://hr1.ibm.com/ogsa/schema/regs"
xmlns:regs="http://hr1.ibm.com/ogsa/schema/regs"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="$Revision: 2.1 $">
  <simpleType name="authorType">
    <annotation>
      <documentation>An identifier for the authore (e.g., a GSH)</documentation>
    </annotation>
    <restriction base="QName"/>
  </simpleType>
  <attributeGroup name="sectionAttributes">
    <annotation>
      <documentation>common attributes for every section</documentation>
    </annotation>
    <attribute name="name" type="NMTOKEN">
      <annotation>
        <documentation>combination of author and name must be unique</documentation>
      </annotation>
    </attribute>
    <attribute name="author" type="regs:authorType" use="required">
      <annotation>
        <documentation>the author of the section (e.g., GSH)</documentation>
      </annotation>
    </attribute>
    <attribute name="creationTime" type="dateTime"/>
  </attributeGroup>
  <simpleType name="propertyTypeType">
    <annotation>
      <documentation>
        Type of property value - should be a name of a type that is
        derived from a primitive type.
      </documentation>
    </annotation>
    <restriction base="string">
      <pattern value="xsd:.*"/>
      <pattern value="regs:.*"/>
      <pattern value=".+.*"/>
    </restriction>
  </simpleType>
  <simpleType name="propertyNameType">
    <annotation>
      <documentation>Name of property, must not start with "REGS_" </documentation>
    </annotation>
    <restriction base="NMTOKEN">
      <pattern value="[^R].*[R[^E]].*|RE[^G].*|REG[^S].**"/>
    </restriction>
  </simpleType>
  <simpleType name="producerPropertyTypeType">
    <annotation>
      <documentation>
        Type of property value - should be a name of a type that is
        derived from a primitive type
      </documentation>
    </annotation>
    <restriction base="regs:propertyTypeType"/>
  </simpleType>
  <simpleType name="producerPropertyNameType">
    <annotation>
      <documentation>Name of property </documentation>
    </annotation>
    <restriction base="NMTOKEN">
      <pattern value="REGS_.*"/>
    </restriction>
  </simpleType>
  <simpleType name="levelStringType">
    <annotation>
      <documentation>Severity level as string</documentation>
    </annotation>
    <restriction base="NMTOKEN">
      <enumeration value="ALL"/>
      <enumeration value="DEBUG"/>
      <enumeration value="ERROR"/>
      <enumeration value="FATAL"/>
      <enumeration value="INFO"/>
      <enumeration value="WARN"/>
    </restriction>
  </simpleType>
  <simpleType name="levelValueType">
    <annotation>
      <documentation>severity level as value</documentation>
    </annotation>
    <restriction base="unsignedShort"/>
  </simpleType>

```

```

<simpleType name="levelType">
  <annotation>
    <documentation>severity level</documentation>
  </annotation>
  <union memberTypes="regs:levelValueType regs:levelStringType"/>
</simpleType>
<simpleType name="categoryType">
  <annotation>
    <documentation>Named hierarchy (eg "package/module/subModule")</documentation>
  </annotation>
  <restriction base="string">
    <pattern value="\w+(\.\/\w+)*"/>
  </restriction>
</simpleType>
<complexType name="bodyType">
  <annotation>
    <documentation>Extensibility place holder</documentation>
  </annotation>
  <sequence>
    <any namespace="##any" processContents="skip" maxOccurs="unbounded"/>
  </sequence>
</complexType>
<complexType name="propertyType">
  <annotation>
    <documentation>User defined property</documentation>
  </annotation>
  <simpleContent>
    <extension base="anySimpleType">
      <attribute name="name" type="regs:propertyNameType" use="required"/>
      <attribute name="type" type="regs:propertyTypeType" use="required"/>
    </extension>
  </simpleContent>
</complexType>
<complexType name="producerLevelPropertyType">
  <annotation>
    <documentation>REGS_level </documentation>
  </annotation>
  <simpleContent>
    <extension base="regs:levelType">
      <attribute name="name" use="required">
        <simpleType>
          <restriction base="regs:producerPropertyNameType">
            <enumeration value="REGS_level"/>
          </restriction>
        </simpleType>
      </attribute>
      <attribute name="type" use="required">
        <simpleType>
          <restriction base="regs:producerPropertyTypeType">
            <enumeration value="regs:levelType"/>
          </restriction>
        </simpleType>
      </attribute>
    </extension>
  </simpleContent>
</complexType>
<complexType name="producerCategoryPropertyType">
  <annotation>
    <documentation>REGS_category</documentation>
  </annotation>
  <simpleContent>
    <extension base="regs:categoryType">
      <attribute name="name" use="required">
        <simpleType>
          <restriction base="regs:producerPropertyNameType">
            <enumeration value="REGS_category"/>
          </restriction>
        </simpleType>
      </attribute>
      <attribute name="type" use="required">
        <simpleType>
          <restriction base="regs:producerPropertyTypeType">
            <enumeration value="regs:categoryType"/>
          </restriction>
        </simpleType>
      </attribute>
    </extension>
  </simpleContent>
</complexType>
<complexType name="producerTimeStampPropertyType">
  <annotation>
    <documentation>REGS_timeStamp (optional)</documentation>
  </annotation>
  <simpleContent>
    <extension base="dateTime">
      <attribute name="name" use="required">
        <simpleType>
          <restriction base="regs:producerPropertyNameType">
            <enumeration value="REGS_timeStamp"/>
          </restriction>
        </simpleType>
      </attribute>
    </extension>
  </simpleContent>
</complexType>

```

```

        </attribute>
        <attribute name="type" use="required">
          <simpleType>
            <restriction base="regs:producerPropertyTypeType">
              <enumeration value="xsd:dateTime"/>
            </restriction>
          </simpleType>
        </attribute>
      </extension>
    </simpleContent>
  </complexType>
  <complexType name="producerSectionType">
    <annotation>
      <documentation>Producer section that is present in every report</documentation>
    </annotation>
    <sequence>
      <element name="level" type="regs:producerLevelPropertyType">
        <annotation>
          <documentation>E.g., INFO, FATAL, 20</documentation>
        </annotation>
      </element>
      <element name="category" type="regs:producerCategoryPropertyType">
        <annotation>
          <documentation>E.g., "package/moduleA/report"</documentation>
        </annotation>
      </element>
      <element name="timeStamp" type="regs:producerTimeStampPropertyType" minOccurs="0"/>
      <element name="property" type="regs:propertyType" minOccurs="0" maxOccurs="unbounded">
        <annotation>
          <documentation>E.g., <property name="REGS_index" type="int">17</property>
        </documentation>
      </annotation>
      </element>
      <element name="body" type="regs:bodyType"/>
    </sequence>
    <attributeGroup ref="regs:sectionAttributes"/>
  </complexType>
  <complexType name="sectionType">
    <annotation>
      <documentation>Sections must have a unique (author+name)</documentation>
    </annotation>
    <sequence>
      <element name="property" type="regs:propertyType" minOccurs="0" maxOccurs="unbounded">
        <annotation>
          <documentation>E.g., <property name="myID" type="xsd:string">"appA:report"</property>
        </documentation>
      </annotation>
      </element>
      <element name="body" type="regs:bodyType" minOccurs="0">
        <annotation>
          <documentation>E.g.,
            <!--<my:appData>DATA<my:MORE_DATA/>
              </my:appData -->
          </documentation>
        </annotation>
      </element>
    </sequence>
    <attributeGroup ref="regs:sectionAttributes"/>
  </complexType>
  <element name="report">
    <annotation>
      <documentation>Report root</documentation>
    </annotation>
    <complexType>
      <sequence>
        <element name="producerSection" type="regs:producerSectionType"/>
        <element name="section" type="regs:sectionType" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
    <unique name="sectionID">
      <selector xpath="./section"/>
      <field xpath="@name"/>
      <field xpath="@author"/>
    </unique>
  </element>
</schema>

```

A.2 regs:filter Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://hrl.ibm.com/ogsa/schema/regs"
xmlns:regs="http://hrl.ibm.com/ogsa/schema/regs"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="$Revision: 2.1 $">
  <simpleType name="levelStringType">
    <annotation>
      <documentation>Severity level as string</documentation>
    </annotation>
    <restriction base="NMTOKEN">
      <enumeration value="ALL"/>
      <enumeration value="DEBUG"/>
      <enumeration value="ERROR"/>
      <enumeration value="FATAL"/>
      <enumeration value="INFO"/>
      <enumeration value="WARN"/>
    </restriction>
  </simpleType>
  <simpleType name="levelValueType">
    <annotation>
      <documentation>Severity level as value</documentation>
    </annotation>
    <restriction base="unsignedShort"/>
  </simpleType>
  <simpleType name="levelType">
    <annotation>
      <documentation>severity level</documentation>
    </annotation>
    <union memberTypes="regs:levelValueType regs:levelStringType"/>
  </simpleType>
  <simpleType name="categorySelectorType">
    <annotation>
      <documentation>
        Category selection pattern. A simplified XPath-like expression
        derived from the selector attribute of the xsd:unique element. "*"
        means wildcard, "/" means descendant.
      </documentation>
    </annotation>
    <restriction base="token">
      <pattern value="((\\c+|\\*)|\\.)(/((\\c+|\\*)|\\.))*\\((\\c+|\\*)|\\.)(/((\\c+|\\*)|\\.))*"/>
    </restriction>
  </simpleType>
  <complexType name="simpleFilterType">
    <annotation>
      <documentation>
        Selection by level, category or producer.
        If left empty includes all.
      </documentation>
    </annotation>
    <sequence>
      <element name="producers" minOccurs="0">
        <annotation>
          <documentation>List of producers to select from.</documentation>
        </annotation>
        <simpleType>
          <list itemType="QName"/>
        </simpleType>
      </element>
      <choice minOccurs="0">
        <annotation>
          <documentation>
            Level selection. A list of levels or a range. Level can
            be a positive integer or one off: 0="ALL", 10="DEBUG", 20="INFO",
            30="WARN", 40="ERROR", 50="FATAL"
          </documentation>
        </annotation>
        <element name="levels">
          <annotation>
            <documentation>List of levels.</documentation>
          </annotation>
          <simpleType>
            <list itemType="regs:levelType"/>
          </simpleType>
        </element>
        <sequence>
          <element name="minLevel" type="regs:levelType" minOccurs="0"/>
          <element name="maxLevel" type="regs:levelType" minOccurs="0"/>
        </sequence>
      </choice>
      <element name="category" type="regs:categorySelectorType" minOccurs="0">
        <annotation>
          <documentation>Category selection pattern. A simplified
            XPath-like expression derived from the selector attribute of the
            xsd:unique element.
            "*" means wildcard, "/" means descendant.
          </documentation>
        </annotation>
      </element>
    </sequence>
  </complexType>

```

```

    </sequence>
  </complexType>
  <complexType name="xpathFilterType">
    <annotation>
      <documentation>Selection by an XPath predicate on reopr. </documentation>
    </annotation>
    <complexContent>
      <extension base="regs:simpleFilterType">
        <sequence>
          <element name="xpath" type="string">
            <annotation>
              <documentation>An xpath expression with report being the root. </documentation>
            </annotation>
          </element>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="filterType">
    <annotation>
      <documentation>One of the regs filters</documentation>
    </annotation>
    <choice>
      <element ref="regs:simpleFilter"/>
      <element ref="regs:xpathFilter"/>
    </choice>
  </complexType>
  <element name="simpleFilter" type="regs:simpleFilterType">
    <annotation>
      <documentation>Selection by level, category or producer.
      If left empty includes all.</documentation>
    </annotation>
  </element>
  <element name="xpathFilter" type="regs:xpathFilterType">
    <annotation>
      <documentation>Selection by an XPath predicate report.</documentation>
    </annotation>
  </element>
</schema>

```

A.3 regs: basketFactory Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://hrl.ibm.com/ogsa/schema/regs"
  xmlns:regs = "http://hrl.ibm.com/ogsa/schema/regs"
  elementFormDefault = "qualified"
  attributeFormDefault = "unqualified"
  version = "$Revision: 2.1 $">
<include schemaLocation = "regs_filter.xsd"/>
<include schemaLocation = "BasketProfile.xsd"/>
<element name = "ServiceParameters" type = "regs:serviceParametersType"/>
<complexType name = "serviceParametersType">
  <sequence>
    <element name = "messageFilter" type = "regs:filterType">
      <annotation>
        <documentation>
          filter type defined in regs_filter
        </documentation>
      </annotation>
    </element>
    <element name = "XSLTpath" type = "anyURI" minOccurs = "0">
      <annotation>
        <documentation>
          A reference to an XSLT file that should be used to
          format message BEFORE sent to basket. Result of
          transform must still be a valid regs_report
        </documentation>
      </annotation>
    </element>
    <element name = "basketProfile" type = "regs:basketProfileType">
      <annotation>
        <documentation>
          a basket profile element of type regs:basketProfileType
        </documentation>
      </annotation>
    </element>
  </sequence>
</complexType>
</schema>
```

A.4 regs:BasketServiceReports Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by Turbo XML 2.3.1.100. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://hrl.ibm.com/ogsa/schema/regs"
  xmlns:regs = "http://hrl.ibm.com/ogsa/schema/regs"
  xmlns:regsl = "http://hrl.ibm.com/ogsa/schema/regs"
  version = "$Revision: 2.1 $"
  elementFormDefault = "qualified"
  attributeFormDefault = "unqualified">

  <include schemaLocation="regs_report.xsd"/>
  <element name = "basketServiceReports" type = "regs:basketServiceReportsType"/>
  <complexType name = "basketServiceReportsType">
    <sequence>
      <element ref = "regs:report" minOccurs = "0" maxOccurs = "unbounded"/>
    </sequence>
  </complexType>
</schema>
```

A.5 regs:BasketProfile Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by Turbo XML 2.3.1.100. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://hrl.ibm.com/ogsa/schema/regs"
  xmlns:regs = "http://hrl.ibm.com/ogsa/schema/regs"
  version = "$Revision: 2.0 $"
  elementFormDefault = "qualified"
  attributeFormDefault = "unqualified">
  <!-- $Revision: 2.0 $ -->

  <element name = "basketProfile" type = "regs:basketProfileType"/>
  <complexType name = "basketProfileType">
    <sequence>
      <element name = "sizePolicy" type = "regs:sizePolicyType" minOccurs = "0"/>
      <element name = "securityPolicy" type = "regs:securityPolicyType" minOccurs = "0"/>
    </sequence>
  </complexType>
  <complexType name = "sizePolicyType">
    <sequence>
      <element name = "reportDeletePolicy" type = "regs:reportDeletePolicyType" minOccurs = "0"/>
    </sequence>
    <attribute name = "maximumBasketSize" type = "string"/>
    <attribute name = "overflowDeletePolicy" type = "regs:overflowDeletePolicyType"/>
  </complexType>
  <complexType name = "reportDeletePolicyType">
    <sequence>
      <element name = "reportDeleteImmediately" type = "regs:reportDeleteImmediatelyType" minOccurs = "0" maxOccurs = "2"/>
    </sequence>
    <attribute name = "expirationTime" type = "duration"/>
  </complexType>
  <simpleType name = "reportDeleteImmediatelyType">
    <restriction base = "string">
      <enumeration value = "DeleteAfterReadOnPush"/>
      <enumeration value = "DeleteAfterReadOnPoll"/>
    </restriction>
  </simpleType>
  <simpleType name = "overflowDeletePolicyType">
    <restriction base = "string">
      <enumeration value = "DeleteOldest"/>
      <enumeration value = "DeleteIncoming"/>
    </restriction>
  </simpleType>
  <complexType name = "securityPolicyType">
    <attribute name = "owner" type = "string"/>
    <attribute name = "accessControlList" type = "string"/>
  </complexType>
</schema>
```

A.6 regs:BasketCurrentState Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by DHL (IBM) -->
<!--Generated by Turbo XML 2.3.0.100. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema targetNamespace="http://hrl.ibm.com/ogsa/schema/regs"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:regs="http://hrl.ibm.com/ogsa/schema/regs"
elementFormDefault="unqualified" attributeFormDefault="unqualified"
version="$Revision: 2.0 $">
  <element name="basketCurrentState" type="regs:basketCurrentStateType"/>
  <element name="sizeUsage" type="regs:sizeUsageType"/>
  <complexType name="basketCurrentStateType">
    <sequence>
      <element ref="regs:sizeUsage"/>
    </sequence>
  </complexType>
  <complexType name="sizeUsageType">
    <attribute name="usedBasketSize" use = "required" type="string"/>
    <attribute name="freeBasketSize" use = "required" type="string"/>
    <attribute name="numberOfReports" use = "required" type="string"/>
  </complexType>
</schema>
```

A.7 regs:BasketActions Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by DHL (IBM) -->
<!--Generated by Turbo XML 2.3.0.100. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema targetNamespace="http://hrl.ibm.com/ogsa/schema/regs"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:regs="http://hrl.ibm.com/ogsa/schema/regs"
elementFormDefault="unqualified" attributeFormDefault="unqualified"
version="$Revision: 2.0 $">
  <!-- $Revision: 2.0 $ -->
  <element name="basketActions" type="regs:basketActionsType"/>
  <complexType name="basketActionsType">
    <sequence>
      <element name="action" type="regs:actionType" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
  <complexType name="actionType">
    <choice>
      <element name="delete"/>
      <element name="addTag"/>
    </choice>
  </complexType>
</schema>
```

A.8 regs:ReportSet Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by Turbo XML 2.3.1.100. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://hrl.ibm.com/ogsa/schema/regs"
  xmlns:regs = "http://hrl.ibm.com/ogsa/schema/regs"
  xmlns:regs1 = "http://hrl.ibm.com/ogsa/schema/regs"
  version = "$Revision: 2.0 $"
  elementFormDefault = "unqualified"
  attributeFormDefault = "unqualified">
  <include schemaLocation = "regs_report.xsd"/>
  <!-- $Revision: 2.0 $ -->

  <element name = "reportSet" type = "regs:reportSetType"/>
  <complexType name = "reportSetType">
    <sequence>
      <element ref = "regs:report" minOccurs = "0" maxOccurs = "unbounded"/>
    </sequence>
    <attribute name = "numberOfReports" use = "required" type = "nonNegativeInteger"/>
    <attribute name = "timeStamp" use = "required" type = "dateTime"/>
  </complexType>
</schema>
```

A.9 regs:SubscribeByReports Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://hr1.ibm.com/ogsa/schema/regs"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:regs="http://hr1.ibm.com/ogsa/schema/regs"
elementFormDefault="unqualified" attributeFormDefault="unqualified"
version="$Revision: 2.0 $">
  <!-- $Revision: 2.0 $ -->
  <element name="subscribeByReports" type="regs:subscribeByReportsType"/>
  <complexType name="subscribeByReportsType">
    <attribute name="minInterval" type="duration"/>
    <attribute name="numberOfReportsToAggregate" type="positiveInteger"/>
    <attribute name="XSLTpath" type="anyURI"/>
  </complexType>
</schema>
```

A.10 regs:SubscribeByBasketStatus Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by Turbo XML 2.3.1.100. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://hrl.ibm.com/ogsa/schema/regs"
  xmlns:regs = "http://hrl.ibm.com/ogsa/schema/regs"
  version = "$Revision: 2.0 $"
  elementFormDefault = "unqualified"
  attributeFormDefault = "unqualified">
  <!-- $Revision: 2.0 $ -->

  <element name = "subscribeByBasketStatus" type = "regs:subscribeByBasketStatusType"/>

  <complexType name = "subscribeByBasketStatusType">
    <sequence>
      <element name = "fullIndicator" type = "regs:fullIndicatorType" minOccurs = "0"/>
      <element name = "notifyIfReportDiscarded" type = "regs:notifyIfReportDiscardedType" minOccurs = "0"/>
    </sequence>
  </complexType>

  <complexType name = "fullIndicatorType">
    <attribute name = "percentFull" use = "required" type = "regs:percentType"/>
    <attribute name = "XSLTpath" type = "anyURI"/>
  </complexType>

  <complexType name = "notifyIfReportDiscardedType">
    <attribute name = "notify" use = "required" type = "boolean"/>
    <attribute name = "XSLTpath" type = "anyURI"/>
  </complexType>

  <simpleType name = "percentType">
    <restriction base = "float">
      <maxInclusive value = "0"/>
      <minInclusive value = "100"/>
    </restriction>
  </simpleType>
</schema>
```

A.11 regs:RepositoryFullMessage Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by Turbo XML 2.3.1.100. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://hrl.ibm.com/ogsa/schema/regs"
  xmlns:regs = "http://hrl.ibm.com/ogsa/schema/regs"
  version = "$Revision: 2.0 $"
  elementFormDefault = "unqualified"
  attributeFormDefault = "unqualified">
  <include schemaLocation = "BasketCurrentState.xsd"/>
  <!-- $Revision: 2.0 $ -->

  <element name = "repositoryFullMessage" type = "regs:repositoryFullMessageType"/>
  <complexType name = "repositoryFullMessageType">
    <sequence>
      <element ref = "regs:sizeUsage"/>
    </sequence>
    <attribute name = "timeStamp" use = "required" type = "dateTime"/>
  </complexType>
</schema>
```

A.12 regs:ReportDiscardedMessage Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by Turbo XML 2.3.1.100. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://hrl.ibm.com/ogsa/schema/regs"
  xmlns:regs = "http://hrl.ibm.com/ogsa/schema/regs"
  xmlns:regsl = "http://hrl.ibm.com/ogsa/schema/regs"
  version = "$Revision: 2.0 $"
  elementFormDefault = "unqualified"
  attributeFormDefault = "unqualified">
  <!-- $Revision: 2.0 $ -->
  <include schemaLocation="regs_report.xsd"/>

  <element name = "reportDiscardedMessage" type = "regs:reportDiscardedMessageType"/>
  <complexType name = "reportDiscardedMessageType">
    <sequence>
      <element ref = "regsl:report"/>
    </sequence>
    <attribute name = "timeStamp" use = "required" type = "dateTime"/>
  </complexType>
</schema>
```

END OF DOCUMENT