GFD-E.166
INFOD-WG

Ronny Fehling, Oracle Corporation
Steve Fisher, Rutherford Appleton Laboratory
Dieter Gawlick, Oracle Corporation
Raghul Gunasekaran, University of Tennessee
Mallikarjun Shankar, Oak Ridge National Laboratory
Aravind Yalamanchi, Oracle Corporation

March 8, 2010

INFOD 1.0 Implementation – Experience Report

Status of This Document

This document provides information to the Grid community on the implementation experience of the INFOD specification GFD.110. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

This document describes experience in building an INFOD system from the specification document. It also includes changes that should be included in a new version of an INFOD specification.

Table of Contents

1. Introduction

The document describes experience of the authors building an INFOD system according to the GFD.110[1] specification and presents information that would help implementing the INFOD model. Our experience in implementing the INFOD specification has helped the INFOD group gain a better understanding of the model. Therefore this document also identifies changes that should be included in any revision of the specification. The document also addresses certain insights of the INFOD approach gained from our experience, and identifies key details that need to be considered in developing an INFOD system and building applications.

2. Implementation Overview

The INFOD registry was implemented on Oracle database 10g, which offers good XML support. Resources in the registry are stored as XML documents which are parsed and stored in a relational table as XMLtype and varchar datatypes.

2.1 Vocabularies

As described in the INFOD specification, INFOD supports two types of vocabulary definitions:

- *Property Vocabularies* are used to describe user entities in the registry such as publishers, subscribers , and consumers
- *Data Vocabularies* are used to describe data such as records of data sources, events and notification messages.

The choice of vocabulary attributes and what constitutes a property or a data vocabulary depends on the use case on hand. An attribute should be selected if it is going to be used in a constraint; e.g., a constraint identifying records or publishers of interest

From our experience, attributes of property vocabularies should be classified as dynamic, transient, or static. Dynamic means the attribute changes frequently such as the latitude and longitude of a moving vehicle. A transient attribute changes over a certain time or in periodic intervals. Static attributes are ones that do not change or change only occasionally. Choosing a property vocabulary attribute that changes frequently will result in the INFOD registry frequently performing the mutual filtering operation with adverse effect on the performance. It is best to choose dynamic attributes as part of the data vocabulary; static attributes could be either in the property or data vocabulary depending on their function of either describing an entity or describing an event of interest. The handling of transient attributes depends upon the information dissemination needs and the capability of the INFOD registry to deal with updates at a certain frequency.

[1] http://www.ogf.org/documents/GFD.110.pdf

2.2 Constraints

The INFOD model allows three types of constraints – property constraint, data constraint and dynamic consumer constraint. Only the property constraints are evaluated in the registry. As described in the specification, a property constraint is defined in terms of the property vocabulary attributes and since the property vocabulary instance is an XML document, constraints should be defined as XQuery or XPath expressions. In our system, property constraints are defined as XQuery statements using FLWR expressions with the *for, let, where,* and *return* clause. The data constraints and the dynamic consumer constraints are evaluated by the publishers. The data constraints are described in terms of the data vocabulary attributes and the dynamic consumer constraints are based on the property vocabulary attributes. However, for evaluating the dynamic consumer constraints the publisher needs information from the registry, which can be obtained through the getMetaData operation. These two constraints can take any form (XQuery, XPath) and is entirely dependent on the capability of the publisher entity evaluating these constraints. The publisher can be a sophisticated system capable of evaluating these constraints or could rely on the registry to evaluate these constraints. In our implementation, both the constraints are defined as XPath expressions. Simple data constraints are evaluated at the publisher while certain data constraints along with the data are evaluated in the database taking advantage of Oracle features. This request for evaluation is not a part of the INFOD API specification. The dynamic consumer constraints are evaluated at the INFOD registry by calling the getMetaData operation with appropriate arguments. The constraint is evaluated for every new notification message received by the publisher entry.

2.3 Mutual Filtering

	Publisher	Consumer	Subscriber	DataSoure
Publisher	X	✓	✓	X
Consumer	✓	X	✓	✓
Subscriber/Subscription	✓	✓	X	✓
DataSource	X	✓	✓	X

Table 1. Resources to be compared in the Registry

Table 1 details which entity can define constraints on which other entities in the INFOD registry and accordingly the matching procedure should enforce all these possible comparisons. The row headers indicate the constraints defined by the entities and the column headers indicate the instances of the entities in the registry. A publisher instance means the publisher entry, publisher property vocabulary instance and publisher data source entries. When a consumer defines a constraint it would be applicable to publishers, subscribers and data source instances in the registry.

Some rules to be followed while performing mutual filtering (explained in sec 3.3)

- A subscription binds publishers/data sources and consumers. In the absence of a subscription publishers/data sources and consumers are NOT matched; there is no information flow.
- Any create/replace/drop in the INFOD registry will trigger the matching procedure and activate evaluation of all relevant constraints in the registry.
- All constraints relating a publisher and a consumer need to be satisfied for notification to occur. If a subscription relates a publisher and a consumer, and the consumer satisfies the publisher's constraint but the publisher does not satisfy the consumer's constraint then the publisher and the consumer are not matched to each other.
- If a subscription does not specify any property constraint then all publishers and consumers in the registry are related through the subscription and they should be associated only based on their individual property constraints.
- A data source entry inherits all the property constraints of the publisher it references. For a data source to be matched with a consumer, the data source entry's property constraint and the referenced publisher's property constraint should be satisfied. For example: if a single publisher contains multiple sensors, then each sensor is identified as a data source entry in the registry and each sensor could have a property constraint. This constraint is in addition to the publisher entry's property constraint and both of these property constraints need to be satisfied.
- Similarly, the subscription inherits all the property constraints of the subscriber that created the particular subscription.

3. Implementation Details

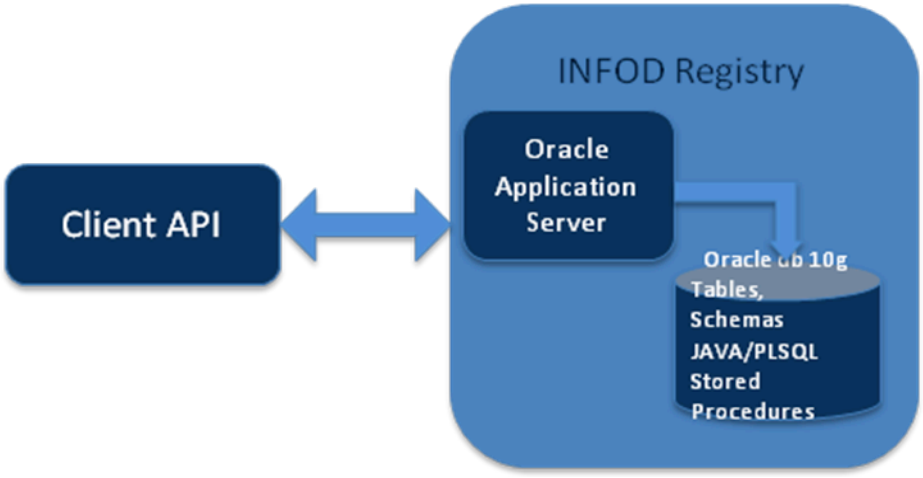

Figure 1. INFOD Registry Implementation Model

The system was developed as a client-server model, which enabled testing, evaluating and building applications. Figure 1 shows the system architecture. The server is the INFOD registry accessible as a

web server through SOAP requests, and the client supports an API for interaction with the registry. The server side system consists of the Oracle database and Oracle Application Server (OC4J - Oracle Containers for J2EE). All INFOD interfaces are available as web services capable of being communicated using SOAP messages, as defined in the INFOD specification. Oracle Application Server OC4J is being used to host the INFOD web services, which connects and communicates with the database. Oracle JDeveloper tool is used to generate INFOD web services from PL/SQL functions in the database and host it on OC4J. The PL/SQL functions define the various INFOD interfaces performing database operations (insert, update, delete) and triggering actions (matching, notification) in the database.

3.1 Registry Structure

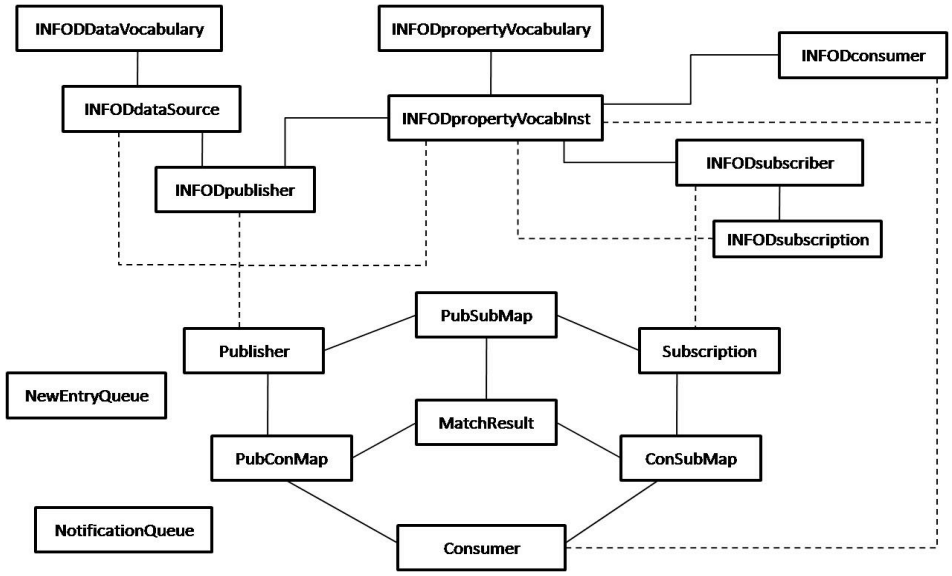

Figure 2. INFOD Registry Tables in Database

Figure 2 illustrates the INFOD tables in the database, the utility tables and their structure. The tables prefixed with 'INFOD' store information on entities registered through the INFOD interfaces as XML documents - as defined in the specification document. The other tables in the database store data in a form suitable for matching and for generating notifications. Figure 3 provides more details on the tables prefixed with 'INFOD'. The *publisher* table in figure 2 is linked to tables INFODpublisher, INFODdatasource and INFODpropertyVocabInst. The publisher table is used for matching, where a publisher is represented with information from its entry and the referenced property vocabulary instance. If a single publisher entry has multiple property vocabulary instances then the publisher table would have multiple entries with each one referring to a different instance. All these entries would have the same property constraints since they refer to a single publisher entry. A data source entry is identified by a separate entry in the publisher table associated to its corresponding property vocabulary instance. The data source entry's property constraint also includes the property constraint from the publisher being referenced in addition to its property constraint. Similarly the *consumer* table combines the consumer entry and property vocabulary instance as a single entity to enable matching. The

subscription table associates the subscriber's entry, including the subscription defined by the subscriber and the property vocabulary into a single entry.

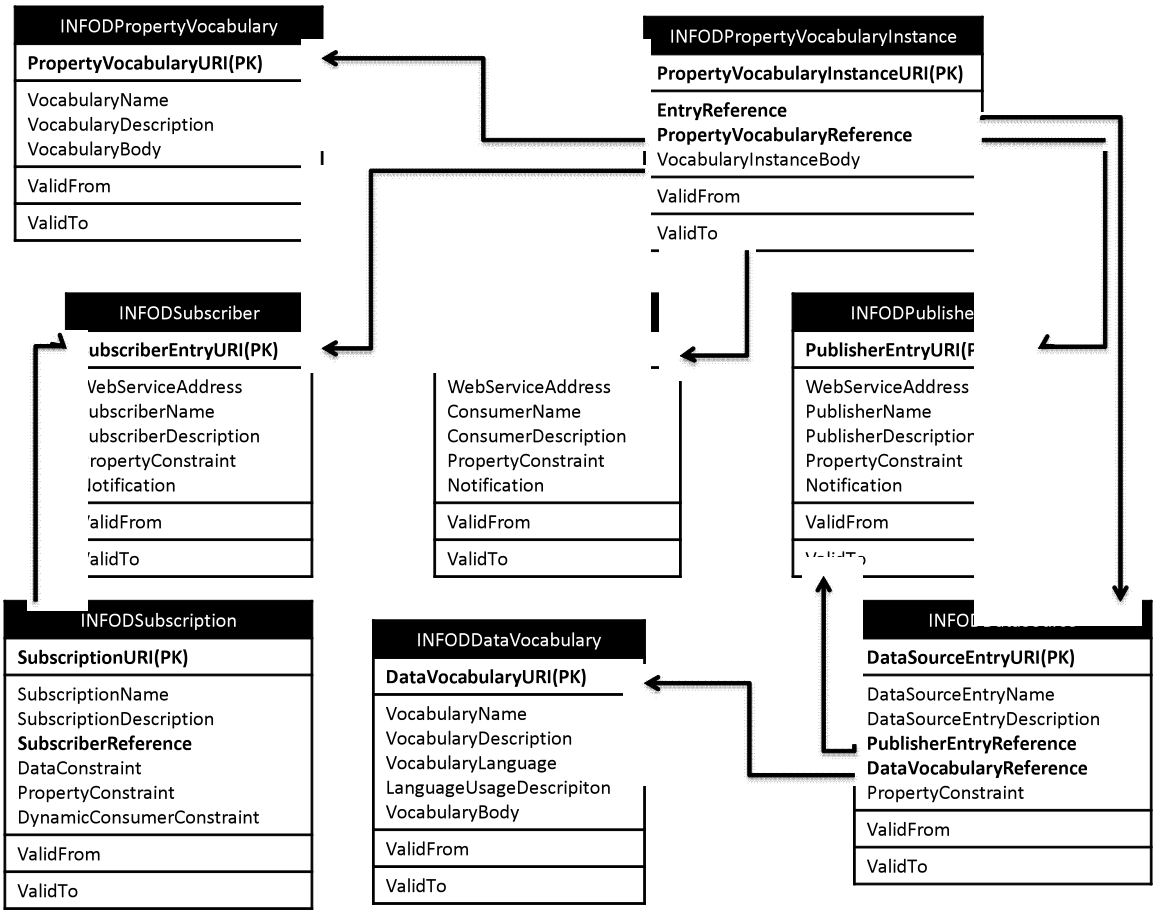

Figure 3. Registry ER Model

Figure 3 shows the structure for the tables that record INFOD metadata. The name for each table is indicated in the grey background. Each table in the figure has four columns; the first column is the primary key element, the ValidFrom and ValidTo fields are the third and fourth columns. The rest of the elements in the respectively tables are in a single XMLType column corresponding to an XML schema. For example, the consumer web service address (WSA), name, description, property constraint and notification details are stored as an XML document in an XMLType column. In the current implementation, this XML content is the body of the SOAP request messages from a consumer, when a user creates an entry in the INFOD registry. The primary key is a unique URI identifying the information in the registry. The ValidFrom and ValidTo fields are used for managing the lifecycle of the entities in the registry. When a new entry is created the ValidFrom field is set to the current time and when any drop operation is requested the ValidTo field is set.

The tables NewEntryQueue and NotificationQueue in Figure 2 represent queues. Every new entry in the registry is added to the NewEntryQueue for the job handler to be matched with other entities in the

registry. Once the mutual filtering process is done the entry is removed from the NewEntryQueue table. After performing the mutual filtering, matched entities need to be notified of the results through web service notification. The entity to be notified with the relevant subscription causing the notification is logged in the NotificationQueue table. A user scheduled job handler reads from the notification queue table and sends a web service notification message based on the entities role as a publisher, consumer or subscriber.

3.2 Registry Operation

The registry supports three kinds of operations: create, replace and drop. The replace operation is restricted only to certain entity information in the registry. Figure 4 illustrates a typical sequence for a consumer entry being created, replaced, and dropped.

When a new entry is created a unique URI (Universal Resource Identifier) is generated, which uniquely identifies every resource in the registry. When a consumer entry is created it triggers a java procedure call for reorganizing the information and parsing the constraints enabling efficient matching within the registry. As noted previously the consumer can define constraints on publishers and subscribers of interest. In this case, the consumer's property constraint is parsed into two constraints one that is applied on the publisher metadata and the other constraint that is applied on subscriber metadata. The constraints need to be parsed such that they can be evaluated using expression filters. If the consumer does not have a constraint on a particular entity, say the subscriber, the constraint is represented as "*1=1*", which matches all entities. Apart from the constraints, other information in the entry is also extracted from the XML doc and stored in columns of varchar type for easy retrieval. The information concerning the new entry is updated in the NewEntryQueue table for mutual filtering in the registry. A job handler repeatedly checks for new entries in the NewEntryQueue table and performs the match operation, explained below. Once the consumer is matched with publishers and subscribers in the registry a notification message is sent to all the matched entities. The entities to be notified and the notification message that needs to be sent are queued up in the NotificationQueue table. Another job handler checks this table periodically and sends the web service notification messages. The need for having a separate table for notification is to avoid network delays inhibiting procedure runs in the registry.

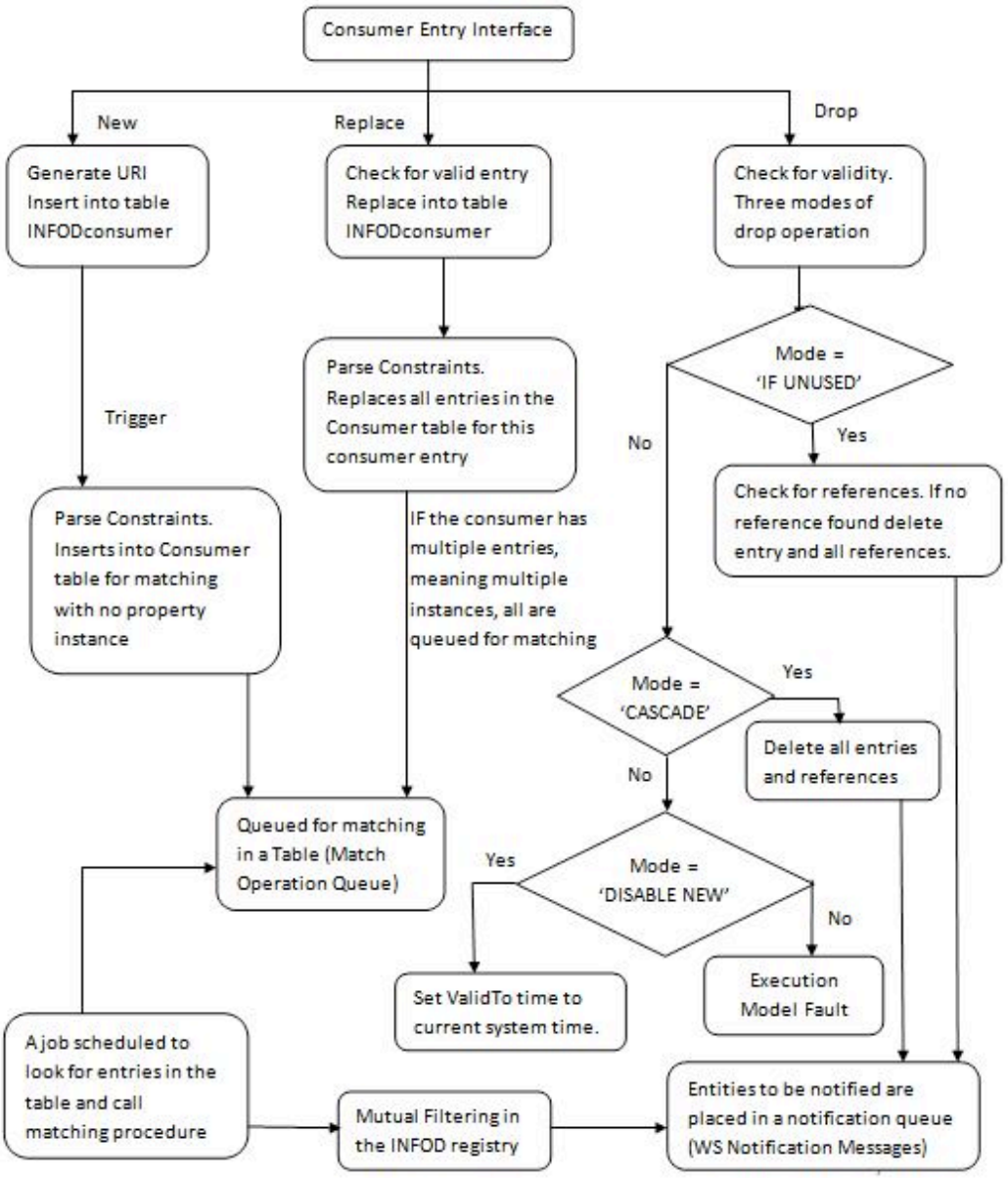

Figure 4. Registry Operation Sequence for Consumer Create/Replace/Drop

When an existing entry is being replaced, first the registry checks if such entry exists based on the resource URI. In the current implementation, the replace operation is treated as deleting the existing entry and then creating a new entry with the same URI. The entity information is parsed and the relevant tables are updated. Similar to operations performed when a new entry is created, the replaced entity information is queued for matching and notification. However, while replacing an entity we need to check if the entry has multiple instances then all the resources associated with the entity needs to be queued for mutual filtering.

The drop operation does not call for mutual filtering but entities needs to be notified when an association is no longer valid. Since, the results of mutual filtering is stored in the registry identifying

which entity is no longer associated with the entry being dropped is a lookup on the results table. The registry first checks if the entity to be dropped is present in the system. The registry supports three kinds of drop operations. *If Unused* mode, invoked when the user wants to drop an entry only when it is not being associated with any other resource in the registry. The *Disable New* option allows the user to restrict the entity from forming new associations with other entities in the system. This is done by updating the ValidTo field associated with the entities, which is checked whenever an entity is being matched. *Cascade* mode deletes the entry and deletes all the associations the system has in the registry.

3.3 Mutual Filtering

Entity descriptions, property constraints, and subscriptions comprise the metadata information that INFOD uses to associate entities within a community. We refer to this model as mutual filtering; the actual implementation is realized with a three-way join across publisher/datasource, consumer, and subscriber/subscription information. The process of mutual filtering attempts to satisfy the following conditions: (a) the publisher and the consumer should satisfy constraints for the subscription and the subscriber which created the subscription, (b) the subscriber and the consumer should satisfy constraints imposed by the publisher, and (c) the publisher and the subscriber should satisfy constraints expressed by the consumer. A publisher and a consumer are associated with each other only when all the conditions are met.

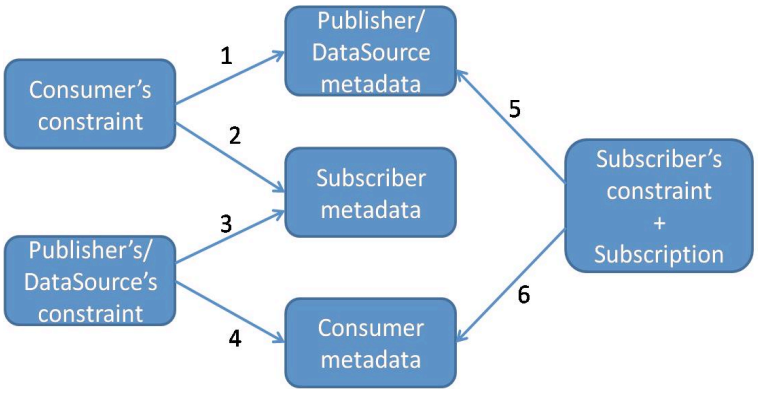

Figure 5. Mutual Filtering Evaluations

Figure 5 illustrates the computations involved in mutual filtering, and the numbers on the arrow indicate the number of computations involved (total six and the sequence of evaluation might vary). We consider the three metadata sets describing the entities – publishers/datasources, consumers, and subscribers. We must constrain the entities with three sets of property constraints defined by the publishers/datasources, consumers, and subscribers/subscriptions. For a publisher and a consumer to be matched, all the six computations must be evaluated. We use the Oracle Database *Expression Filter* feature to index and evaluate the constraints captured as XPath expressions in INFOD metadata tables. The XPath expressions indexing using Expression Filter can be evaluated using an XML document that is transient or persistently stored in the database using XMLtype data type. The property vocabularies of

publishers, consumers, and subscriptions, which are captured as XML data, are used to evaluate the corresponding constraints as described in section 4.1.

The mutual filtering operation can be performed in a single SQL select statement by specifying all the six computations in the WHERE clause. This query may be executed for each incoming publisher, consumer, or subscription. It may be observed that for the addition of a specific publisher, consumer, or subscription there is only the need to evaluate the constraints that exist between the specific entity and other two entity types. Hence, the mutual filtering operation can be performed with fewer joins and computations if some of the mappings between the entities are pre-computed and materialized.

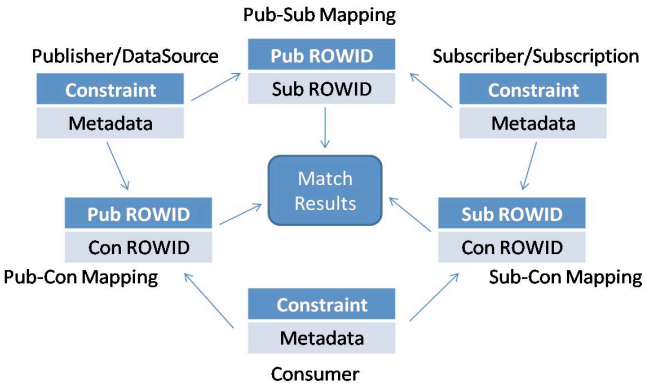

Figure 6. Mutual filtering in the INFOD registry

The optimization is based on the following observations

- The initial approach of using a single SQL statement for mutual filtering does many re-evaluations for every new entry, which could be avoided if the results of previous evaluations are stored.
- When a new publisher is created, the new publisher needs to be matched only with consumers and subscriptions, and the knowledge of which consumer and which subscription should be associated may be pre-computed.
- Doing a pair-wise association between publishers, consumers, and subscriptions may reduce the number of computations when a new entry is created.
- It was faster to do a three-way join (mutual filtering) using ROWID's in the SQL-WHERE clause rather than evaluating constraints across three tables.

Figure 6 illustrates the tables in the database with the mapping tables representing the intermediate stage. The publishers and consumers are matched on their constraints and metadata, and the results are stored in the pub-con mapping table. Similarly, the sub-con mapping table has the list of the subscription and consumer pairs that can be associated and the pub-sub mapping table has the publisher and subscription pairs. With the intermediate steps, the mutual filtering function is a three-step process. Say, a new publisher joins the INFOD system. The first step is in finding all the consumers

the new publisher can be matched with. For this we evaluate the new publisher's constraint on all consumers' metadata and all the consumers' constraints are evaluated on the new publisher's metadata. The result is the new publisher being matched with consumers and the results are updated in the pub-con mapping table. Similarly, the second step involves finding all the subscribers/subscriptions the new publisher can be mapped with and updating the pub-sub mapping table. The final step is a three-way join across the mapping tables for determining the subscription-publisher-consumer tuples. These tuples give us the desired information on which publisher and which consumer are associated and through which subscription. However, the final step depends on the number of entries existing in the system, which determine the number of rows in the mapping table and correspondingly affect the computation time.

More work is necessary to understand how scalable INFOD is and whether or not an implementation can be generic or has to be tailored for each Use Case.

3.4 Notification Messages

Publishers, consumers, and subscribers get notified by the INFOD registry. A publisher is notified of the matching subscriptions and the consumers bound to those subscriptions. Similarly, a consumer is notified of the matching subscriptions and publishers, and the subscriber of the matching consumers and publishers bound by a specific subscription. The INFOD specification document provides details of the format of the notification messages.

Notification messages are sent when any change to the information in the registry results in a change to the association between publishers, consumers, and subscribers (or subscriptions). If a consumer's property constraint has changed, this would require the re-evaluation of the consumer's constraint with that of the information in the registry. By the end of the process new associations have formed and existing associations would have been broken due to the change in the consumer's property constraint. For example, if a consumer C1 was associated by subscription S1 to publishers P1, P2, P4 and C1 was also associated by subscription S2 to publishers P1, P3, and P4. If consumer C1's constraint has changed resulting in C1 not associated by S1 and is associated by another subscription S3 to publishers P1, P4. Then publisher P1 is notified of the changes in association though S1 and is also notified of the new association with C1 through S3.

The web service notification message to be sent to entities is created in a PL/SQL procedure using a DOM XML parser. The process of sending notification messages is handled by a java procedure called within the PL/SQL procedure. As described earlier, the entities to receive the notification messages are queued in a NotificationQueue table and are handled individually by a job handler. This is to avoid the delays in sending the notification messages affecting the completion of the mutual filtering process and performing other registry functions. From the above example a change in C1's constraint results in sending a number of notification messages, which might be time consuming depending on the network connectivity.

3.5 Garbage Collection

This procedure is for removing resources in the registry, which have been dropped by 'DISABLE NEW' execution mode. The procedure is run periodically to check for resources, which are no longer associated with any other resources in the registry. The procedure looks for entries, which have a ValidTo field time stamped and looks for any references associated to the resource.

3.6 Client API

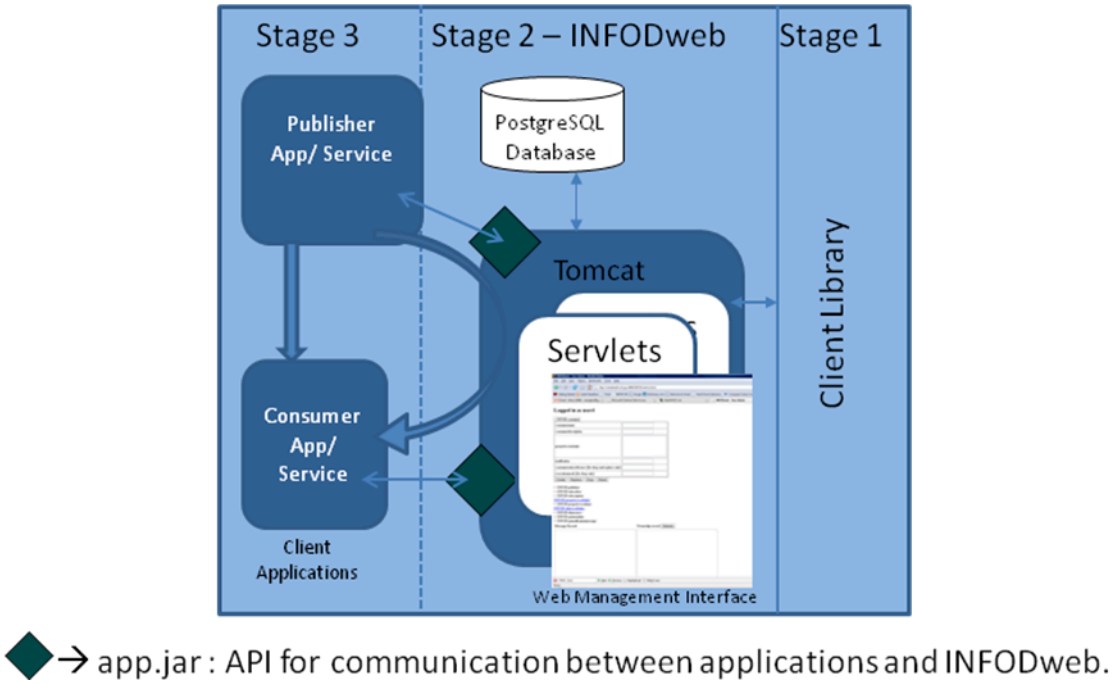

Figure 7. Client Implementation

The client module was developed to interact with the INFOD registry. The client environment was developed as three components/stages, as shown in the figure 7. The first component is the client library coded in java for interaction with the INFOD registry service. The library defines all the operations as function calls that the user can call for interacting with the registry. The function call supports all interfaces that are defined in the INFOD specification document. Application developers who would like to interact with the INFOD system can use this software package for developing custom applications interacting with the registry. The second component is a web interface for accessing the registry. Users can use this web interface to create, replace, and drop resources in the registry. First time users are allowed to create an account signing up for a username and password. The next time the user returns, uses the login information to view the resources the user had created and receives all the notification messages received when the user was not logged in. This interface helps users build applications and understand how the INFOD registry associates users based on the resources they had created in the registry. The user information and the associated resource URI's are stored in a PostgreSQL database at the client side. The third component is for binding user applications to the INFOD system. A standard set

of java interfaces have been identified, which a user application can call for communicating with the existing client setup. The user through the web interface uploads a jar package while creating an entry in the registry through the web interface. The jar file for publishers would help forward the notification messages to the user applications and any notification messages generated by the application can be directed using the existing client module. Similarly, the jar file for subscribers and consumers directs notification messages to their respective application modules.

4. Changes to the INFOD specification

Appendix A lists the errata that should be applied to the INFOD specification version 1, GFD 110 document. Some of the major changes are

- The specification document had some namespace discrepancies; http://www.ggf.org/INFOD/ shall be the recognized namespace.
- In the current specification, the data source entry and property vocabulary instance have no replace operation. However, replace operation for data source entry and property vocabulary instance is allowed and the interface specification shall be added in the next version of the specification.
- Each resource in the INFOD registry shall be identified by a unique URI (Universal Resource Identifier), any reference to EPR (End Point Reference) in the document shall be referred to as URI.
- **The GetNotificationMessages Operation**

The GetNotificationMessages operation provides the notification messages that are sent by the registry to publishers, subscribers and consumers assuming notification has been requested.

The GetNotificationMessages is a special case of the GetMetaData operation.

The format of the request message for a GetMetadata operation is:

```
<infod:GetNotificationMessages>
<infod:EntryReference>
   {xsd:URI}
</infod:EntryReference>
<infod:SubscriptionReference>
   {xsd:URI}
</infod:SubscriptionReference> ?
</infod:GetNotificationMessages>
```

The elements of the GetNotificationMessages message are further described as follows:

/infod:EntryReference

This element must be a valid URI of either a publisher, subscriber, or consumer entry.

/infod:SubscriptonReference

This element must be a valid URI of a subscription. Only messages related to the specified URI will be create.

If this element is omitted all subscriptions will be evaluated. Notification messages will only be created for those subscriptions for which a notification message would be send.

A WS-Addressing Action header with the value http://www.ggf.org/INFOD/INFODRegistry/GetMetaData MUST accompany the message.

INFOD Registry Response

The response of the INFOD registry is:

```
<infod:GetNotifcationMessagesResponse
<infod:GetNotificationMessagesResult>
infod:PublisherNotification
</infod:GetNotificationMessagesResult> * |
<infod:GetNotificationMessagesResult>
infod:ConsumerNotification
</infod:GetNotificationMessagesResult> * |
<infod:GetNotificationMessagesResult>
infod:SubscriberNotification
</infod:GetNotificationMessagesResult> *
<infod:GetNotificationMessagesResponse>
```

The content of infod:GetNotificationMessagesResult MUST be structured according to the type of entry for which the information was requested:.

- If the EntryReference represents a publisher, the message represents a PublisherNotification.
- If the EntryReference represents a subscriber, the message represents a SubscriberNotification.
- If the EntryReference represents a consumer, the message represents a ConsumerNotification.

One of the following fault codes MUST be sent if the operation fails:

- GetNotificationMessagesAuthorizationFailure: User not authorized to use the operation at this INFOD registry
- MissingRequiredParameterFault: A required parameter was not specified
- InvalidURIFault: The URI does not define a proper resource

The message MUST be structured according to the WS-Base Faults specification. For examples using SOAP, see the SOAP v1.2. Base Fault Spec (see http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf).

5. Conclusion

The experience helped the group understand some of the shortcomings of the specification. A few of them have been addressed in the errata and any future version of the specification should address these and add more features to the existing model.

6. Author Contact Information

Ronny Fehling
Oracle Corporation
600 Blvd. de Maisonneuve Ouest
Montreal
Quebec H3A 3J2
Canada

Steve Fisher
Rutherford Appleton Laboratory
Chilton
Didcot
Oxon
OX11 0QX
UK

Dieter Gawlick
Oracle Corporation
500 Oracle Parkway
Redwood Shores
CA 94065
USA

Raghul Gunasekaran
University of Tennessee
1508 Middle Drive
Knoxville TN 37996
USA

Mallikarjun Shankar
Oak Ridge National Laboratory
OakRidge
TN 37831
USA

AravindYalamanchi
Oracle Corporation
1 Oracle Drive
Nashua NH 03062
USA

7. Intellectual Property Statement

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

8. Disclaimer

This document and the information contained herein is provided on an "As Is" basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

9. Full Copyright Notice

Appendix A: Errata

1. Lines 172 and 221 - Dynamic consumer constraints have to be handled by the publishers.
2. Line 171-173 - add the following explanation: 'The registry will use static constraints to determine which consumers are to be notified. This list can be restricted by dynamic consumer constraints, which, if present, have to be processed by the publisher for each message.'
3. Lines 383 and 491: ', data sources' has to be deleted
4. Line 384 and all other explanations for infod:PropertyConstraint: The text "an XQuery" has to be replaced by "A sequence of namespace declarations following the XQuery prologue entry syntax http://www.w3.org/TR/xquery/#id-namespace-declaration and local function definitions following the XQuery prologue entry syntax http://www.w3.org/TR/xquery/#dt-udf and let statements http://www.w3.org/TR/xquery/#id-for-let followed by a single WHERE clause as defined in XQueryhttp://www.w3.org/TR/xquery/#id-where. Each namespace and local function must be terminated by a semicolon. In any namespace definition the URILiteral must exactly match the URI of a property vocabulary. In each WHERE clause the namespace must always be specified to match a property vocabulary referenced in a namespace declaration. The following variables are predefined $con, $subr, $subn $pub and $dse: the property vocabulary instance of a consumer entry (or subscriber, subscription, publisher or data source entry).
5. After line 413: Syntax error in property constraint
6. Line 757 and all other explanations for ExecutionFaultMode have to be replaced by: Invalid ExectionMode provided
7. Lines 1122-1124 - the description has to reference /infod:status as in lines 894 and 895
8. Line 1618 and throughout, state that GetMetaData call is complete XQuery 1.0
9. Line 1627 has a typo: replace *specify specific a* by *specify a specific*
10. Line 1628 should be removed
11. Before line 1632 add 'To identify all resources in an INFOD registry - fn:collection('$$infodResources')'
12. Line 1632 has to be replaced by: 'To identify all publishers - fn:collection('$$infodPublishers')' The equivalent changes need to be made for lines 1633 - 1640 - as described in the Extended Specification in section 2.7.1
13. After line 1639 add: 'To identify all instances of a specific property vocabulary - fn:collection('epr') where epr is the EPR assigned by the INFOD registry'. An error code has to be added after line 1656 'InvalidEPRFault - An EPR in the XQuery does not reference a valid INFOD registry vocabulary.'
14. Line 1659 after this line add 3 new calls: GetPublisherNotificationMessages, GetSubscriberNotificationMessages and GetConsumerNotificationMessages the input in each case is the URI (ex EPR) of the Publisher, Subscriber or Consumer entry for which the information is desired. The response is the set of messages that would be sent as notifications if the Publisher, Subscriber or Consumer entry had just been created.
15. Line 1669 should be removed
16. Line 1650-1651 should be extended by: as defined in 'XSLT 2.0 and XQuery 1.0 Serialization.' (the reference http://www.w3.org/TR/xslt-xquery-serialization/ should be added as a footnote)
17. Line 1681: after this line ConsumerReference of type wsa:ReferenceType has to be added. This field has to be defined at least once. The explanation has to be added after line 1702: *The endpoint reference of all consumers that have to receive the message and are related to the WSAReference used to send this message*. Additionally, the XML and the WSDL appendices have to be updated.

18. Line 1696: 'VocabularyAssociation' has to be replaced by 'DataSource'
19. Line 1771: the ConsumerEntryReference (at least one occurrence) has to be complemented by the ConsumerReference, the external EPR of the consumer (exactly one occurrence). These 2 fields have to become part of a complex element called ConsumerList. ConsumerList (at least one occurrence). The text starting on line 1788 has to be updated accordingly. Additionally, the XML and the WSDL appendices have to be updated.
20. Line 1776: after this line a field infod:DataSource has to be added. This field specifies to which data source or data sources the following data constraints have to be applied to (at least one occurrence). An explanation has to be added after line 1796. Additionally, the XML and the WSDL appendices have to be updated.
21. In the intro and throughout, state that all resources are represented as individual documents
22. In the intro and throughout, state that the URI of the document representing the resource is returned (rather than EPR) when a resource is created and that if this URI is U then document-uri(doc(U)) == U
23. Still need to define what data and dynamic constraints look like - at least for XML - there are concerns about how to specify for non-XML
24. Throughout the document, the INFOD namespace shall be 'www.ggf.org/INFOD' and all the WS-Address action header URI's shall be prefixed by 'www.ggf.org/INFOD' instead of 'www.ogf.org/infod'.