GWD-I

Category: Informational
INFOD-WG

Vijay Dialani, IBM Corporation
Stephen Davey, Edinburgh University
Abdeslem Djaoui, Rutherford Appleton Laboratory
Ronny Fehling, Oracle Corporation
Steve Fisher, Rutherford Appleton Laboratory
Dieter Gawlick, Oracle Corporation
Christopher Kantarjiev, Oracle Corporation
Cecile Madsen, IBM Corporation
Susan Malaika, IBM Corporation
Shailendra Mishra, Oracle Corporation
Mallikarjun Shankar, Oak Ridge National Laboratory
July 19, 2006

INFOD (Information Dissemination) Base Use Case Scenarios

Status of This Memo

This memo provides information to the Grid community motivating scenarios for the Information Dissemination working group. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

INFOD (Information Dissemination) is a working group in GGF focusing on publishing and consuming data within a grid or distributed system infrastructure. A variety of commercial and scientific scenarios are introduced in this document to illustrate how the INFOD base specification interfaces are used. They may also provide a source of materials for the data and information architecture activities in the OGSA working group. INFOD patterns are also included in this document.

Contents

1 Introduction

The goal of this document is to define use cases to illustrate the use of the INFOD base specification interfaces [INFOD] in a variety of environments. The document ends with appendices on use case patterns and ways of accessing the INFOD registry.

The following use cases are included:

- NextGRID Graphical Animations Use Case
- Car Dealer Use Case
- Sensor Networks Use Case
- 3rd Party Delivery of Query Results Use Case

You will note that each use case in its general form follows the same structure:

1. Define a community which is achieved by defining vocabularies in the INFOD registry.

2. Create instances which are achieved by defining publishers, subscriptions, and consumers associated with particular vocabularies in the INFOD registry.

3. Determine whether to publish which publishers achieve by checking if there are any relevant subscriptions in the INFOD registry.

4. Publish which publishers achieve by producing messages.

4. Consume which consumers achieve through the consume interface.

When you read the use cases, please keep in mind that in most scenarios, portions of the use cases would not be repeated, e.g., defining communities (defining vocabularies) would be done once for each use case. Vocabularies are often re-used.

2 NextGRID Graphical Animations Use Case

2.1 Introduction

This use case is based on the Digital Media use case described in the NextGRID Vision and Architecture White Paper, http://www.nextgrid.org/download/publications/NextGRID_Architecture_White_Paper.pdf.

Nowadays, almost all films and commercials use computer graphics animations to implement the special effects that the artists want to depict on the screen. Designers can use several software applications for creating 3D scenes like 3D Studio Max and Maya. These applications can build a 3D environment or just a single scene and render it. The large number of objects, textures, light sources and effects, like shiny surfaces and fog, is a factor that limits the design of a scene due to the increased computational effort. The best solution is to combine the summed power of many single PCs to accomplish the job with the existing software, thus combining the advantages of a powerful computer cluster with the “single PC” way.

The designer develops the job on a single PC with the client’s instructions followed as well as possible to ensure that the final result is the expected one. A close online collaboration between the client and the designer is required in order to obtain results close to the client’s needs.

KINO, a leading producer of TV commercials and films in Greece, anticipates this novel business model can be supported by a Grid enabled rendering infrastructure that can handle not only the in-house production of urgent jobs and small jobs, but also large tasks with a task based negotiation. This negotiation, on the outsourcing of large tasks, has such parameters as the deadline, the complexity of the task, the number of frames and the total computational time needed.

2.2 Actors

In this use case the actors are the Designers, Reviewers, Bosses, Rendering Services and the INFOD Registry. The Designer has submitted animation rendering jobs to the Grid, and the Designer and Reviewers are interested in knowing when those jobs have finished.

The Designer and Reviewer want to be informed as soon as each animation is completed.

The Boss wants to know which Designers are available between certain dates.

The Publisher is the Rendering Service.

The INFOD Registry manages the vocabularies and subscriptions etc.

2.3 Scenarios

In these scenarios the designer submits jobs (i.e., animated scenes to be rendered) to the rendering service (compute cluster). On submitting the rendering job, parameters such as time limit, complexity, number of frames in the scene, type of rendering software to use, data transport protocol etc may all be specified. But how the job is submitted to the rendering service and the animation data transport are outside the scope of this document.

These scenarios will just concentrate on the notification of job completion to the designer and reviewers, within the INFOD framework.

134 It is also assumed that the Rendering Services have already been created and their addresses are
135 known.

136 2.3.1 Vocabularies

137 The first stage is to define the appropriate vocabularies for the Animators, Reviewers, Bosses &
138 Rendering Services community:

Rendering Service Vocabulary Components	Comment
Organisation_Name	Name of the organisation hosting the rendering service.
Organisation_Location	Address of the organisation hosting the rendering service.
Organisation_Email	General e-mail address of organisation.
Organisation_Website	URL.
BBB_rating	A number between 1 and 5.
Security_Policy	Security policy.

139 **Table 2-1: Media Organisation Vocabulary.**

Rendering Service Vocabulary Components	Comment
Service_Name	Name of rendering service.
Service_Address	URL.
Organisation_Name	Name of the organisation hosting the rendering service.
Organisation_Location	Address of the organisation hosting the rendering service.
BBB_rating	A number between 1 and 5.
Rendering_Software	List of rendering software products available, e.g. 3D Studio Max, Maya.

140 **Table 2-2: Rendering Service Vocabulary.**

Designer Vocabulary Components	Comment
Employee_Name	Name of the animator/designer, reviewer or boss.
Employee_Email	E-mail address.
Employee_Type	Designer, Reviewer or Boss.
Organisation_Name	Name of the organisation hosting the rendering service.
Organisation_Location	Address of the organisation hosting the rendering service.
Hourly_Rate	Hourly rate, for example for designing animations.
Availability	Dates and times when the employee is available.

141 **Table 2-3: Employee Vocabulary.**

Animation Vocabulary Components	Comment
Designer_Name	Name of the designer that submitted the animation for rendering.
Rendering_Job_Name	Name for the submitted rendering job.
Project_Name	Name of the project (e.g. film or commercial perhaps).
Number_of_Scenes	Number of scenes to be rendered (i.e. some measure of the complexity of the animation).
Rendering_Software	Rendering software used, e.g. 3D Studio Max or Maya etc.
Job_Time_Limit	Maximum completion time for the rendering job.
Job_Start	Actual start time for the rendering job.
Job_End	Actual end time for the rendering job.
Job_Status	Status of the rendering job, e.g. Accepted, Rejected, Submitted, Started, Completed, Reviewed, Passed, Failed, Exceeded Time Limit.
Animation_EPR	Location (EPR) of the resulting stored animation.

142 **Table 2-4: Animation Vocabulary.**

143 ## 2.3.2 Subscriptions

144 The next stage is to create relevant subscriptions:

Subscription Entity **INFOD Component**	**Values**
infod:Type	Subscription
infod:Name	Find a Rendering Service
infod:Description	Watch available Rendering Services
infod:Data_Constraints	Rendering_Service_Inventory: SHOW (Service Name, Organisation, Website) FOR Rendering_Software = 3D Studio Max
infod:Property_Constraints	Rendering Service Organisation: BBB_Rating > 2

145 **Table 2-5: Designer finding a rendering service subscription.**

Subscription Entity **INFOD Component**	**Values**
infod:Type	Subscription
infod:Name	Find completed animations
infod:Description	Watch for completed animations
infod:Data_Constraints	Rendering_Jobs_Inventory: SHOW (Rendering_Job_Name, Animation_EPR) FOR Designer Name = designer, Project = Toy Story 3, Status = Completed
infod:Property_Constraints	*None*

146 **Table 2-6: Designer monitoring animation jobs subscription.**

Subscription Entity INFOD Component	Values
infod:Type	Subscription
infod:Name	Find animations for review
infod:Description	Watch for animations requiring review
infod:Data_Constraints	Rendering_Jobs_Inventory: SHOW (Designer Name, email, job name, Animation_EPR) FOR Project = Toy Story 3, Status = Completed
infod:Property_Constraints	*None*

147 **Table 2-7: Reviewers monitoring animation jobs subscription.**

Subscription Entity INFOD Component	Values
infod:Type	Subscription
infod:Name	Find a designer.
infod:Description	Watch for available designers.
infod:Data_Constraints	Employee_Inventory: SHOW (Name, Email) FOR Type = Designer, Rendering_Software = 3D Studio Max, Hourly rate < x £ per hour, Availability ∩ [25/12/2005:27/12/2005] = true
infod:Property_Constraints	Employees Organisation: BBB_Rating > 3

148 **Table 2-8: Boss watching for available designers' subscription.**

149 2.3.3 Joining the media community

150 Each appropriate Boss needs to enroll themselves and the Designers and Reviewers that they
151 manage into the INFOD Registry. They (or an Administrator at one of the organizations) also need to
152 add the relevant vocabularies and subscriptions.

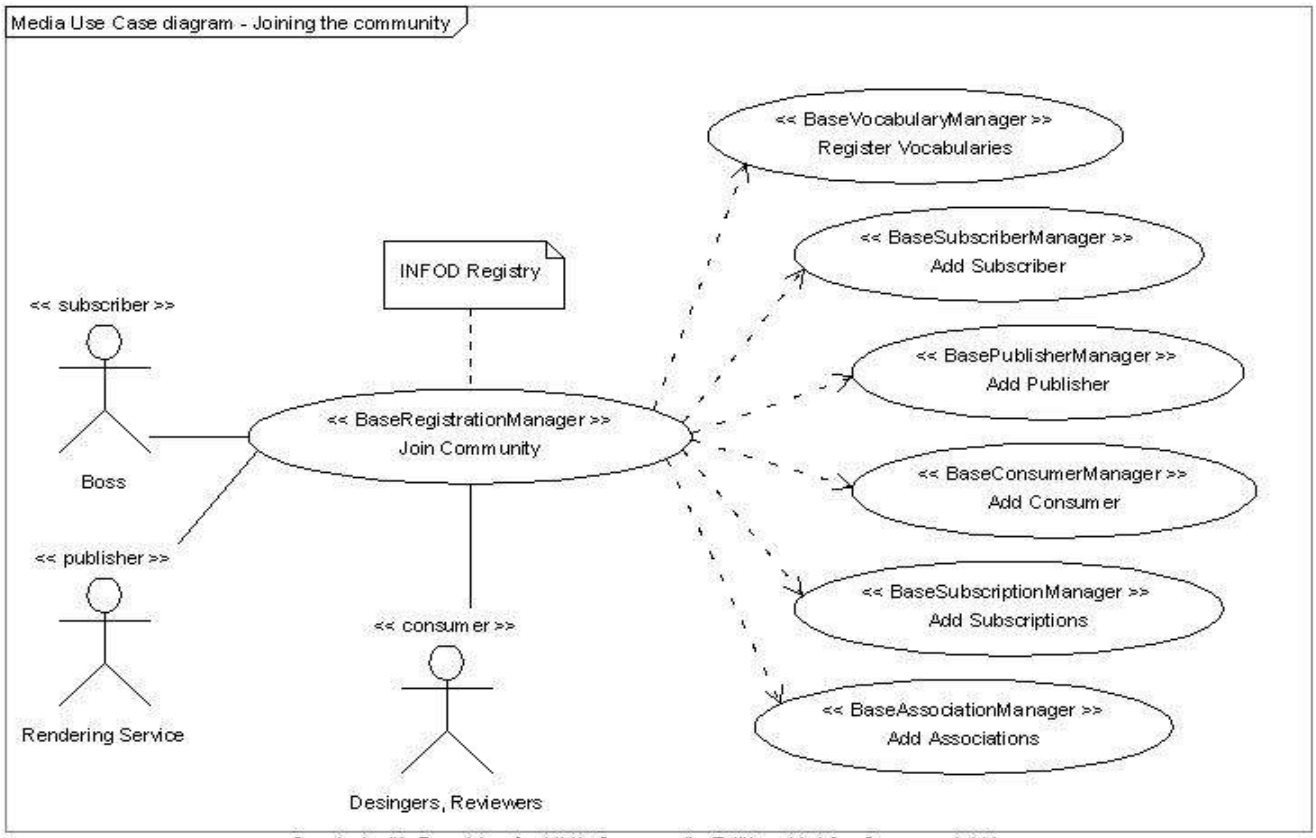


Figure 2-1: Use case diagram for joining the media community.

A summary of these steps is as follows:

1. Boss (or Organization Admin) registers the vocabularies with the INFOD Registry.
2. Rendering Service adds itself as a publisher.
3. Boss adds himself into the INFOD Registry as a subscriber.
4. Boss adds their designers and reviewers as consumers.
5. Boss creates associations between the publisher and the data vocabulary.
6. Boss creates instances of the relevant property vocabularies for the publisher, subscriber and consumers, identifying for example the designer and reviewer from roles and job characteristics defined in their employee property vocabulary.
7. Boss adds the relevant subscriptions to the INFOD Registry.

2.3.4 Designer and Reviewer Subscription

Designer and Reviewer want to be informed as soon as each animation is completed.

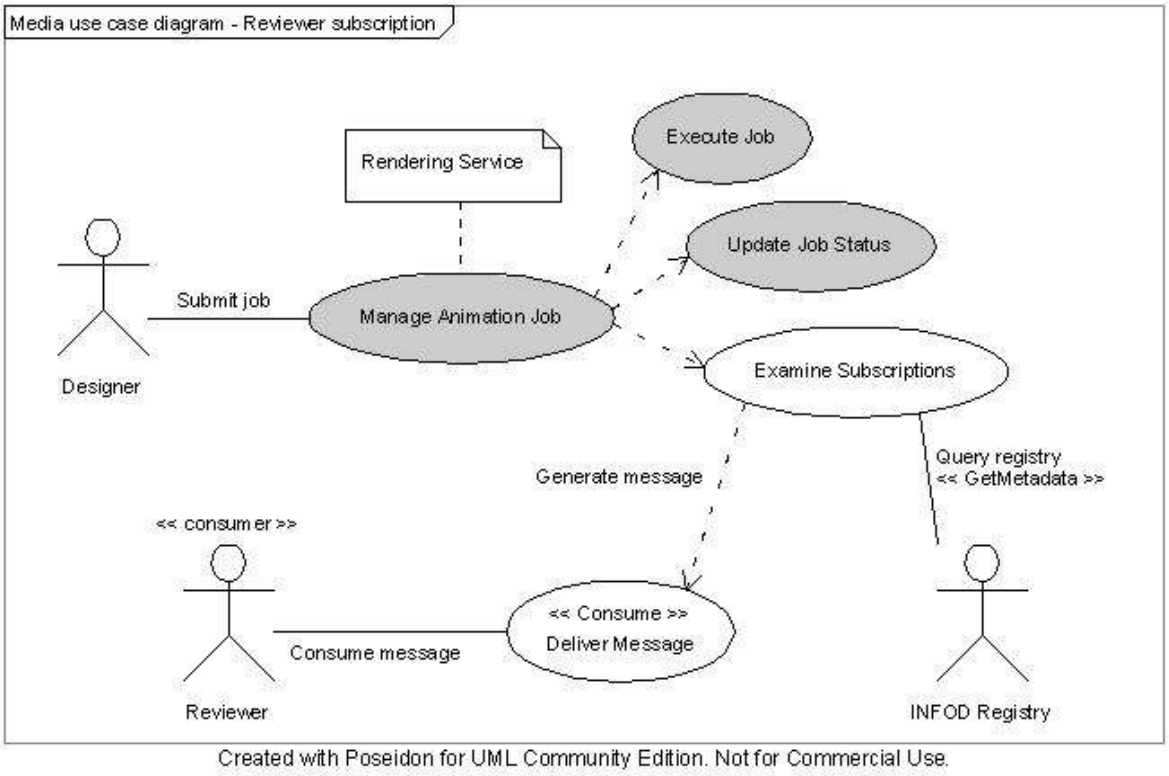


Figure 2-2: Use case diagram for a designer or reviewer subscription.

Note that the steps in *italics* (and also shown in grey in Figure 2-2) take place outside of the INFOD framework.

1. *Designer submits an animation job to the Rendering Service, which adds the job to its inventory.*
2. *The job completes (i.e. Job Status = "Completed") and the rendered animation is stored by the Rendering Service.*
3. The Rendering Service examines the associated subscriptions which it has been notified of by the INFOD Registry.
4. The Rendering Service generates messages for the relevant consumers.

2.3.5 Examples of XML messages

The following text gives an example of part of the XML messages that would be sent to and from the INFOD Registry when each of the relevant interfaces is called:

Step 1: Register the vocabularies with the INFOD Registry.

a) Registration of Animation (Data) Vocabulary

Request message:

```
<infod:RegisterDataVocabulary>
<infod:VocabularyName>NextGridAnimationDataVocab </infod:VocabularyName>
```

```
<infod:VocabularyLanguage>XML Schema(Namespace/URI of DataFormat)
</infod:VocabularyLanguage>
<infod:VocabularyBody>
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.w3.org/INFOD/Entity"
            targetNamespace="http://www.w3.org/INFOD/Entity">
   <xsd:element name = "DesignerName" type = "xs:string"/>
   <xsd:element name = "RenderingJobName" type = "xsd:string"/>
   <xsd:element name = "JobStart" type = "xsd:time"/>
   <xsd:element name = "JobStatus" type = "xsd:string"/>
</infod:VocabularyBody>
</infod:RegisterDataVocabulary>
```

Response message (for success case):

```
<infod:RegisterVocabularyResponse>
<infod:INFODVocabularyIdentifier>
<wsa:Address>http://www.nextgrid.org/NGInfoDRegistry/NextGridAnimationDataV
ocabEPR</wsa:Address>
</infod:INFODVocabularyIdentifier>
</infod:RegisterVocabularyResponse>
```

b) Registration of Rendering Service (Property) Vocabulary

Request message:

```
<infod:RegisterPropertyVocabulary>
<infod:VocabularyName>NextGridRenderingServicePropertyVocab
</infod:VocabularyName>
<infod:VocabularyBody>
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.w3.org/INFOD/Entity"
            targetNamespace="http://www.w3.org/INFOD/Entity">
   <xsd:element name = "ServiceName" type = "xsd:string"/>
   <xsd:element name = "ServiceAddress" type = "xsd:uri"/>
   <xsd:element name = "ServiceOrganization" type = "xsd:string"/>
   <xsd:element name = "RenderingSoftware" type = "xsd:complexType"/>
   <xsd:element name = "PricingModel" type = "xsd:string"/>
</infod:VocabularyBody>
</infod:RegisterPropertyVocabulary>
```

The response message would be very similar to that for step 1a and would include the EPR of the INFOD vocabulary identifier. In this example the EPR returned is:

http://www.nextgrid.org/NGInfoDRegistry/NextGridRenderingServicePropertyVocabEPR

Similarly request and response messages would be sent for registering the employee and organization property vocabularies.

The graphic in Fig. 3 depicts the relations of INFOD objects.

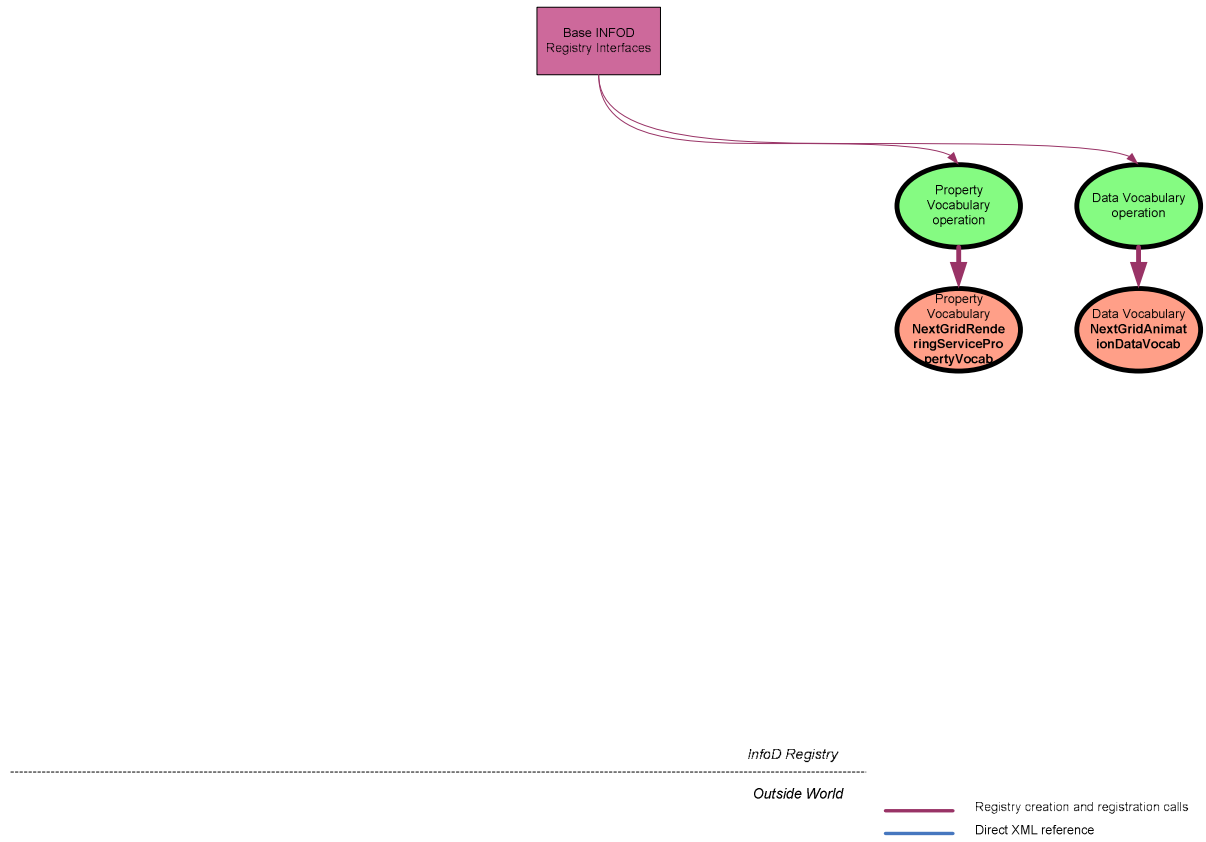


Figure 3 – Step 1 INFOD object relations

Step 2: Rendering Service added as a Publisher.

Request message:

```
<infod:CreatePublisher>
<infod:WSEntityIdentifier>
<wsa:Address> http://www.nextgrid.org/NextGridRenderingService
</wsa:Address>
   </infod:WSEntityIdentifier>
<infod:PublisherName>NextGridRenderingService</infod:PublisherName>
   <infod:Notification>
                TRUE
   </infod:Notification>
</infod:CreatePublisher>
```

Again, there would be a similar response message which would include the EPR of the INFOD entity identifier. In this example the EPR returned is:

http://www.nextgrid.org/NGInfoDRegistry/NextGridRenderingServiceEPR

The graphic in Fig. 4 depicts the relations of INFOD objects.

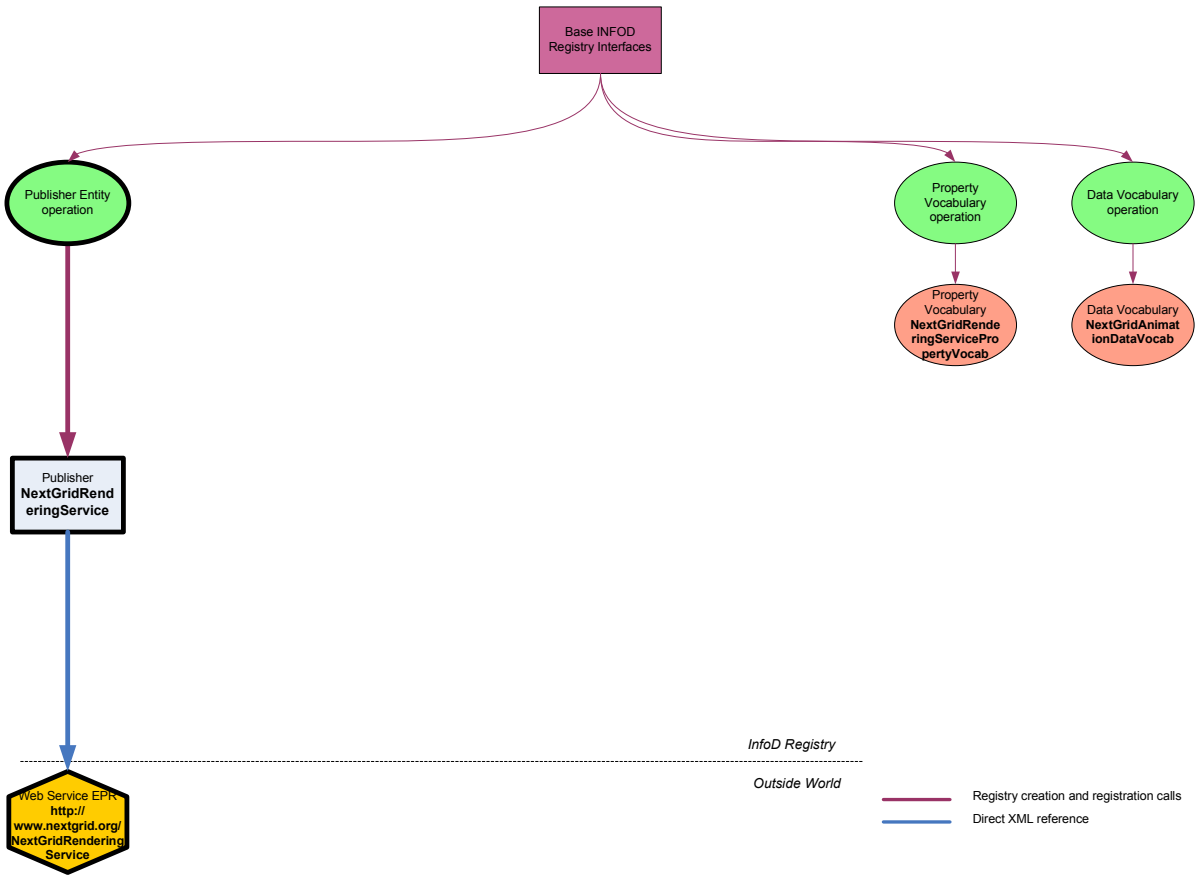


Figure 4 - Step 2 INFOD object relations

Step 3: Boss added to the INFOD Registry as a Subscriber.

Request message:

```
<infod:CreateSubscriber>
<infod:SubscriberName>John Boss</infod:SubscriberName>
</infod:CreateSubscriber>
```

In this example the returned EPR is:

http://www.nextgrid.org/NGInfoDRegistry/JohnBossEPR

The graphic in Fig. 5 depicts the relations of INFOD objects.

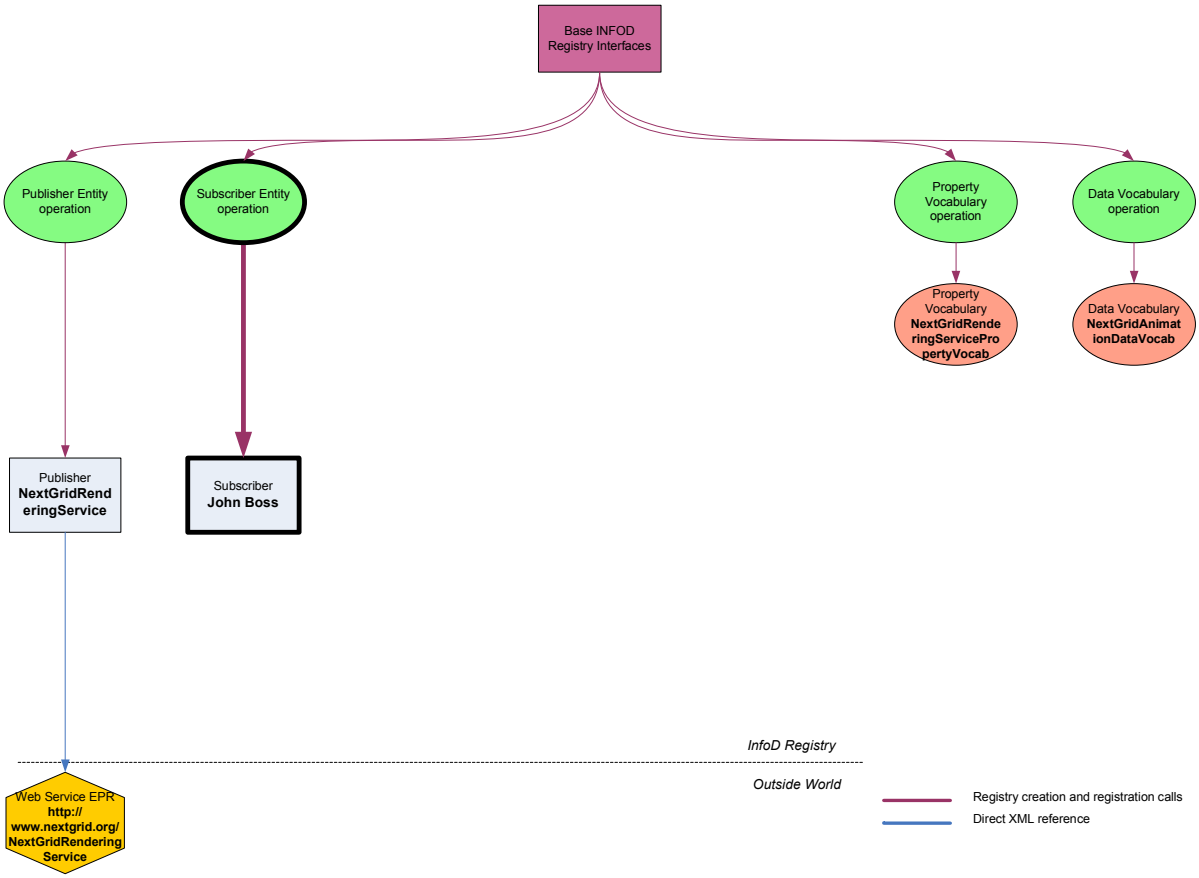


Figure 5 - Step 3 INFOD object relations

Step 4: Add Designer as a Consumer.

Request message:

```
<infod:CreateConsumer>
<infod:WSEntityIdentifier>
<wsa:Address>http://www.films.tv/PeterDesigner </wsa:Address>
   </infod:WSEntityIdentifier>
<infod:ConsumerName>Peter Designer</infod:ConsumerName>
</infod:CreateConsumer>
```

In this example the returned EPR is:

http://www.nextgrid.org/NGInfoDRegistry/PeterDesignerEPR

The graphic in Fig. 6 depicts the relations of INFOD objects.

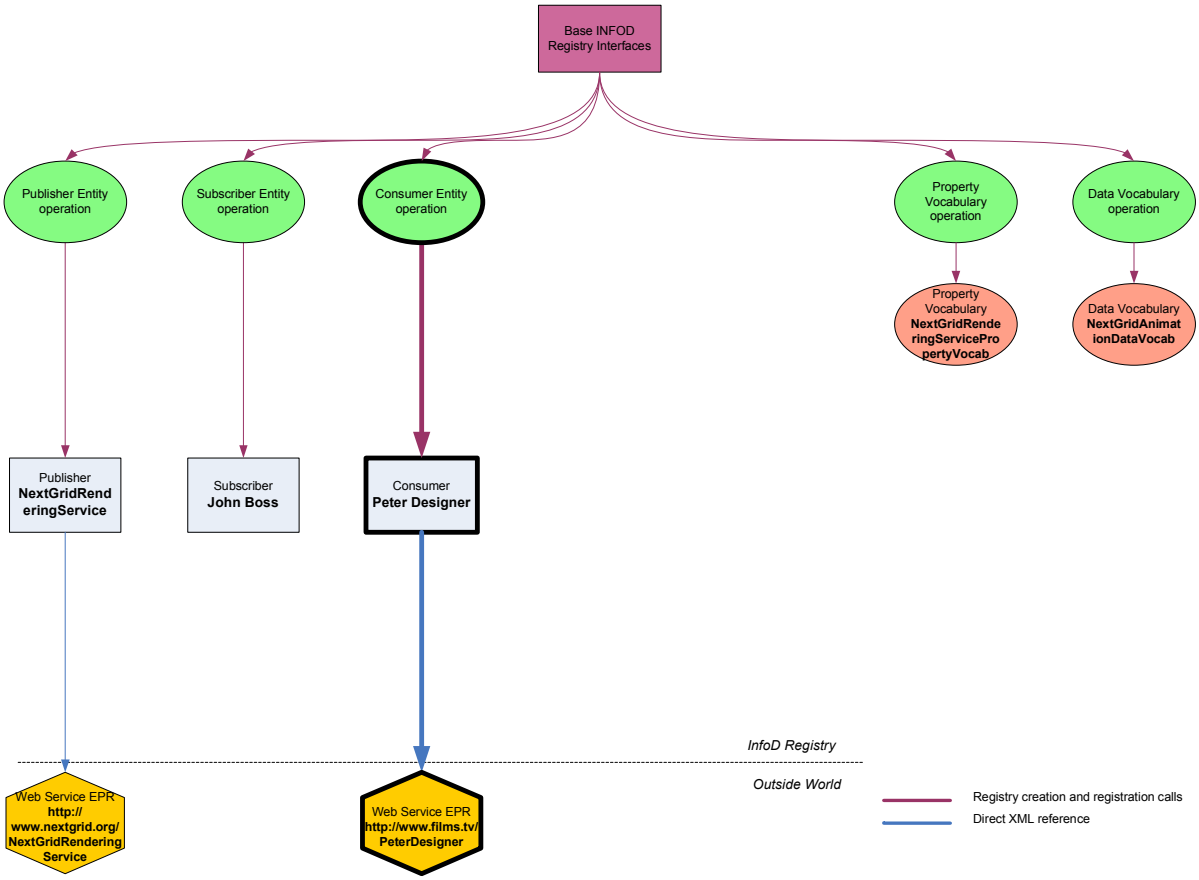


Figure 6 - Step 4 INFOD object relations

Step 5: Create associations between the publisher and the data vocabulary.

Request message:

```
<infod:AssociateVocabulary>
<infod:AssociateVocabularyName>PublisherAndDataVocabAssociation
</infod:AssociateVocabularyName>
<infod:AssociationEntityReference>
<wsa:Address>http://www.nextgrid.org/NGInfoDRegistry/
NextGridRenderingServiceEPR</wsa:Address>
</infod:AssociationEntityReference>
<infod:VocabularyReference>
<wsa:Address>http://www.nextgrid.org/NGInfoDRegistry/NextGridAnimationDataV
ocabEPR</wsa:Address>
</infod:VocabularyReference>
</infod:AssociateVocabulary>
```

In this example the returned EPR is:

```
http://www.nextgrid.org/NGInfoDRegistry/PublisherAndDataVocabAssociationEP
R
```

The graphic in Fig. 7 depicts the relations of INFOD objects.

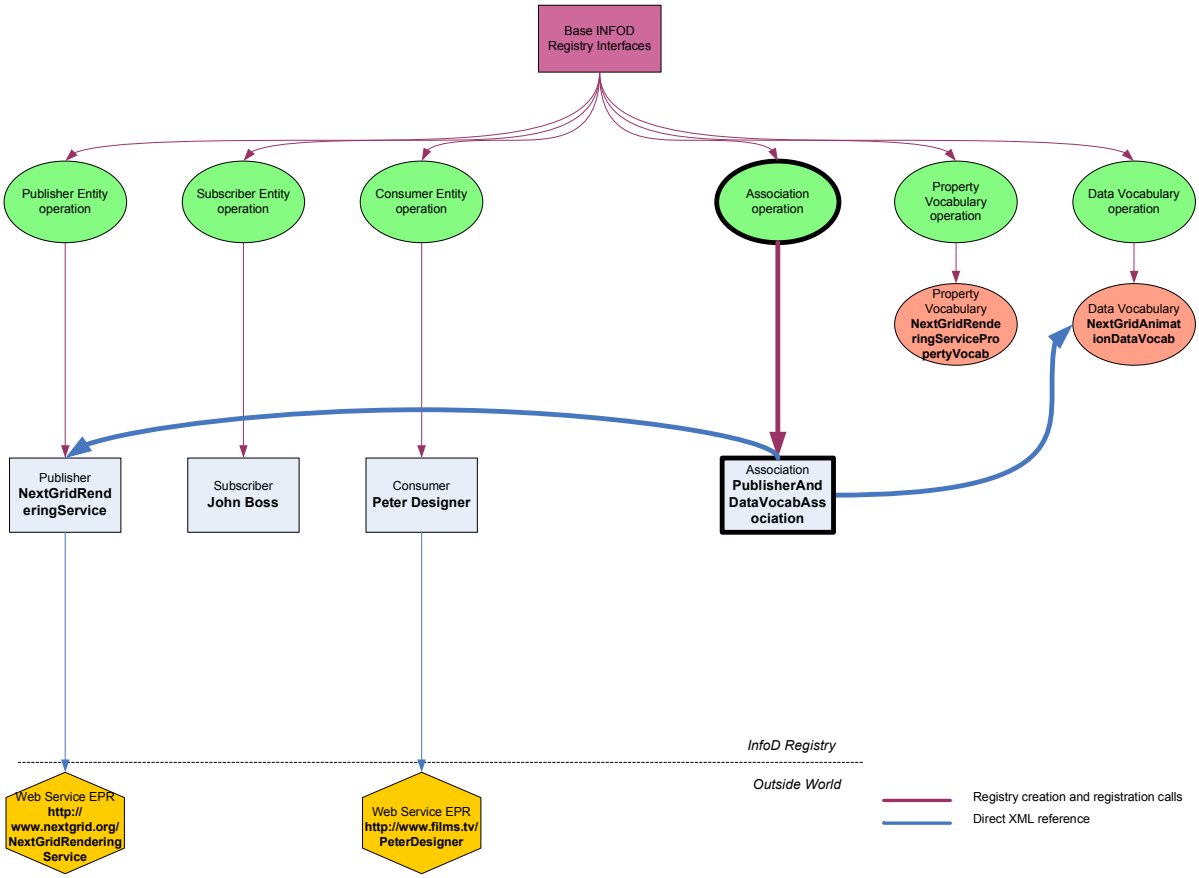


Figure 7 - Step 5 INFOD object relations

Step 6: Create instances of the relevant property vocabularies for the publisher, subscriber and consumers.

Request message:

```
<infod:CreatePropertyVocabularyInstance>
  <infod:VocabularyInstanceEntityReference>
    <wsa:Address>
      http://www.nextgrid.org/NGInfoDRegistry/NextGridRenderingServiceEPR
    </wsa:Address>
  </infod:VocabularyInstanceEntityReference>
  <infod:VocabularyInstanceVocabularyReference>
    <wsa:Address>
      http://www.nextgrid.org/NGInfoDRegistry/
        NextGridRenderingServicePropertyVocabEPR
    </wsa:Address>
  </infod:VocabularyInstanceVocabularyReference>
  <infod:VocabularyInstanceVocabularyBody>
    <?xml version="1.0"?>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:ident="http://www.w3.org/INFOD/Entity"
          targetNamespace="http://www.w3.org/INFOD/Entity">
    <ServiceName>NextGrid Rendering Service</ServiceName>
    <ServiceOrganisation>SuperInc</ServiceOrganization>
    <ServiceAddress>
      http://www.nextgrid.org/NextGridRenderingService
    </ServiceAddress>
```

```
    <RenderingSoftware>3D Max</RenderingSoftware>
    <PricingModel>myPricingModel</PricingModel>
  </infod:VocabularyInstanceVocabularyBody>
</infod:CreatePropertyVocabularyInstance>
```

In this example the returned EPR is:

`http://www.nextgrid.org/NGInfoDRegistry/PublisherVocabInstanceEPR`

Similar messages would be sent for creating the instances of the subscriber and consumer property vocabularies.

The graphic in Fig. 8 depicts the relations of INFOD objects.

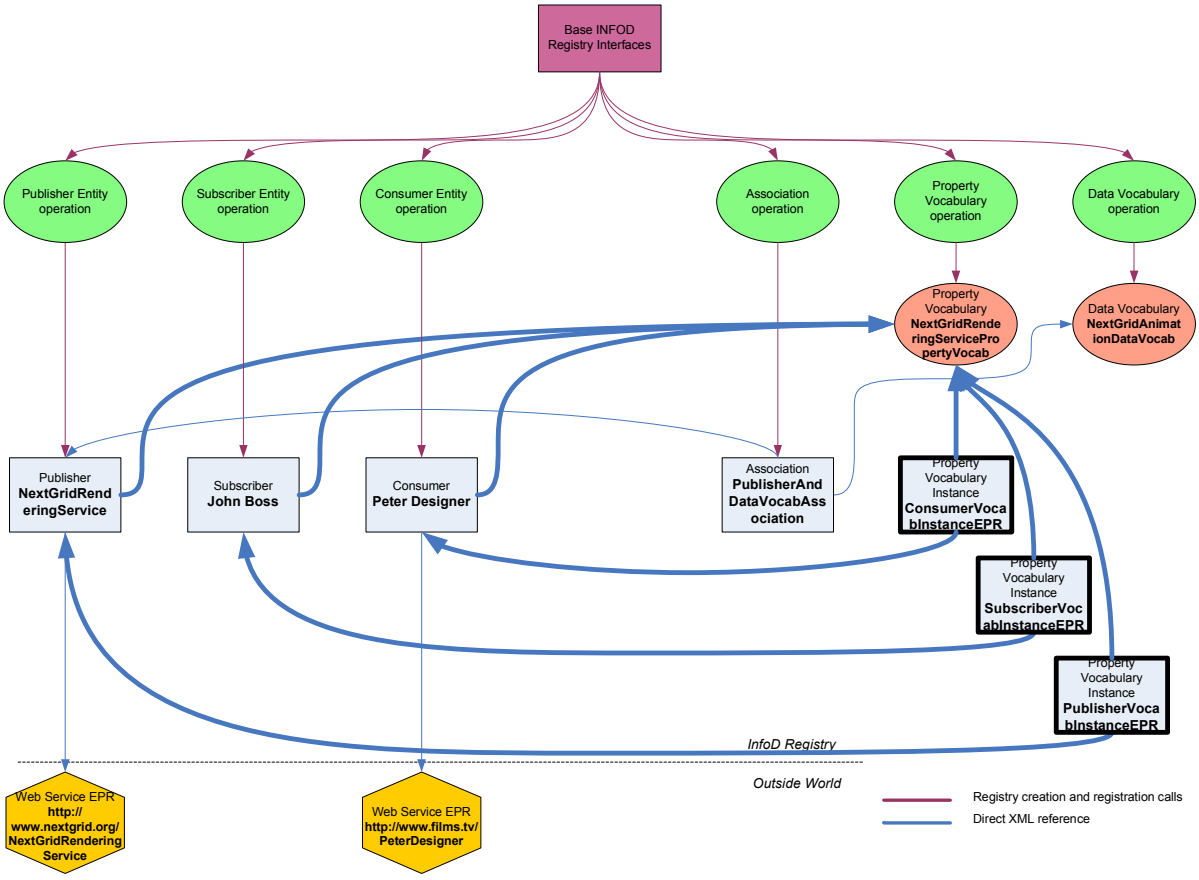


Figure 8 - Step 6 INFOD object relations

Step 7: Boss adds the relevant subscriptions to the INFOD Registry.

Request message:

```
<infod:CreateSubscription>
  <infod:SubscriptionName>
    DesignerMonitoringJobsSubscription
  </infod:SubscriptionName>
  <infod:SubscriptionDescription>
    Designer monitoring animation jobs subscription
  </infod:SubscriptionDescription>
  <infod:SubscriberReference>
    <wsa:Address>
```

```
      http://www.nextgrid.org/NGInfoDRegistry/JohnBossEPR
    </wsa:Address>
  </infod:SubscriberReference>
  <infod:DataConstraints>
    For $data in doc("http://www.nextgrid.org/NGInfoDRegistry/
        datavocabularies/AnimationDataVocabEPR")
    Where
      $data/DesignerName = $StaticConsumers/EmployeeName
      $data/Status = "Completed"
      $data/Project= "Toy Story 3"
    Return
      $data/Project
      $data/JobName
      $data/DesignerName
      $data/Status
  </infod:DataConstraints>
  <infod:DynamicConsumerConstraints>
    For $employees in doc("http://www.nextgrid.org/NGInfoDRegistry/
        propertyvocabularyinstances/EmployeeVocabEPR")
      For $consumers in
        doc("http://www.nextgrid.org/NGInfoDRegistry/consumers
            /infodConsumer")
        Where
          $employees/VocabularyInstanceEntityReference = GetEPR($consumers)
        Return
           $consumers/WSEntityIdentifier
  </infod:DynamicConsumerConstraints>
</infod:CreateSubscription>
```

In this example the returned EPR is:

`http://www.nextgrid.org/NGInfoDRegistry/MonitoringJobsSubscriptionEPR`

Similar request messages would be sent for the other subscriptions.

The graphic in Fig. 9 depicts the relations of INFOD objects.

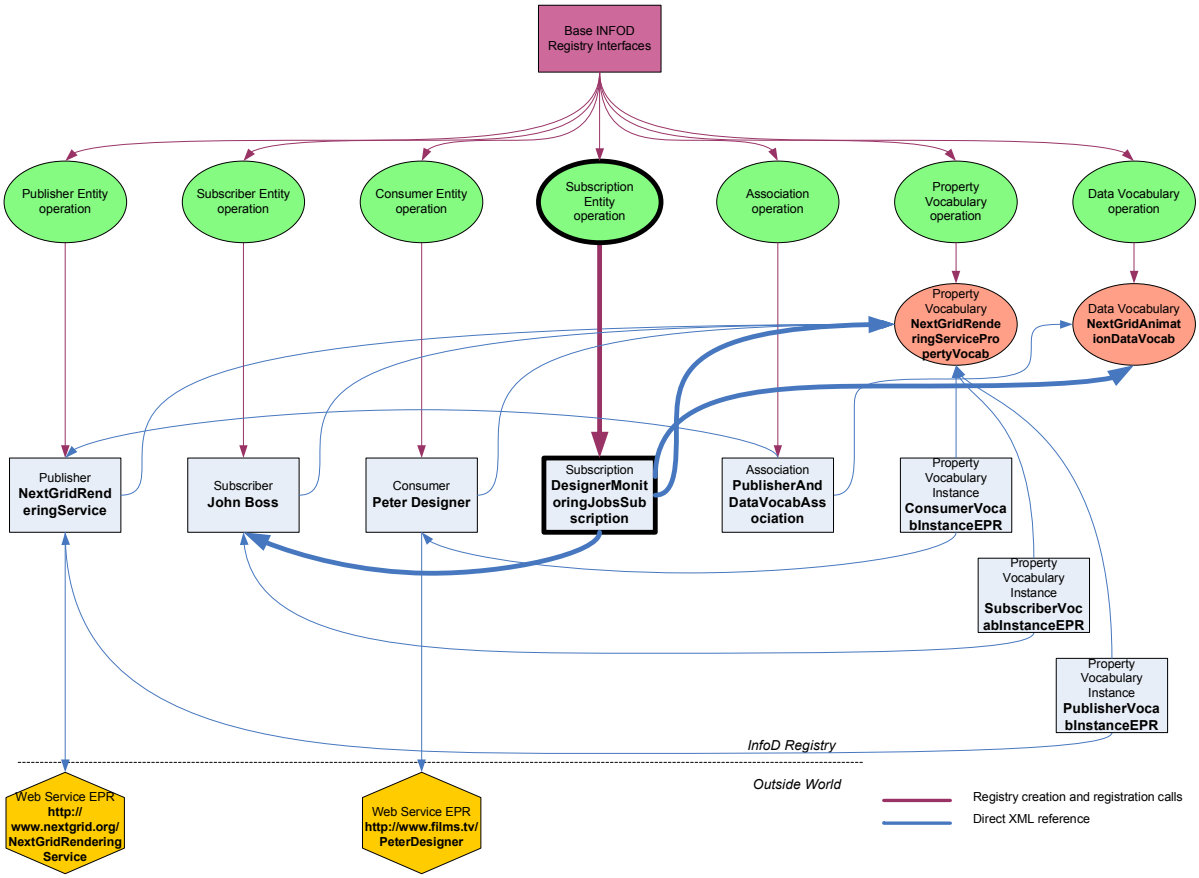


Figure 9 - Step 7 INFOD object relations

Step 8: INFOD registry sends Subscription Notification to Rendering Service

The text below gives an example XML for the notification message body.

```
<Body>
<infod:SubscriptionNotification>
<infod:SubscriptionReference>
<wsa:Address>
http://www.nextgrid.org/NGInfoDRegistry/MonitoringJobsSubscriptionEPR
<wsa:Address>
</infod:SubscriptionReference>
<infod:StaticConsumers>
<wsa:Address>
http://www.nextgrid.org/NGInfoDRegistry/PeterDesignerEPR
<wsa:Address>
</infod:StaticConsumers>
<infod:DataConstraints>
         For $data in doc("http://www.nextgrid.org/NGInfoDRegistry/
datavocabularies/AnimationDataVocabEPR")
Where
$data/DesignerName = $StaticConsumers/EmployeeName
                $data/Status = "Completed"
$data/Project= "Toy Story 3"
         Return
                $data/Project
                $data/JobName
                $data/DesignerName
```

```
                $data/Status
            </infod:DataConstraints>
   </infod:SubscriptionNotification>
</Body>
```

Step 9: The Rendering Service generates messages for the relevant consumers.

Output message to consumers:

The text below gives some example XML for a message body. The structure of the message is as defined in WS-BaseNotification.

```
<wsnt:Notify>
  <wsnt:NotificationMessage>
    <wsnt:SubscriptionReference>
<wsa:Address>
http://www.nextgrid.org/NGInfoDRegistry/MonitoringJobsSubscriptionEPR
<wsa:Address>
    </wsnt:SubscriptionReference>
    <wsnt:ProducerReference>
<wsa:Address> http://www.nextgrid.org/NextGridRenderingService
</wsa:Address>
    </wsnt:ProducerReference>
    <wsnt:Message>
<Type>XML</Type>
<Length>196</Length>
<Data>
<ProjectName>Toy Story 3</ProjectName>
<JobName>AnimationJobID12345</JobName>
<DesignerName>Peter Designer</DesignerName>
<Status>Completed</Status>
</Data>
<Flag>1</Flag>
   <Transform>None</Transform>
    </wsnt:Message>
  </wsnt:NotificationMessage>
</wsnt:Notify>
```

The following diagram shows the entities and vocabularies that have been defined and the relations between them - Figure 2-10.

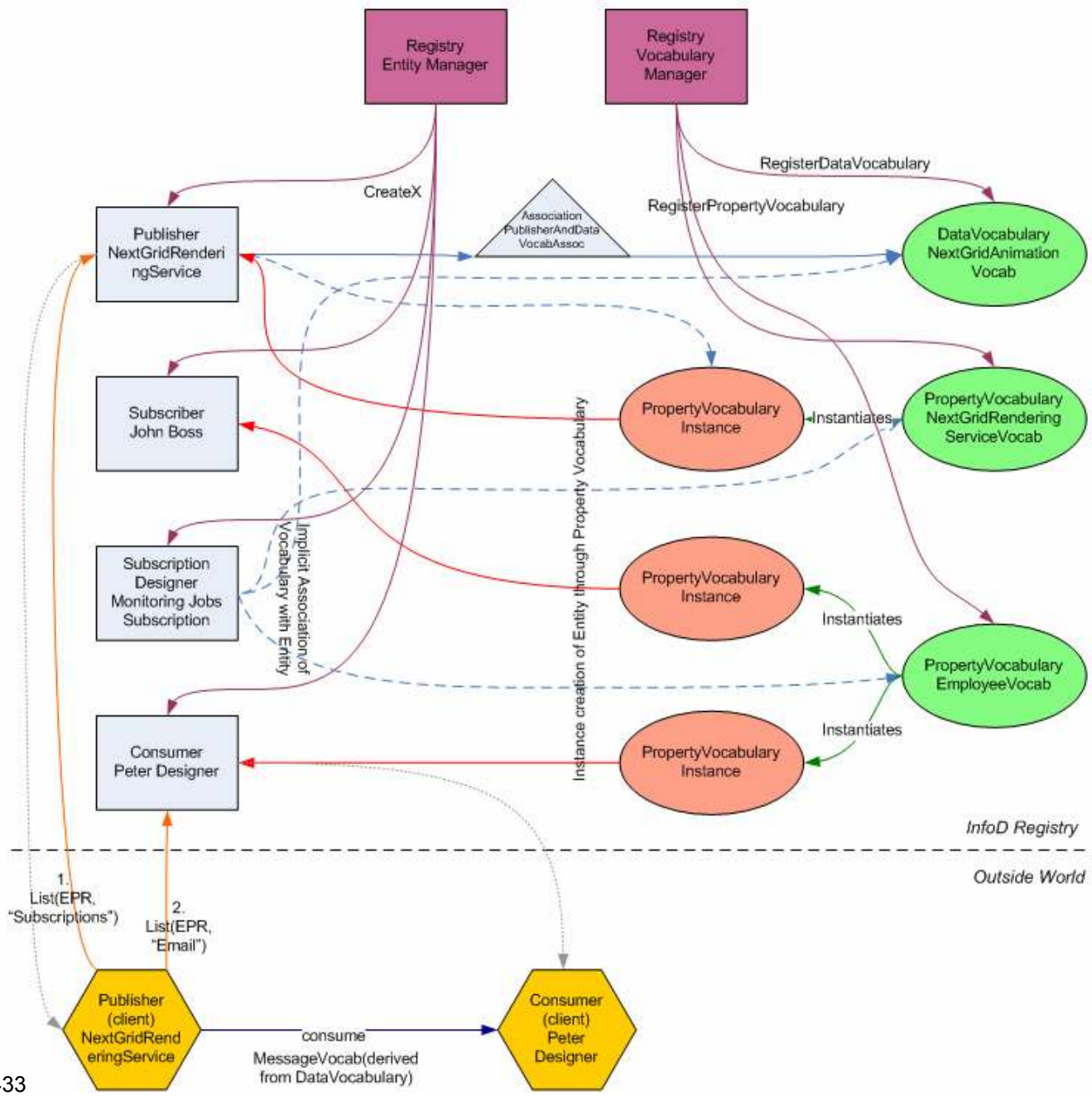


Figure 2-10: Relationships between entities and vocabularies.

2.4 Security

Security would be very important for a film that is currently in production say. As well as the need to authenticate the identities of designers, reviewers & rendering services, one may also want job submission data and returned animations (MPEGs) to be encrypted.

But it is assumed that no additional security requirements would be needed beyond the security common to the entire set of scenarios.

2.5 Performance

Response times of around a few seconds would be required but this is unlikely to be beyond the typical performance requirements common to many of the use cases.

2.6 Requirements Implied

R1	Constraints, which restrict the flow of messages from publishers to consumers, must be composable.	The constraints contained in subscriptions must be composed with the constraints specified by the rendering services and designers.
R2	Publishers should be able to describe their available messages, events and states in terms of a vocabulary.	Rendering services need to define the information about rendering jobs that consumers can receive.
R3	Subscribers must be able to constrain messages based on message content and publisher information.	Designers, acting as their own subscribers, must be able to select publishers based on their properties.
R4	Publishers must be able to choose what messages to publish based on consumer and subscriber information.	Rendering services must be able to restrict consumers according to their policies (constraints on designers etc).
R5	Consumers must be able to constrain messages based on message content, publisher information and subscriber information.	Designers and reviewers need to define the messages they receive from which rendering services.
R6	Any component can request that it be notified by the registry, via WSN, of changes that the component considers interesting.	Rendering services need to know which subscriptions are relevant to them and of any changes or additions that occur.

3 Car Dealer Use Case

3.1 Introduction

Car buyers like to be aware of all cars of interest from those dealers who are located close by and who have good BBB and service ratings. Instead of receiving pre-canned information buyers like to specify which information is relevant to them.

Car dealer too like to put restrictions on potential buyers; they like to communicate only with those buyers who have good credit rating and do not live too far away.

Car dealers as well as buyers dealers like to specify their interest and constraints in a terminology that is meaningful to the car buyer and seller community and does not require any IT terminology.

3.2 Actors

The actors are car dealers (acting as publishers) and car buyers (acting as subscribers and consumers).

It is assumed that car dealers and car buyers form a (virtual) community. This community is described through community vocabularies, the car dealers (buyers) by the car dealer (buyer) vocabulary respectively.

3.3 Scenarios

The scenario requires the following activities:

- The creation of the Car Dealer and a Car Buyer community
- Joining and leaving the car dealer community
- Joining and leaving the car buyer community
- Subscribing to (the inventory of) car dealers
- Publishing Information
- Consuming Information

XML schema and data are presented in tables; Operations are provided in each sub-section.

3.3.1 Creating the Car Dealer/Buyer Communities

The (ideal) car dealer community consists of all car dealers independent of the place of business; this community is described by the Car_Dealer vocabulary. Here is an example of a Car_Dealer property dealer vocabulary.

Car Dealer Vocabulary Components	Comment
Name	Name of dealer
Location	Address of dealer
Phone	Phone number of dealer
E-mail	E-mail address of dealer
Web-site	URL
Open_since	Year founded
BBB_rating	A number between 1 and 5 – this information must contain a source reference
Service_Rating	A number between 1 and 12 – this information must contain a source reference

474 **Table 3-1: Car_Dealer Vocabulary**

475 The (ideal) car buyer community consists of all people or organizations interested in buying cars; this
476 community could be defined by the Car_Buyer property vocabulary. Here is an example of a
477 Car_Buyer vocabulary:

Car Buyer Vocabulary Components	Comment
Name	Name of car buyer
Location	Address of car buyer
Phone	Phone number of car buyer
E-mail	An e-mail address of car buyer
Credit_Rating	A number between 360 and 720 – this information must contain a source reference
Interest	Optional expression describing interest of the buyer; could be used to create subscriptions

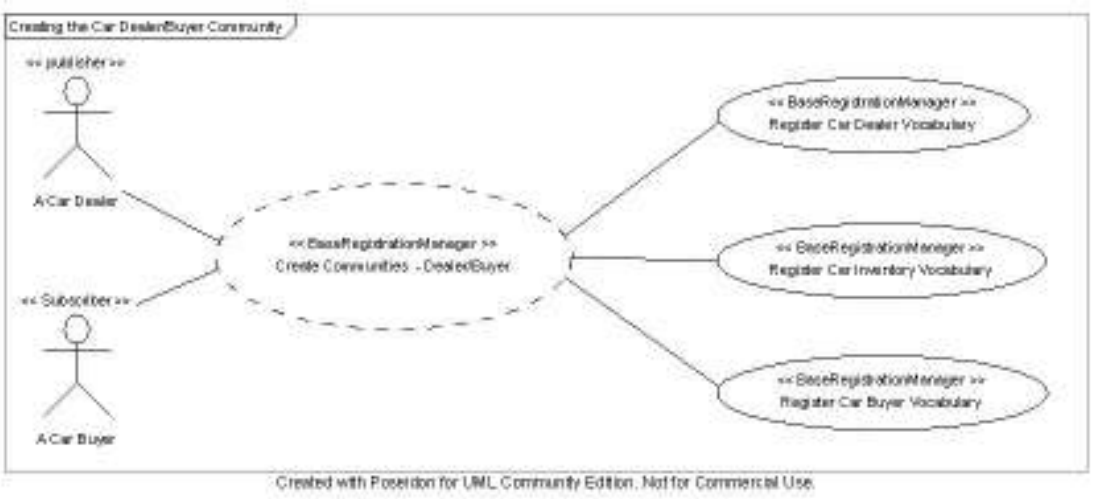
478 **Table 3-2: Car_Buyer Vocabulary**

479 Car dealers and buyers also need a data vocabulary to describe and select the item of their shared
480 interest - the cars in the inventory of the car dealers. Here is an example of a Car_Inventory.

Car Inventory Vocabulary Components	Comment
Inventory number	Inventory number
Make	Brand name
Model	Model
Year	Model year
Type	Type of car
External Color	External color
Internal Color	Internal color
Dealer	Dealer – this could be a reference
Price	Amount $

481 **Table 3-3: Car_Inventory Vocabulary**

482 The Car_Inventory data vocabulary is an XML schema. As a consequence XQuery can be used the
483 select cars of interest. The specific query is part of the subscription. Since the consumer is interested
484 in relevant changes the specified query will be re-evaluated whenever there is a change of the car
485 inventory[1]. If there is a new result set either the new results set or the changes will be disseminated to
486 the consumer (depending on the subscription directives).



487

488 **Figure 3-3-1: Creating Car Dealer / Buyer Communities**

[1] It is assumed that a performing implementation does not require a full evaluation of each query after each change.

3.3.2 Joining/Leaving the Car Dealer Community

In order to join the car dealer community, a car dealership has to represent itself as a publisher and as a car dealer.

Here is an example of a car dealer represented as publisher (representing the IT perspective) and as a dealer (representing the community perspective):

Publisher Entity **INFOD Components**	**Values**
infod_Type	Publisher
wsinfod _Name	Frontier Ford
wsinfod _Description	Oldest Ford Dealer in SFO Bay Area Featuring also fine Italian Cars
wsinfod _Property_Constraint	Consumer: Buyer (Distance to customer < 30 miles, Credit rating > 700)

Table 3-4: Car Dealer as Publisher

Assignment of a publisher to a Dealer Vocabulary	**Values**
EntityReference	The EPR of the publisher entity:
Name	Frontier Ford
Location	101 Auto Row, Redwood city, CA 94065
Phone	+1-650-000-0000
E-mail	Info@Frontier_Ford .com
Web-site	www.Frontier_Ford.com
Open_since	1953
BBB_rating	5
Service_Rating	10

Table 3-5: Car Dealer as Member of Car Dealer Community

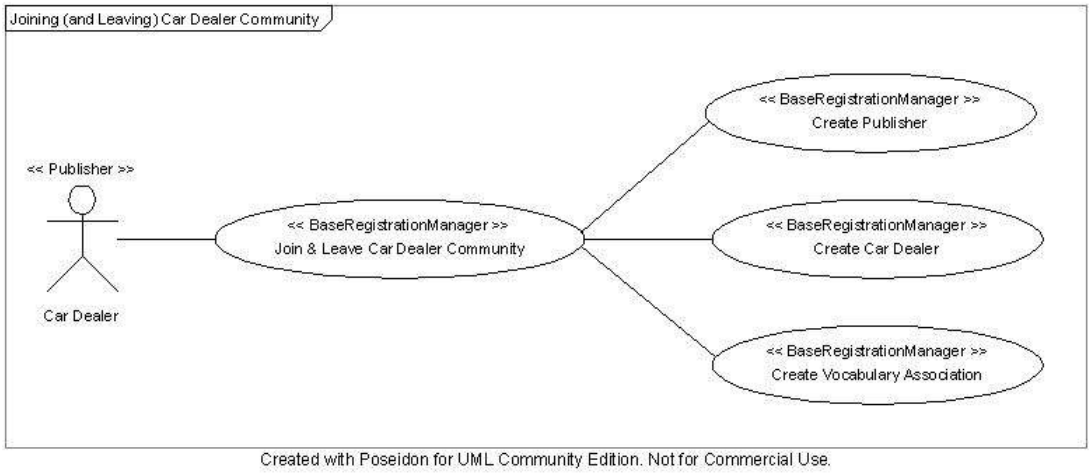
Publishers (dealers) can specify if they should be notified about new, Replaced, and deleted subscriptions. In order to get these notifications the publisher (dealer) has to support the consume interface and must be able to process notification regarding subscriptions.

The publishers (dealers) have to indicate that they have instances based on the car vocabulary. Creating a vocabulary association does this. The car data are characterized through the car data vocabulary and its vocabulary association to the specify car dealer. There are no user properties associated to the vocabulary association.

Association of publisher to Car_Inventory Vocabulary	Values
Infod_Type	Vocabulary Association
Infod_Name	Inventory
Infod_Description	Cars on sale by Frontier Ford
Infod_Entity	Publisher1 (EPR to publisher entity)
Infod_Associate	CarVocab (EPR to vocabulary entity)

503 **Table 3-6: Inventory Data of Frontier Ford**

504 It is assumed that all the cars of the large set of dealers are contained in one collection. So the dealer
505 has to describe which part of the selection represents his cars.



506

507 **Figure 3-2: Joining (and Leaving) Car Dealer Community**

508 3.3.3 Joining/Leaving the Car Buyer Community

509 In order to join the car buyer community, a car buyer has present him/herself as a consumer and as a
510 car buyer.

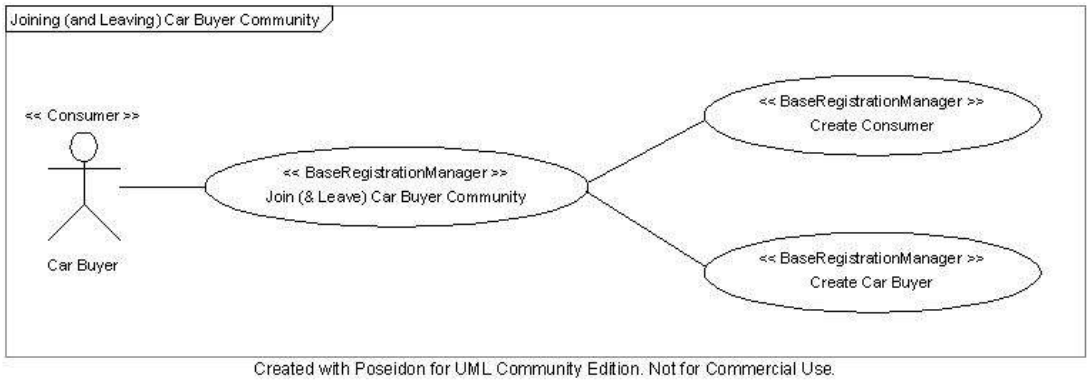
511 Here is an example of a car buyer represented as consumer (representing the IT perspective) and as
512 a buyer (representing the community perspective):

Consumer Entity **INFOD Components**	**Values**
infod_Type	Consumer
infod_Name	Susan Maria Callas
infod_Description	Buyer of fancy cars
infod_Property_Constraint	Dealer: Years in business > 10 years, BBB rating > 3, Service rating > 10

513 **Table 3-7: Car Buyer as Consumer**

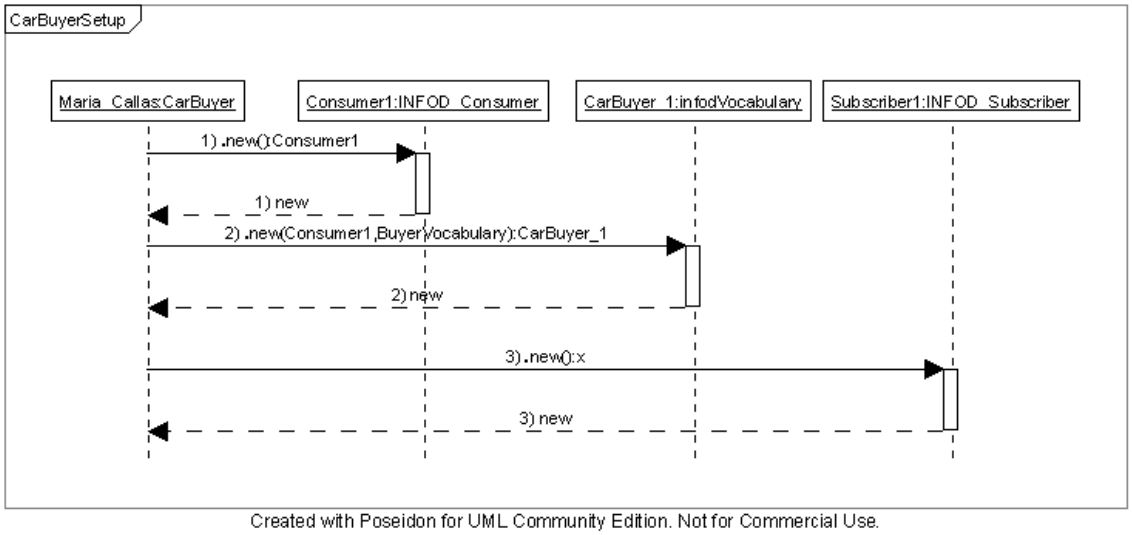
Assignment of a consumer to a Buyer Vocabulary	
EntityReference	EPR of consumer entity (Consumer1 - see below)
VocabReference	*EPR of vocabulary (BuyerVocab – see above)*
Name	Susan Maria Callas
Location	15998 Portola Off Road, Portola Valley, CA
Phone	+1-650-000-000
E-mail	callas&opera.music
Credit_Rating	700
Interest	Cars: Make = ‘Italian’ and Model = ‘Sport’ cars’ and Year < 1995 and Exterior Color ‘Red’

514 **Table 3-8: Car Buyer as Member of Car Buyer Community**



515

516 **Figure 3-3-2: Joining (and Leaving) Car Buyer Community**



517

518 **Figure 3-4: Joining (and Leaving) Car Buyer Community (interaction diagram)**

519 3.3.4 Subscribing to the Inventory of Car Dealer

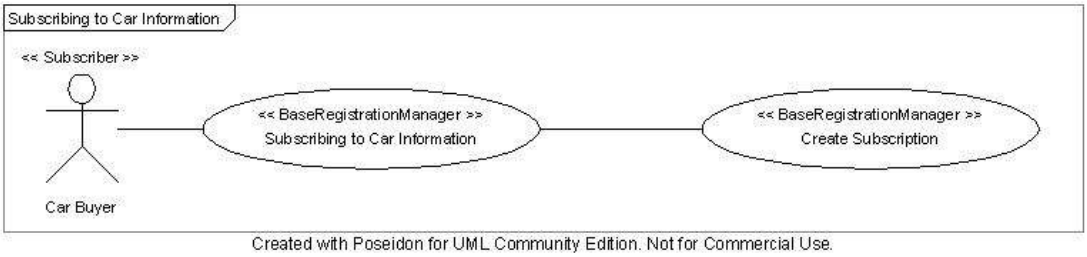
520 The consumer (car buyer), acting as subscriber, has to specify which cars are of interest. Specifying
521 a subscription does this: Before this can be done the car buyer has to register as a consumer

Subscription Entity **INFOD Component**	**Values**
infod_Type	Subscription
infod_Name	Find a Car
infod_Description	Watch for car offers
infod_Data_Constraint	Make Italian, Price < 250000
infod_Property_Constraints	Dealer: Distance < 25 miles

522 **Table 3-9: Subscription**

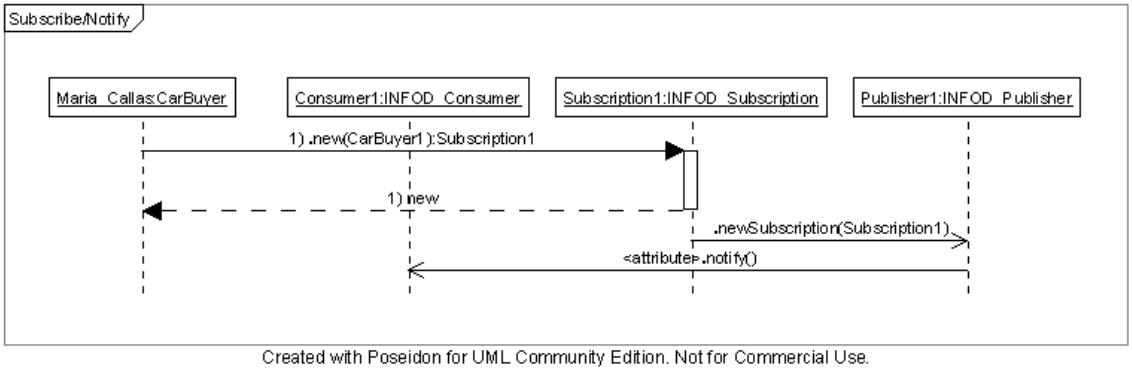
523 The subscription manager processes the subscription and notifies the relevant publishers (car
524 dealers). Car dealers who are matching the (property) constraint defined by car buyers will be
525 informed if the consumer (buyer) matches their (property) constraint.

526 The notification to the publisher contains the data constraints, what information is of interest and the
527 consumer reference.



528

529 **Figure 3-5: Creating a Subscription**



530

531 **Figure 3-6: Creating a Subscription (interaction diagram)**

532 3.3.5 Publishing Information

533 The publisher publishes information according to the subscription; e.g.; the publications are tailored to
534 the request of the consumer.

535 After receiving the subscription the publisher sends immediately a message for each car that matches
536 the criteria. Once this is done the publisher sends a message for each car that is either added or
537 removed from the inventory and matches the criteria.

538 3.3.6 Consuming Information

539 The consumer receives the messages from the publisher using the consume interface.

540 3.3.7 Examples of XML messages

541 The following text gives an example of part of the XML messages that would be sent to and from the
542 INFOD Registry when each of the relevant interfaces is called:

543 **Step 1: Register the vocabularies with the INFOD Registry.**

544 **a) Registration of Car Dealer (Property) Vocabulary**

545 Request message:

546
```
<infod:RegisterPropertyVocabulary>
```

```
<infod:VocabularyName>CarCommunityDealerVocab </infod:VocabularyName>
<infod:VocabularyLanguage>XML Schema(Namespace/URI of DataFormat)
</infod:VocabularyLanguage>
<infod:VocabularyBody>
NOTE: this would need to be encoded correctly (escaped etc.)
e.g. "<" becomes &lt
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.w3.org/INFOD/Entity"
            targetNamespace="http://www.w3.org/INFOD/Entity">
   <xsd:element name = "DealerName" type = "xsd:string"/>
   <xsd:element name = "DealerLocation" type = "xsd:string"/>
   <xsd:element name = "DealerPhoneNo" type = "xsd:string"/>
<xsd:element name = "DealerEmail" type = "xsd:string"/>
<xsd:element name = "DealerURL" type = "xsd:string"/>
<xsd:element name = "DealerOpened" type = "xsd:int"/>
<xsd:element name = "DealerBBBRating" type = "xsd:int"/>
   <xsd:element name = "DealerServiceRating" type = "xsd:int"/>
</infod:VocabularyBody>
</infod:RegisterPropertyVocabulary>
```

Response message (for success case):

```
<infod:RegisterPropertyVocabularyResponse>
<infod:INFODVocabularyIdentifier>
<wsa:Address>http://www.carcommunity.com/CCInfoDRegistry/CarCommunityDealer
VocabEPR</wsa:Address>
</infod:INFODVocabularyIdentifier>
</infod:RegisterPropertyVocabularyResponse>
```

b) Registration of Car Buyer (Property) Vocabulary

Request message:

```
<infod:RegisterPropertyVocabulary>
<infod:VocabularyName>CarCommunityBuyerVocab </infod:VocabularyName>
<infod:VocabularyLanguage>XML Schema(Namespace/URI of DataFormat)
</infod:VocabularyLanguage>
<infod:VocabularyBody>
NOTE: this would need to be encoded correctly (escaped etc.)
e.g. "<" becomes &lt
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.w3.org/INFOD/Entity"
            targetNamespace="http://www.w3.org/INFOD/Entity">
   <xsd:element name = "BuyerName" type = "xsd:string"/>
   <xsd:element name = "BuyerLocation" type = "xsd:string"/>
   <xsd:element name = "BuyerPhoneNo" type = "xsd:string"/>
<xsd:element name = "BuyerEmail" type = "xsd:string"/>
<xsd:element name = "BuyerCRRating" type = "xsd:int"/>
   <xsd:element name = "BuyerInterest" type = "xsd:string"/>
</infod:VocabularyBody>
</infod:RegisterPropertyVocabulary>
```

Response message (for success case):

```
<infod:RegisterVocabularyResponse>
<infod:INFODVocabularyIdentifier>
<wsa:Address>:http://www.carcommunity.com/CCInfoDRegistry/CarCommunityBuyer
VocabEPR</wsa:Address>
</infod:INFODVocabularyIdentifier>
</infod:RegisterVocabularyResponse>
```

602

603 **c) Registration of Car (Data) Vocabulary**

604 Request message:

```
605 <infod:RegisterDataVocabulary>
606 <infod:VocabularyName>CarCommunityCarVocab </infod:VocabularyName>
607 <infod:VocabularyLanguage>XML Schema(Namespace/URI of DataFormat)
608 </infod:VocabularyLanguage>
609 <infod:VocabularyBody>
610 NOTE: this would need to be encoded correctly (escaped etc.)
611 e.g. "<" becomes &lt
612 <?xml version="1.0"?>
613 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
614             xmlns:ident="http://www.w3.org/INFOD/Entity"
615             targetNamespace="http://www.w3.org/INFOD/Entity">
616    <xsd:element name = "CarInvNumber" type = "xsd:number"/>
617    <xsd:element name = "CarMake" type = "xsd:string"/>
618    <xsd:element name = "CarModel" type = "xsd:string"/>
619 <xsd:element name = "CarYear" type = "xsd:int"/>
620 <xsd:element name = "CarType" type = "xsd:string"/>
621 <xsd:element name = "CarExtColor" type = "xsd:string"/>
622 <xsd:element name = "CarIntColor" type = "xsd:string"/>
623    <xsd:element name = "CarDealer" type = "xsd:string"/>
624    <xsd:element name = "CarPrice" type = "xsd:int"/>
625 </infod:VocabularyBody>
626 </infod:RegisterDataVocabulary>
```

627 Response message (for success case):

```
628 <infod:RegisterPropertyVocabularyResponse>
629 <infod:INFODVocabularyReference>
630 <wsa:Address>:http:www.carcommunity.com/CCInfoDRegistry/CarCommunityCarVoca
631 bEPR</wsa:Address>
632 </infod:INFODVocabularyReference>
633 </infod:RegisterPropertyVocabularyResponse>
```

634 **Step 2: Car Dealer added as a Publisher and Community Member**

635 **a) Registration of Car Dealer as Publisher**

636 Request message:

```
637 <infod:CreatePublisher>
638    </infod:WSEntityReference>
639 wsa:http://www.carcommunity.com/CarDealerServices
640 </infod:WSEntityReference>
641 <infod:PublisherName>Frontier Ford</infod:PublisherName>
642 <infod:PublisherDescription>Oldest Ford Dealer in SFO Bay Area Featuring
643 also fine Italian Cars</infod:PublisherDescription>
644 <infod:PropertyConstraints>Car_Buyer (Distance < 30 miles) and CR > 4)
645 </infod:PropertyConstraints>
646 </infod:CreatePublisher>
```

647 Returned EPR:

648 http://www.carcommunity.com/CCInfoDRegistry/Publisher/Frontier_Ford

b) Registration of Car Dealer as Community Member

Request message:

```
<infod:CreatePropertyVocabularyInstance>
<infod:VocabularyInstanceEntityReference>
<wsa:Address>:http://www.carcommunity.com/CCInfoDRegistry/Publisher/Frontie
r_FordEPR</wsa:Address>
</infod:VocabularyInstanceEntityReference>
<infod:VocabularyInstanceVocabularyReference>
<wsa:Address>:http://www.carcommunity.com/CCInfoDRegistry/CarCommunityDeale
rVocabEPR</wsa:Address>
   </infod:VocabularyInstanceVocabularyReference>
<infod:VocabularyInstanceVocabularyBody>
NOTE: this would need to be encoded correctly (escaped etc.)
e.g. "<" becomes &lt
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.w3.org/INFOD/Entity"
            targetNamespace="http://www.w3.org/INFOD/Entity">
   <DealerName>Frontier Ford</DealerName>
<DealerLocation>101 Auto Row, Redwood City, CA 94065</DealerLocation>
   <DealerPhoneNo>+1-650-000-000</DealerPhoneNo>
   <DealerEmail>info@frontier_ford.com</DealerEmail>
   <DealerURL>www.frontier_ford.com</DealerURL>
   <DealerOpened>1953</DealerOpened>
   <DealerBBBRating>5</DealerBBBRating>
   <DealerServiceRating>10</DealerServiceRating>
</infod:VocabularyInstanceVocabularyBody>
</infod:CreatePropertyVocabularyInstance>
```

Returned EPR:

http://www.carcommunity.com/CCInfoDRegistry/CarDealer/Frontier_Ford

Step 3: Car Dealer creates Association with Data Vocabulary

Request message:

```
<infod:AssociateVocabulary>
<infod:AssociateVocabularyName>
CarsAtFrontierFord
</infod:AssociateVocabularyName>
<infod:AssociateVocabularyDescription>
A full list of cars at Frontier Ford
</infod:AssociateVocabularyDescription>
<infod:AssociateEntityIdentifier>
<wsa:Address>:http://www.carcommunity.com/CCInfoDRegistry/CarCommunityDeale
rVocabEPR</wsa:Address>
</infod:AssociateEntityIdentifier>
<infod:AssociateVocabularyIdentifier>
<wsa:Address>:http://www.carcommunity.com/CCInfoDRegistry/CarCommunityCarVo
cabEPR<wsa:Address>
</infod:AssociateEVocabularyIdentifier>
</infod:AssociateVocabulary>
```

Returned EPR:

http://www.carcommunity.com/CCInfoDRegistry/Association/Frontier_FordCars

Step 4: Car Buyer added as a Consumer and Community Member

a) Registration of Car Buyer as Consumer

Request message:

```
<infod:CreateConsumer>
   </infod:WSEntityIdentifier>
<wsa:Address>http://www.carcommunity.com/CarBuyerEmail</wsa:Address>
</infod:WSEntityIdentifier>
<infod:ConsumerName>Susan Maria Callas</infod: ConsumerName>
<infod:ConsumerDescription>Buyer of fancy cars </infod:ConsumerDescription>
<infod:PropertyConstraints>Car_Dealer (Dealer: Years in business > 10
years, BBB rating > 3, Service rating > 10) </infod:PropertyConstraints>
</infod:CreateConsumer>
```

Returned EPR:

http://www.carcommunity.com/CCInfoDRegistry/Consumer/Susan_Maria_Callas

b) Registration of Car Buyer as Community Member

Request message:

```
<infod:CreatePropertyVocabularyInstance>
<infod:VocabularyInstanceEntityReference>
<wsa:Address>http://www.carcommunity.com/CCInfoDRegistry/Consumer/Susan_Mar
ia_Callas</wsa:Address>
</infod:VocabularyInstanceEntityReference>
<infod:VocabularyInstanceVocabularyReference>
<wsa:Address>http://www.carcommunity.com/CCInfoDRegistry/CarCommunityBuyerV
ocabEPR</wsa:Address>
   </infod:VocabularyInstanceVocabularyReference>
<infod:VocabularyInstanceVocabularyBody>
NOTE: this would need to be encoded correctly (escaped etc.)
e.g. "<" becomes &lt
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.w3.org/INFOD/Entity"
            targetNamespace="http://www.w3.org/INFOD/Entity">
   <BuyerName>Susan Maria Callas</BuyerName>
   <BuyerLocatiom>15998 Portola Off Road, Portola Valley,
CA</BuyerLocation>
   <BuyerPhoneNo>+1-650-000-0000</BuyerPhoneNo>
<BuyerEmail>callas&opera.music</BuyerEmail>
<BuyerCRRating>705</BuyerCRRating>
<BuyerInterest>Cars (Make = 'Italian' and Model = 'Sport' and Year < 1995
and ExtColor "Red') = </BuyerInterest>
</infod:VocabularyInstanceVocabularyBody>
</infod:CreatePropertyVocabularyInstance>
```

Returned EPR:

http://www.carcommunity.com/CCInfoDRegistry/CarBuyerr/Susan_Maria_Callas

Step 5: Car Buyer added as a Subscriber

Request message:

```
<<infod:CreateSubscriber>
   </infod:WSEntityReference>
```

```
<wsa:Address>http://www.carcommunity.com/CarBuyerEmail</wsa:Address>
</infod:WSEntityReference>
<infod:SubscriberName>Susan Maria Callas</infod:SubscriberName>
<infod:SubscriberrDescription>Interested in fancy
cars</infod:SubscriberDescription>
```

Returned EPR:

http://www.carcommunity.com/CCInfoDRegistry/Subscriber/Susan_Maria_Callas

Step 6: Car Buyer adds Subscription

Request message:

```
<infod:CreateSubscription>
<infod:SubscriptionName>
SusanMariaCallasFancyCars
</infod:SubscriptionName>
<infod:SubscriptionDescription>
   Maria Callas is looking for fancy cars
</infod:SubscriptionDescription>
   <infod:SubscriberReference>
<wsa:Address>http://www.carcommunity.com/CCInfoDRegistry/Subscriber/Susan_M
aria_Callas</wsa:Address>
   </infod:SubscriberReference>
<infod:DataConstraints>
CarCommunityCarVocabEPR.Make = 'Italian'
CarCommunityCarVocabEPR.Price = < 250000
   </infod:DataConstraints>
<infod:PropertyConstraints>
Distance (CarCommunityDealerVocabEPR.Location,
http://www.carcommunity.com/CCInfoDRegistry/CarBuyerr/Susan_Maria_Callas) <
25</infod:PropertyConstraints>
</infod:CreateSubscription>
```

Returned EPR:

http://www.carcommunity.com/CCInfoDRegistry/Subscription/SusanMariaCallasFancyCars

Subscribers could determine the information they like to see; this has been omitted in order to reduce complexity

Step 7: INFOD registry sends Subscription Notification to Car Dealers

The text below gives an example XML for a message body. The type of the body is infoDMsg:BodyType

```
<Body>
<infod:SubscriptionNotification>
<infod:SubscriptionReference>
<wsa:Address>http://www.carcommunity.com/CCInfoDRegistry/Subscription/Susan
MariaCallasFancyCars<wsa:Address>
</infod:SubscriptionReference>
<infod:StaticConsumers>
<wsa:Address>http://www.carcommunity.com/CCInfoDRegistry/Consumer/Susan_Mar
ia_Callas<wsa:Address>
</infod:StaticConsumers>
<infod:DataConstraints>
fn:doc('INFODRegistry.xml')/datavocabularies/infodDataVocabulary/CarCommuni
tyCarVocab(Make ='Italian' andPrice = < 250000)
         </infod:DataConstraints>
```

```
   </infod:SubscriptionNotification>
</Body>
```

Step 8: Car Dealer sends Message to Buyer

The car dealer receives the message using the CONSUME operation. It has the same forma as the WS-Notification; the consumer could use NOTIFY as alias.

Step 9: Car Dealer sends Message to Buyer

The text below gives an example XML for a message body delivered by the car dealer (publisher) to the car buyer (consumer). The type of the body is infoDMsg:BodyType

```
<Body>
<Type>XML</Type>
<Length>as calculated</Length>
<Data>
<CarMake>Ferrari</CarMake>
<CarModel>Roma</CarModel>
<CarYear>1990</CarYear>
<CarType>Sports</CarType>
<CarExtColor>Red</CarExtColor>
<CarIntColor>White</CarIntColor>
<CarDealer>Frontier Ford</CarDealer>
<CarPrice>$150000</CarPricer>
</Data>
</Body>
```

Step 10: Car Buyer receives Message from Car Dealer

The car buyer receives the message using the CONSUME operation. It has the same forma as the WS-Notification; the consumer could use NOTIFY as alias.

3.4 Security

Existing security technology allows the protection of the data.

A message is only delivered when the consumer has the right to see (access) the data.

3.5 Performance

Matching publishers and buyers in large communities such as the car dealer/buyer may require the *mutual filtering* of a large set of constraints. Creating efficient indices for constraints will be very important.

The performance of the actual message traffic between publisher and consumers and not impacted by the INFOD registry.

3.6 Requirements Implied

R1	Constraints, which restrict the flow of messages from publishers to consumers, must be composable.	The constraints contained in subscriptions must be composed with the constraints specified by the car dealers and car buyers

R2	Publishers should be able to describe their available messages, events and states in terms of a vocabulary.	Car dealers need to define what information about cars is available and what information people can receive.
R3	Subscribers must be able to constrain messages based on message content and publisher information.	Car buyers, acting as their own subscribers, must be able to select publishers based on their properties.
R4	Publishers must be able to choose what messages to publish based on consumer and subscriber information.	Car dealers must be able to restrict consumers according to their policies (constraints on car buyers)
R5	Consumers must be able to constrain messages based on message content, publisher information and subscriber information.	Car buyers, acting as their own subscribers, need to define which message they receive from which car dealer.
R6	Any component can request that it be notified by the registry, via WSN, of changes that the component considers interesting.	Car dealers need to know which subscriptions are relevant to them.

832

4 Sensor Networks Use Case

4.1 Introduction

We describe the requirement and use for data dissemination for *Sensor*[2] networks and applications that run on them. A sensor is typically a piece of hardware that detects an aspect of the environment. A sensor may also be a software module that acts as a source of data. Examples of sensors include radiation detectors, cameras, etc.. The sensors we consider here have a means of communicating with *Nodes* (also known as *Sensor Data Hubs*) which are compute entities that aggregate, dispatch, and receive data over a network connection. We consider the Node as the publisher of the data since it acts as a proxy for sensors.

*Application*s are the computational processing modules that operate on sensor network data. Applications model the intelligence that produces utility from the sensor network. Although the usage of the term applications may include a broader function in the sensor network, for purposes of this use case we use the term application for a single software module with well-defined inputs and outputs. Examples of applications include modules that perform data archiving, alert detection, and chemical concentration analysis.

The sensor networks we consider here have sufficient computing power and network bandwidth to participate in a middleware that supports data exchange and sharing.

4.2 Actors

Actors are the Nodes (equivalent in functionality are the SDH), Applications (the AlertListener), and the INFOD Registry.

Sensors are the producers of data and consumers of commands and data.

Nodes are publishers of data in the sensor network. Nodes may also consume directives such as commands and data from applications (not illustrated here).

Applications are consumers of data from the publishers. Applications also publish data consumed by the Nodes as well as other Applications, and so they will appear in the sensor network as publishers as well. We focus on an alert listener application

The INFOD Registry manages the vocabularies and subscriptions etc.

4.3 Scenarios

The scenario focusses narrowly on the following activities:

- The creation of the publishing entity.
- An application subscribing to sensor alerts
- A Node publishing Information

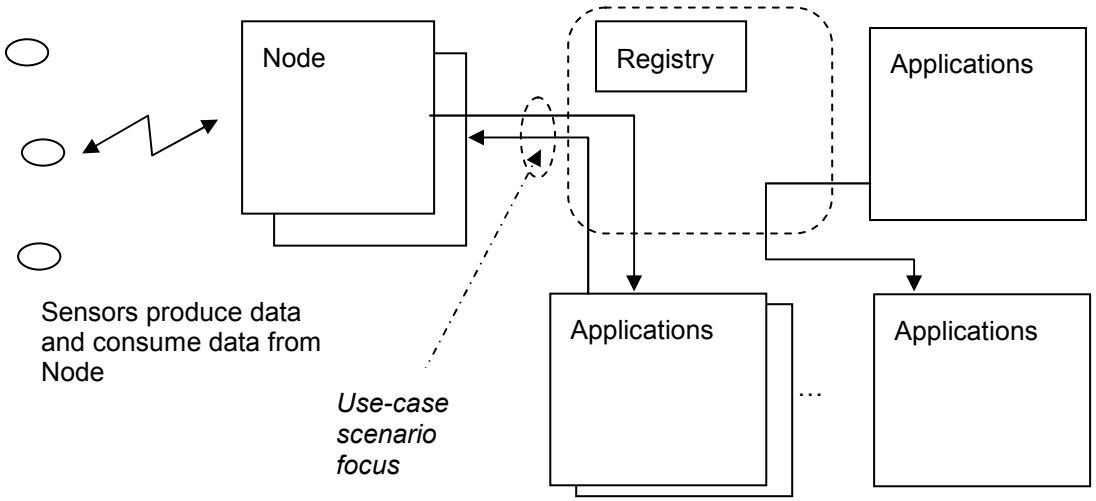
[2] We use the term sensor networks to refer to both sensor and actuator networks that cover a feedback loop of sense, response, and actuation. Actuators are entities that have the inverse behavior of sensors, that is, they effect actions on the environment (e.g., a camera turn, or change in a sensitivity setting of a sensor).

865 • An application consuming Information

866 XML schema and data are presented in tables; Operations are provided in each sub-section;

867 Figure 1 describes the scenario pictorially. We consider here the scenario of a Node sending a
868 message to an application that consumes it (for recording and displaying alerts). We do not discuss
869 the production and consumption of data between individual Sensors and Nodes in this use case. (The
870 relationship between the Sensors and the Nodes is also a publisher and consumer type relationship
871 with the Sensor publishing data and the Node consuming based on certain filters that the Node may
872 impose on the data that it receives.)

873



874

875 **Figure 5.4-14-2: Sensor Network Component Diagram**

876 4.3.1 Sensor and Applications – Common Vocabulary of the Community

Name of Predicate	Comment
Id	Sensor Source unique identification
Name	Org. Name
Spatio-temporal Location	X, Y, Z, t
(Feature)Type	Type of data source
Data Ranges	Range of detection
Owner	Sensor or Org. Owner
Data schema EPR	Pointer to data-schema itself
Confidence level	Accuracy
Functional label	For case-based access

877 **Table 5-1: Community Vocabulary**

Name of Predicate	Comment
ID	IP address of SDH
Name	Name of SDH
StateChange1	Describes a type of Event that is published by the SDH. Example could be: ChemSpill Temperature Change ZoneChangeEvent ZoneAccessException SecurityAlert These events contain a pointer to the relevant data schemas
StateChange2…	

878 **Table 5-2: SDH Community Vocabulary**

879 We assume that there is a vocabulary registered (somewhere?) for the Gamma Radiation Sensor and
880 others like it as well as a community vocabulary for each of the SDH StateChanges. Also, a
881 vocabulary for Chemicals is required in this case which contains the MSDS (Material Safety Data
882 Sheet) information detailing the chemical components and safety procedures. In this example, the
883 MSDS information is published as a standard web-service accessible for every federal facility. As
884 soon as a new chemical product is released, the MSDS information is made available. Each INFOD
885 entity requiring information from that source is responsible to deploy a web service to this source (but
886 could also cache it since its stale data)

887 ### 4.3.2 Sensor Characteristics

Publisher Entity INFOD Components	Values

infod_Type	(Feature) Type = Gamma-Sensor
wsinfod _Name	Name = Gamma Radiation Sensor
wsinfod _Description	Nuc Safe Technologies
wsinfod _Data_Constraint	Reading Range[A, B]
wsinfod _Property_Constraint	Spatio-temporal Location = [X,Y,Z,t]
infod_Property_Constraint	Owner = dhs.tn.rad.nucsafe.*
infod_vocabulary	Data schema EPR = Rad_sensor_vocabulary(EPR-of xsd)
infod_Property_Constraint	Functional label = first_responder
infod_Property_Constraint	Confidence level = 10%

888 **Table 5-3: Node/Sensor as Publisher**

889 ### 4.3.3 Application Subscription

infod_Type	Subscription
infod_Name	AlertListener
infod_Description	Retrieve all current alerts
infod_Data_Constraint	Reading > C
infod_Property_Constraints	Sensors's X, Y, Z, t values within requested range
infod_Property_Constraint	wisinfod_Name == Gamma Radiation Sensor
infod_Property_Constraint	infod_vocabulary==Required_EPR of sensor community vocabulary

890 **Table 5-4: Application Subscription**

Name of Predicate	Values
infod_Type	Subscription
infod_Name	ChemSpillDetection
infod_Description	Detection of Chemical Spills
infod_Data_Constraint	Sensor Data Inventory: SHOW (Chemical(Obs.SensorReading.Data), Obs.SensorReading.Time) FOR Obs.SensorReading.SubType = SPILL
infod_Property_Constraint	Publisher: Obs.SensorReading.Type = RFID Sensor
infod_Property_Constraint	infod_vocabulary==Required_EPR of Chemical, Chemical_Spill and RFID_Sensor community vocabulary (static, enabled through other web services)
infod_Property_Constraint	Consumer: Authorized to receive Spill Information && Inventory(Chemical).Organization == Consumer.Organization

891 **Figure 5-4: SDH as Subscriber to detect Chemical Spills**

INFOD_Type	Subscription
INFOD_Name	MyChemSpills
INFOD_Description	Notify me of Chemical Spills in my Zone
INFOD_Data_Constraint	Chemical_Spill.RFID.Zone=My Current Zone && Chemical_Spill.Chemical.HazardLevel > 3 && Chemical_Spill.time = within working hours
INFOD_Property_Constraints	Publisher: Name = NASA SDH
infod_Property_Constraints	infod_vocabulary==Required_EPR of Chemical & Chemical_Spill community (static, enabled through other web services)

892 **Figure 5-5: End-user / application Subscription for ChemSpill to SDH**

4.3.4 Scenario Steps

Nodes/SDH (Publishers) and AlertListener (Subscriber and Consumer) registers the vocabularies with the INFOD Registry.

Node adds itself as a publisher.

AlertListener adds itself into the INFOD Registry as a subscriber.

AlertListener adds itself as a consumer.

Nodes and AlertListeners add the relevant subscriptions to the INFOD Registry.

Nodes creates associations between the instances and the data vocabulary.

4.3.5 Examples of XML messages

The following text gives an example of part of the XML messages that would be sent to and from the INFOD Registry when each of the relevant interfaces is called:

Step 1: Register the vocabularies with the INFOD Registry.

a) Registration of Sensor Vocabulary

Request message:

```
<infod:RegisterDataVocabulary>
<infod:VocabularyName>SensorDataVocabulary </infod:VocabularyName>
<infod:VocabularyLanguage>XML Schema(Namespace/URI of DataFormat)
</infod:VocabularyLanguage>
<infod:VocabularyBody>
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns:ident=http://www.w3.org/INFOD/Entity
         xmlns:sn=http://infod.sensornetwork.com/snschema
           targetNamespace="http://www.w3.org/INFOD/Entity">
   <xsd:element name = "ID" type = "xsd:string"/>
   <xsd:element name = "Name" type = "xsd:string"/>
   <xsd:element name = "Type" type = "xsd:time"/>
   <xsd:element name = "SpatioTemporalLocation" type = "sn:coordinates1"/>
   <xsd:element name = "DataRange" type = "sn:range"/>
      <xsd:element name = "StateChange" type = "xsd:string" />
</infod:VocabularyBody>
</infod:RegisterDataVocabulary>
```

Response message (for success case):

```
<infod:RegisterVocabularyResponse>
<infod:INFODVocabularyIdentifier>
<wsa:Address>http://infod.sensornetwork.com/vocabEPR </wsa:Address>
</infod:INFODVocabularyIdentifier>
</infod:RegisterVocabularyResponse>
```

b) Registration of Application (Alert Subscriber) Vocabulary

Request message:

```
<infod:RegisterPropertyVocabulary>
<infod:VocabularyName>AlertSubscriberVocabulary </infod:VocabularyName>
<infod:VocabularyBody>
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.w3.org/INFOD/Entity"
            targetNamespace="http://www.w3.org/INFOD/Entity">
   <xsd:element name = "ApplicationType" type = "xsd:string"/>
   <xsd:element name = "Name" type = "xsd:uri"/>
   <xsd:element name = "ListenSourcesType" type = "xsd:string"/>
</infod:VocabularyBody>
</infod:RegisterPropertyVocabulary>
```

The response message would be very similar to that for the previous step and would include the EPR of the INFOD vocabulary identifier. In this example the EPR returned is:

http://infod.sensornetwork.com/alertlistenervocabEPR

Step 2: Node added as a Publisher.

Request message:

```
<infod:CreatePublisher>
<infod:WSEntityIdentifier>
<wsa:Address> http://infod.sensornetwork.com/publisher/node1 </wsa:Address>
   </infod:WSEntityIdentifier>
<infod:PublisherName>Node1</infod:PublisherName>
</infod:CreatePublisher>
```

Again, there would be a similar response message which would include the EPR of the INFOD entity identifier. In this example the EPR returned is:

http://infod.sensornetwork.com/publisherEPR/node1

Step 3: Alert Listener added to the INFOD Registry as a Subscriber.

Request message:

```
<infod:CreateSubscriber>
<infod:SubscriberName>AlertListener1 </infod:SubscriberName>
</infod:CreateSubscriber>
```

In this example the returned EPR is:

`http://infod.sensornetwork.com/subscriberEPR/AlertListener1`

Step 4: Alert Listener also added as a Consumer.

Request message:

```
<infod:CreateConsumer>
<infod:WSEntityIdentifier>
<wsa:Address>http://infod.sensornetwork.com/consumer/A lertListener1
</wsa:Address>
   </infod:WSEntityIdentifier>
<infod:ConsumerName>AlertListener1</infod:ConsumerName>
</infod:CreateConsumer>
```

In this example the returned EPR is:

`http://infod.sensornetwork.com/consumerEPR/AlertListener1`

Step 5: Alert Listener adds the relevant subscriptions to the INFOD Registry.

Request message:

```
<infod:CreateSubscription>
<infod:SubscriptionName>
GetNodeSDHAlerts
</infod:SubscriptionName>
<infod:SubscriptionDescription>
   Listening for Node and SDH Alerts
</infod:SubscriptionDescription>
   <infod:SubscriptionSubscriberIdentifier>
<wsa:Address> http://infod.sensornetwork.com/subscriberEPR/AlertListener1
</wsa:Address>
   </infod:SubscriptionSubscriberIdentifier>
<infod:SubscriptionDataConstraints>
         http://infod.sensornetwork.com/vocabEPR.StateChange =
"TemperatureChange"
   </infod:SubscriptionDataConstraints>
<infod:SubscriptionPropertyConstraints>
http://infod.sensornetwork.com/vocabEPR.Type="Thermometer"
</infod:SubscriptionPropertyConstraints>
</infod:CreateSubscription>
```

In this example the returned EPR is:

```
http://infod.sensornetwork.com/subscriptions/AlertListener1
```

Step 6: Node/SDH creates associations between the publisher and the data vocabulary.

Request message:

```
<infod:AssociateVocabulary>
<infod:AssociateVocabularyName>PublisherAndDataVocabAssociation
</infod:AssociateVocabularyName>
<infod:AssociateVocabularyEntityIdentifier>
<wsa:Address> http://infod.sensornetwork.com/publisherEPR/node1
</wsa:Address>
</infod:AssociateVocabularyEntityIdentifier>
<infod:AssociateVocabularyVocabularyIdentifier>
<wsa:Address> http://infod.sensornetwork.com/vocabEPR </wsa:Address>
</infod:AssociateVocabularyVocabularyIdentifier>
</infod:AssociateVocabulary>
```

Step 7: Node/SDHs create instances of the relevant property vocabularies for the publisher, subscriber and consumers.

Request message:

```
<infod:CreatePropertyVocabularyInstance>
<infod:VocabularyInstanceEntityReference>
<wsa:Address> </wsa:Address>
</infod:VocabularyInstanceEntityReference>
<infod:VocabularyInstanceVocabularyReference>
<wsa:Address> http://infod.sensornetwork.com/vocabEPR </wsa:Address>
   </infod:VocabularyInstanceVocabularyReference>
<infod:VocabularyInstanceVocabularyBody>
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.w3.org/INFOD/Entity"
            targetNamespace="http://www.w3.org/INFOD/Entity">
   <xsd:element name = "ID" type = "xsd:string"/>
```

```
      <ID>uuidAA298320</ID>
   <Name>node1</Name>
      <Type>chemicalAggregator</Type>
   <SpatioTemporalLocation><lat>85.2</lat><long>-50</long>
      </SpatioTemporalLocation>
   <DataRange><low>0</low><high>124</high></DataRange>
</infod:VocabularyInstanceVocabularyBody>
</infod:CreatePropertyVocabularyInstance>
```

In this example the returned EPR is:

http://infod.sensornetwork.com/vocabinstances/PublisherVocabInstanceEPR

Step 8: The AlertListener examines possible publishers.

Request message:

The format of the request message for a GetMetadata operation that finds out types of publishers with a certain type is:

```
<infod:GetMetadata>
<infod:INFODQueryExpression Dialect="SQL">
InfodRegistry: SHOW (InfodRegistry.Entities.*) FOR Entity.Type = Publisher
AND http://infod.sensornetwork.com/vocabEPR.Type="Thermometer"
</infod:INFODQueryExpression>
</infod:GetMetadata>
```

Step 9: The Publisher generates messages for the relevant consumers.

Output message to consumers:

The text below gives some example XML for a message body. The type of the body is infoDMsg:BodyType

```
<Body>
<Type>XML</Type>
<Length>100</Length>
<Data type="StateChange">
    <Source>SourceEPR</Source>
    <Type>TemperatureChange</Type>
</Data>
</Body>
```

4.4 Security

The registration of the subscription should be authenticated against the application. This can appear as a service outside INFO-D. A certificate infrastructure enables applications to create their subscriptions in the registry.

Once a sensor publishes data of interest (e.g., a data element that satisfies a subscription) the data will need to be transmitted securely to the correct consumer application. This means that the pub-sub (or dissemination) channel needs to support privacy, and authentication, and non-repudiation – this will be provided outside INFO-D. There is a need for a way of specifying the security requirement to the publisher and consumer when the registry finds that a publisher's data matches a subscription.

4.5 Performance

The typical requirement on alert messages is a detection and dispatch in under a second.

4.6 Requirements Implied

R1	Constraints, which restrict the flow of messages from publishers to consumers, must be composable.	Subscriptions can compose vocabulary elements.
R2	Publishers should be able to describe their available messages, events and states in terms of a vocabulary.	Nodes and SDH devices describe their available data as a vocabulary.
R3	Subscribers must be able to constrain messages based on message content and publisher information.	Subscriptions can select based on vocabulary instances.
R4	Publishers must be able to choose what messages to publish based on consumer and subscriber information.	Nodes and SDH can impose publish constraints based on receiver vocabulary (not illustrated in above use-case)
R5	Consumers must be able to constrain messages based on message content, publisher information and subscriber information.	Shown similar to R3.
R6	Any component can request that it be notified by the registry, via WSN, of changes that the component considers interesting.	Nodes and Alert Listeners can talk directly to the registry (not illustrated in above use-case).

5 3rd Party Delivery of Query Results Use Case

5.1 Introduction

Clients of databases may be interested in disseminating query results to a set of consumers. Therefore, it should be possible to send query results to selected consumers; consumers should also be able to receive query results as they become available.

5.2 Actors

3rd party delivery of query results requires the following actors:

- (Database) clients as providers of queries-
- Owners of tables/files/collections as publishers of query results; the tables/files/collections become associations
- Clients as recipients of query results
- Owners of tables/files/collections or clients in the role of subscribers as providers of Continuous Queries

5.3 Scenarios

3rd party delivery can leverage the INFOD infrastructure on various levels. The INFOD registry can be used to identify the proper publishers and consumers by using the getMetaData operation.

Continuous Queries can be treated like subscriptions and handled by the INFOD registry directly. The Continuous Queries can be issued by clients or owners.

These scenarios will be covered:

- **Publishers identified by Clients, Consumers identified by Publishers** - a client uses the INFOD registry to identify publishers, sends a query request to the publisher along with consumer constraints. The publisher uses the INFOD registry to determine the proper consumers. The model allows publishers to determine consumers based on query results.
- **Publishers and Consumers identified by Clients** - a client uses the INFOD registry to identify the proper publishers and consumers. The query request with the list of consumers is directed to the selected publishers.
- **Continuous Queries** - a client or an owner of a table/file/collection acting as subscriber sends a Continuous Query to the INFOD registry; the INFOD registry evaluates the request and notifies each publisher along with the list of consumers. The INFO registry notifies effected databases of any changes. This scenario obviously assumes support for Continuous Queries.

These scenarios, like any other scenario, require the specification of vocabularies and the creation of entities and communities.

Consumers will only receive messages if they are entitled to do so according to the security settings.

Note: In it simplest case, the client knows the publisher and the consumers, 3rd party delivery does not require any INFOD support.

Details of the following steps will not be described; they are covered in the proceeding use cases.

5.3.1 Creating Data Vocabularies

Data vocabularies are best created by importing vocabularies of participating data bases into an INFOD registry. We will assume that data base repositories are able to (automatically) create, modify and drop data vocabularies as a result of operations involving vocabularies.

5.3.2 Creating Property Vocabularies

Property vocabularies have to be added if the creation of communities is required. This is done – as described in previous use cases – by defining property vocabularies and creating instances that are associated to INFOD entities.

5.3.3 Creating Entities

The description of entities can be driven by the database repository (repositories) or by using directly the INFOD operations. The owner of tables/files/collections can become publishers; all or a subsets of the clients can become consumers and subscribers. Obviously, consumers and publishers that are not in database registries can be added.

5.3.4 Creating Vocabulary Associations

One more step is required; publishers have to be associated to vocabularies. This can be done by the database registries by associating the owners of tables/files/collections to the appropriate vocabulary.

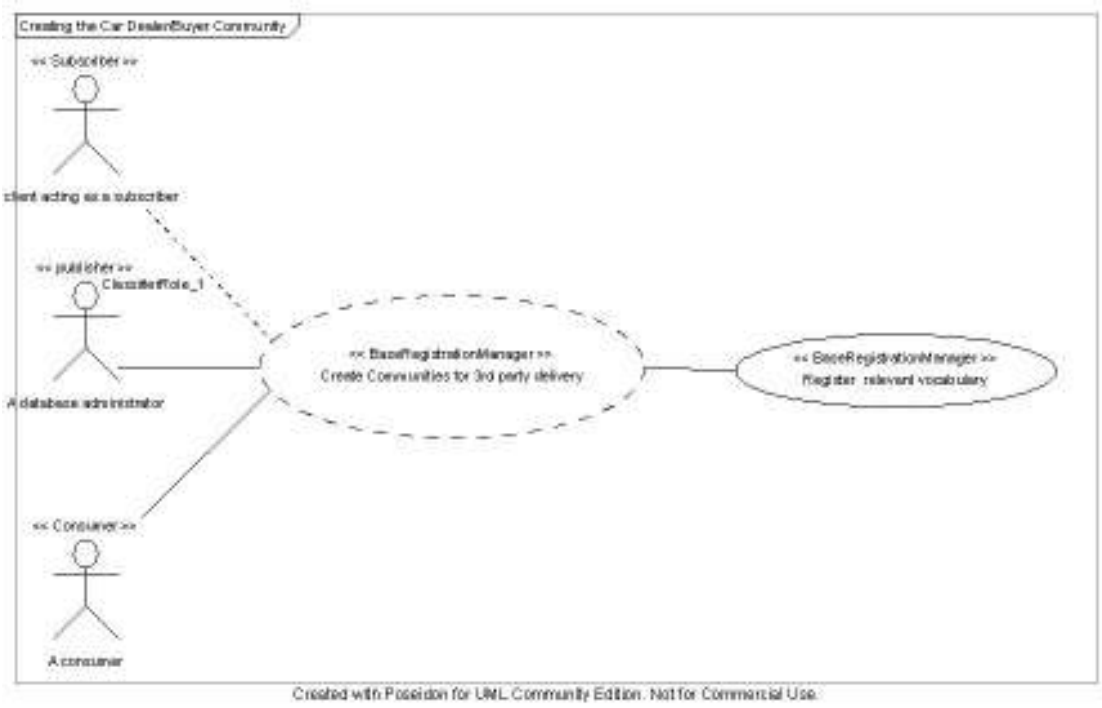


Figure 4-5-1: 3rd Party Delivery Use Case - Creating Communities

5.3.5 Examples of XML Messages

The examples are constructed following the discussion in Appendix B.

Case 1: Publishers identified by Clients, Consumers identified by Publisher

Step 1: Client identifies Publishers

The setup from the Car Use Case is used to formulate the getMetaData the following XQuery statement.

```
For $CD in doc('www.carcommunity.com.xml')//CarDealers
$Asc in doc('www.carcommunity.com.xml')//Associations
$Voc in doc('www.carcommunity.com.xml')//Vocabularies
$Pub in doc('www.carcommunity.com.xml')//Publishers

Where

$CD/DealerOpened < 1996 and $CD/DealerBBBRating > 3 and
$CD/DealerServiceRating >10 and Fn:WithinDist($CD/DealerLocation, '15998
Portola Off Road, Portola Valley, CA' ) < 25 and
$Voc/infod:VocabularyName = 'CarCommunityCarVocab' and
$CD/infod:VocabularyInstanceEntityReference=$Asc/infod:AssociateEntityIdent
ifier and
$CD/infod:VocabularyInstanceEntityReference = Fn:GetEPR($Pub) and
$Asc/infod:VocabularyInstanceEntityReference = Fn.GetEPR($Voc)

Return

($Pub/infod:WSEntityIdentifier)
```

Step 2: Client sends Requests to Publishers

This is the same request as specified in the Car Use Case, see Step 7: INFOD registry sends Subscription Notification to Car Dealers ..

Step 3: Publishers identify Consumers

The setup from the Car Use Case is used to formulate the getMetaData the following XQuery statement.

```
For $CB in doc('www.carcommunity.com.xml')//CarBuyer
$Con in doc('www.carcommunity.com.xml')//Consumer

Where

$CB/BuyerCRRating > 700 and Fn:WithinDist($CB/BuyerLocation, '101 Auto Row,
Redwood City, CA 94065') and
$CB/infod:VocabularyInstanceEntityReference = Fn:GetEPR/($Con)

Return

($Con/infod:WSEntityIdentifier)
```

Note: This XQuery is added to show how publishers can find consumers; in this specific use case there is one specific consumer.

Step 4: Publishers send Results to Consumers

This step has been covered by the previous use cases.

Case 2: Publishers and Consumers identified by Clients

Step 1: Client identifies Publishers and Consumers

The setup from the Car Use Case is used to formulate the getMetaData the following XQuery statement.

```
For $Pub in doc('www.carcommunity.com.xml')//Publishers,
$Con in doc('www.carcommunity.com.xml')//Consumers,
$Voc in doc('www.carcommunity.com.xml')//Vocabularies,
$Asc in doc('www.carcommunity.com.xml')//Associations,
$CD  in doc('www.carcommunity.com.xml')//CarDealers,
$CB  in doc('www.carcommunity.com.xml')//CarBuyer,

Where

Fn:WithinDist($CD/DealerLocation, '15998 Portola Off Road, Portola Valley,
CA ') < 25 and

$Voc/infod:VocabuklaryName = 'CarCommunityCarVocab' and

fn:evaluate($Pub/PublisherPropertyConstraint, $CB) = 1 and
fn:evaluate($Con/ConsumerProperttyConstraints, $CD) = 1 and

$CD/infod:VocabularyInstanceEntityReference =
$Asc/infod:VocabularyInstanceEntityReference and
$Asc/infod:VocabularyInstanceEntityReference = Fn:GetEPR/($voc)

$CB/infod:VocabularyInstanceEntityReference = Fn:GetEPR/($con)

return

    <PublisherRecipientMapping>
       {
         $pub, $Con
       }
    </PublisherRecipientMapping>
```

Note: This XQuery is more complex than necessary to show mutual filtering between publishers and consumers; in this specific use case there is one specific consumer.

Step 2: Client sends Request to Publishers

This is the same request as specified in the Car Use Case, see Step 7: INFOD registry sends Subscription Notification to Car Dealers on page63. on page

Step 3: Publishers send Results to Consumers

This is the same request as specified in previous use case.

Case 3: Continuous Queries

Step 1: Client sends Requests to INFOD Registry

This is the same request as specified in the Car Use Case see Step 6: Car Buyer adds Subscription.

Step 2: INFOD identifies Publishers and Consumers and notifies Publishers

This is the same request as specified in the Car Use Case, see Step 7: INFOD registry sends Subscription Notification to Car Dealers .

Note: The INFOD registry will inform any publisher if any change in the registry changes the respective consumer list.

Step 3: Publishers send Results to Consumers

This step has been covered by the previous use cases.

5.4 Security

If consumers of query results are database clients, the database (fine grain) security can be used to ensure that data are only distributed to those consumers that are entitled to receive the query results.

If consumers are not database clients the constraint information in the INFOD registry will be used to ensure that data are only distributed to those consumers that are entitled to receive the query results.

5.5 Performance

The identification of the databases and consumers with an INFOD registry has to be added to the response time. The performance of such a request is highly dependent on the implementation of that registry.

The performance of the creation of the query result and the notification of the clients depends on the databases acting as query engine and as publisher.

5.6 Requirements Implied

R1	Constraints, which restrict the flow of messages from publishers to consumers, must be composable.	The constraints contained in queries must be composed with the constraints related to security

R2	Publishers should be able to describe their available messages, events and states in terms of a vocabulary.	Depending on the support of the database, there may be access to messages, events, and states. The focus is the access to a single state.
R3	Subscribers must be able to constrain messages based on message content and publisher information.	N/A - clients acting as subscribers determine the content of the message.
R4	Publishers must be able to choose what messages to publish based on consumer and subscriber information.	N/A – databases acting as publishers react only to requests of clients
R5	Consumers must be able to constrain messages based on message content, publisher information and subscriber information.	Consumers must be able to reject messages based on content and client acting as subscriber.
R6	Any component can request that it be notified by the registry, via WSN, of changes that the component considers interesting.	If a database shares a repository with other databases it needs to be informed of changes. These changes are of special importance if the database offers Continuous Query support.

1239

6 Editor and Contributor Information

Vijay Dialani
IBM Corporation
Almaden Research Center
650 Harry Road,
San Jose, CA 95120-6099
vdialani@us.ibm.com

Steven Davey
EPCC,
University of Edinburgh,
James Clerk Maxwell Building,
Mayfield Road,
Edinburgh EH9 3JZ,
United Kingdom.
sdavey@nesc.ac.uk

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

Steve Fisher
Rutherford Appleton Laboratory (CCLRC)
Chilton
Didcot
Oxon OX11 0QX, UK
s.m.fisher@rl.ac.uk

Dieter Gawlick
Oracle Corporation
500 Oracle Parkway
Redwood Shores
CA 94065
dieter.gawlick@oracle.com

Christopher Kantarjiev
Oracle Corporation
500 Oracle Parkway
Redwood Shores
CA 94065
chris.kantarjiev@oracle.com

Cecile Madsen
IBM Silicon Valley Laboratory
555 Bailey Avenue
San Jose, CA 95141
madsen@us.ibm.com

Susan Malaika,
IBM Corporation,
Silicon Valley Laboratory,
555 Bailey Avenue,
San Jose, CA 95141,
USA.
malaika@u.ibm.com

Shailendra Mishra
Oracle Corporation
500 Oracle Parkway
Redwood Shores
CA 94065
shailendra.mishra@oracle.com

Mallikarjun Shankar
Oak Ridge National Laboratory
Oak Ridge
TN 37831
shankarm@ornl.gov

7 Acknowledgements

The INFOD Working Group of the Global Grid Forum has benefited from many people contributing to discussions within the group.

8 Intellectual Property Statement

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF 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 GGF Executive Director.

9 Full Copyright Notice

10 References

[EVAL]

. Dieter Gawlick, Dmitry Lenkov, Aravind Yalamanchi, Lucy Chernobrod, "Applications for Expression Data in Relational Database Systems," *icde*, p. 609, 20th International Conference on Data Engineering (ICDE'04), 2004

[OGSA]

I. Foster (Ed), H. Kishimoto (Ed), A. Savva (Ed), D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, R. Subramaniam, J. Treadwell, J. Von Reich. *The Open Grid Services Architecture, Version 1.0.* Global Grid Forum. GFD-I.030. 29 January 2005. http://forge.gridforum.org/projects/ggf-editor/document/GFD.30/en/1.

[OGSA Glossary]

J. Treadwell, *Open Grid Services Architecture Glossary of Terms*, GFD-I.044, January 25th 2005. http://forge.gridforum.org/projects/ggf-editor/document/GFD.44/en/1.

11 Appendix A – INFOD Patterns of Interaction

11.1 Introduction

INFOD is designed to support a variety of information dissemination patterns. The following patterns have been identified.

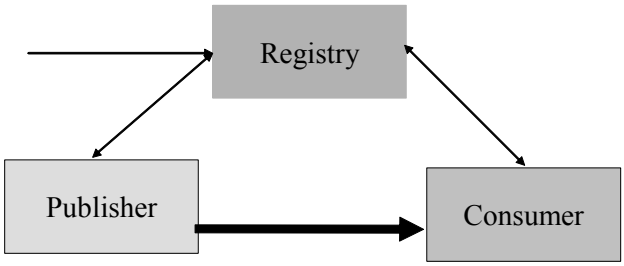


Figure 11-1: Base INFOD Service

- **No subscriptions** – publishers send messages to consumers of their choice. The following sub-pattern are identified:
  - Publishers use an INFOD registry to find consumers
  - Consumers use an INFOD registry to filter publishers
  - Publishers and consumers use an INFOD registry to share vocabularies
- **Subscriptions managed by an INFOD registry** – subscribers send subscriptions to an INFOD registry. The INFOD registry identifies publishers (and consumers) and sends subscriptions to publishers
- **Subscription based publications** - publishers create and deliver messages according to subscription requests.

11.2 Description of Patterns

11.2.1 No Subscriptions

The patterns of this section represent the basic form of disseminating information: publishers send messages to consumers of their choice.

The following information in the INFOD registry provides the context for this pattern:

- User data vocabularies - optional
- Publishers and consumers - optional
- Vocabulary associations - optional
- User property vocabularies - optional
- User properties - optional

The following functionality is available:

- Publishers query the INFOD registry to find matching consumers – requires consumer and optionally property vocabulary entries in INFOD registry.

Publishers should avoid querying the INFOD registry for each single message. Instead, publishers should group messages into classes and associated consumers to these classes and additionally cache the consumer information.

If requested, changes of the result set of any query trigger notification of the publisher. These notifications contain updated information about consumers.

- Consumer filter messages based on publishers properties - requires publisher and optionally property vocabulary entries in INFOD registry.

  Consumers can query the INFOD registry to find information about publisher using the information about publishers in the manifest section of the INFOD message.

  Consumers should avoid querying the INFOD registry for each single message. Consumers have to obtain and cache relevant information.

  If requested, changes to the result set of any query will trigger notification of the consumers. These notifications contain updated information about publishers.

- Publishers and consumers use vocabulary information in the INFOD registry before sending/consuming a message - requires user data vocabulary entries in INFOD registry.

  Publishers and consumers should avoid querying the INFOD registry for each message. Both, publishers and consumers, should obtain and the cache relevant vocabulary information.

  If requested, changes to the result set of any query will trigger notification of the publishers and consumers respectively. These notifications contain updated information about vocabularies.

Note: Publishers and consumers use only limited set services of the INFOD registry; e.g., publishers and consumers are not matched by the INFOD registry but do so using the GetMetaData interface.

11.2.2 Subscriptions - Managed by Registry

The association of publishers and consumers to a subscription is the responsibility of the INFOD registry.

The following information in the INFOD registry provides the context for this pattern:

- User data vocabularies - mandatory
- Publishers and consumers - mandatory
- Subscribers - mandatory
- Subscriptions - mandatory
- Vocabulary associations - mandatory
- User property vocabularies – optional
- User properties - optional

The following functionality is available:

- Subscribers manage subscriptions using the Create/Replace/DropSubscription operation.

  The INFOD registry determines which publishers are offering messages of interest using the vocabulary and vocabulary association information. The evaluation of all constraints that may limit the selection of publishers and consumers follows. The INFOD registry will make adjustment if subscriptions, publishers or consumers are created, replaced or dropped.

  If requested, changes that impact subscriptions will trigger notification of the publishers. These notifications contain updated information about subscriptions.

11.2.3 Subscription Based Publications

Publishers are often not able to determine which messages are of interest to consumers. Therefore, publishers must allow the creation of messages based on subscription referencing events or states.

The following information in the INFOD registry provides the context for this pattern:

- User data vocabularies - mandatory
- Publishers and consumers - mandatory
- Subscribers - mandatory
- Subscriptions - mandatory
- Vocabulary associations - mandatory
- User property vocabularies – optional
- User properties - optional

The following functionality is available:

- Subscribers specify what messages are published in reaction to events or state changes.

  Publishers have to support vocabularies that include events or even allow the specification of events. The associate language must allow filtering of events and also include the ability to define what messages need to be created to reaction to events and state changes respectively.

  The difference between this and the previous patterns is the functionality of the information source (vocabulary associations). In the previous patterns, publishers use vocabularies that represent messages; using this pattern publishers offer access to events and state transitions. Messages are created when an event satisfies certain constraints or when a state transition results in certain constraint becoming true.

  If requested, changes that impact subscriptions will trigger notification of the publishers and subscribers. These notifications contain updated information about vocabularies.

11.3 Outline of Operations

11.3.1 No Subscriptions

The following functionality is available:

- Publishers find relevant consumers in INFOD registry

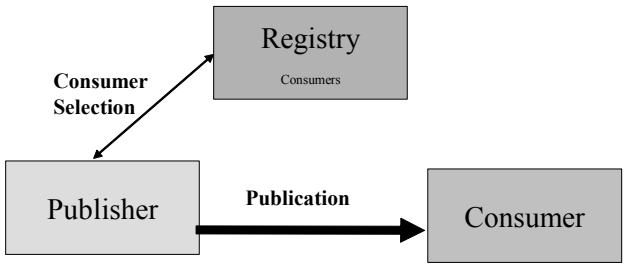


Figure 11-2 Base INFOD Service - No Subscription, Consumer Selection

Tasks:

- **Manage** consumer information in **INFOD registry** with
  - Create/Replace/DropConsumer operation - mandatory
  - RegisterPropertyVocabulary operation – optional
  - Create/DropPropertyVocabularyInstance operation - optional
  - UnregisterVocabulary operation - optional
- **Publishers** select consumers using the GetMetaData operation
  - If requested, publishers receive update notifications from the INFOD registry reflecting changes of consumer information
- **Publishers** create and send messages using WS-Notification message structure.

The INFOD registry provides up-to-date information about consumers.

- Consumers filter messages based on publishers properties

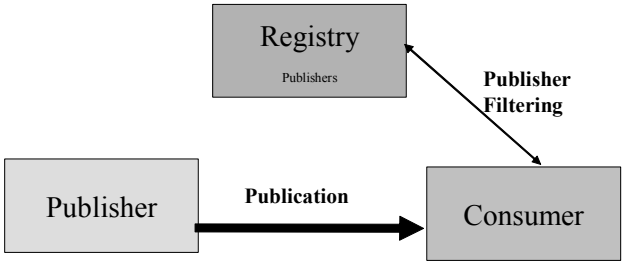


Figure 11-3 Base INFOD Service - No Subscription, Publisher Filtering

Tasks:

- **Manage** publisher information in **INFOD registry** with
  - Create/Replace/DropPublisher operation - mandatory
  - RegisterPropertyVocabulary operation – optional
  - Create/DropPropertyVocabularyInstance operation - optional
  - UnregisterVocabulary - optional
- **Consumers** find publisher information using the GetMetaData operation
  - If requested, consumers receive update notifications form the INFOD registry reflecting changes of publisher information
- **Publishers** create and send messages using WS-Notification message structure
- **Consumers** verifies publishers and consumes message

The INFOD registry provides up-to-date information about publishers.

- Consumer can filter message based on message content

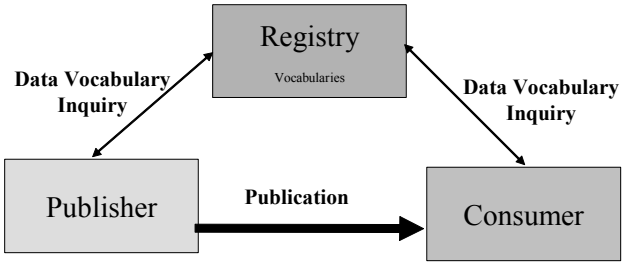


Figure 11-4 Base INFOD Service - Vocabulary Inquiry

Tasks:

- **Manage** vocabulary information in **INFOD registry** with
  - RegisteDataVocabulry[3] operation - mandatory
  - UnregisterVocabulry operation - optional
- **Publishers** find vocabularies using the GetMetaData operation
  - If requested, publishers receive update notifications reflecting changes of vocabularies
- **Consumers** find vocabularies using the GetMetaData operation
  - If requested, consumers receive update notifications reflecting changes of vocabularies
- **Publishers** create and send messages using WS-Notification message structure with a payload according to a vocabulary
- **Consumers** interprets message using vocabulary information and processes it

The INFOD registry provides up-to-date information about data vocabularies.

11.3.2 Subscriptions - Managed by Registry

- Publishers publish messages according to subscription directives

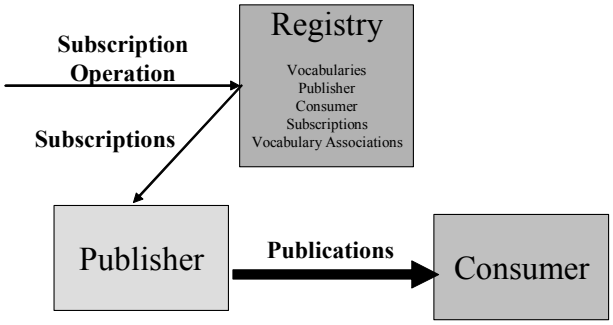


Figure 11-5 Base INFOD Service - Subscription Managed by INFOD Registry

[3] INFOD does not manage user data

Tasks:

- **Manage** information in **INFOD registry** with
  - Create/Replace/DropPublisher operation – mandatory
  - Create/Replace/DropSubscriber operation – mandatory
  - Create/Replace/DropConsumer operation – mandatory
  - Create/Replace/DropSubscription operation – mandatory
  - RegisterPropertyVocabulary operation – optional
  - Create/DropPropertyVocabularyInstance operation - optional
  - RegisteDataVocabulry operation - mandatory
  - UnregisterVocabulry operation - optional
- **Manage** vocabularies, publishers, consumers and vocabulary associations in **INFOD registry** using the Create/Replace/DropPublisher/Consumer and the Register/AddVersion/DeregisterVocabulary operation respectively
- **Subscribers** direct subscribe request to INFOD registry
  - **INFOD registry** determines relevant publishers
  - **INFOD registry** checks all constraints
  - **INFOD registry** notifies publisher about new subscriptions – these subscription requests reflect constraints in the INFOD registry
  - **INFOD registry** notifies publishers about altered/dropped subscriptions
- **Publishers** receive and process subscription request
- **Publishers** create and send messages to all relevant consumers using WS-Notification message structure
- **Consumers** receive message using the consume operation

By directing subscription request to the INFOD registry, subscribers can direct the INFOD registry to find relevant publishers, check all the constraints, and create a tailored subscription to each of the appropriate publishers. Furthermore, the INFOD registry informs publishers if changes in the directory information require changes in the subscription. This includes that a publication is retracted from some publisher while other publishers are added.

11.3.3 Subscription Based Publications

The only difference to the previous pattern is the structure and the language support for selected user data vocabularies. The subscriber chooses vocabularies, which support the creation of messages based on events or even supports the definition of events.

- Publishers create and publish messages according to subscription directives

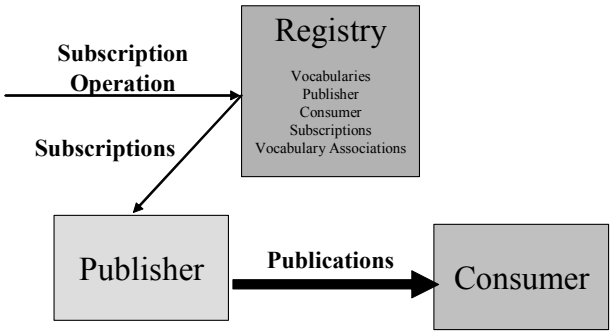


Figure 11-6 Base INFOD Service - Subscription Managed by INFOD Registry - Messages and Events Specified by Subscriber

Tasks:

- **Manage** information in **INFOD registry** with
  - Create/Replace/DropPublisher operation – mandatory
  - Create/Replace/DropSubscriber operation – mandatory
  - Create/Replace/DropConsumer operation – mandatory
  - Create/Replace/DropSubscription operation – mandatory
  - RegisterPropertyVocabulary operation – optional
  - Create/DropPropertyVocabularyInstance operation - optional
  - RegisteDataVocabulry operation - mandatory
  - UnregisterVocabulry operation - optional
- **Subscribers** direct subscribe request to INFOD registry
  - **INFOD registry** determines relevant publishers
  - **INFOD registry** checks all constraints
  - **INFOD registry** notifies publisher about new subscriptions – these subscription requests reflect constraints in the INFOD registry
  - **INFOD registry** notifies publishers about Replaced/dropped subscriptions
- **Publishers** receive and process subscription request
- **Publishers** create and send messages to all relevant consumers using the WS-Notification message structure
- **Consumers** receive message using the consume operation

12 Appendix B – Accessing the INFOD Registry

The use of XQuery to access the information in the INFOD registry to described in this appendix. It is expected that providers of INFOD will provide tools automating steps and hiding details of the implementation from the INFOD users.

The first step is to describe publishers, consumers and subscriptions as seen in a specific context.

12.1 The Publisher View

Given a context, publishers are best described by an XML document called PublisherView – in a real application the name should reflect the context.

Here is a generic example – the boxes of the picture represent entities and property instances expressed with the named XML schema:

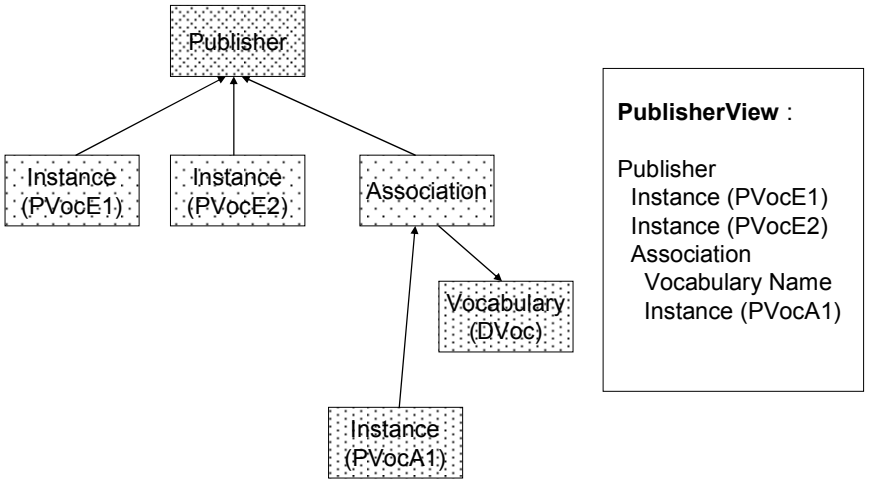


Figure 12-1: PublisherView

The PublisherView document can be created with the following XQuery statements:

```
For       $Pub in doc('MyINFODRegistry.xml')//Publishers,
$PVocE1 in doc('MyINFODRegistry.com.xml')//PVocE1,
$PVocE2 in doc('MyINFODRegistry.xml')//PVocE2,
$Asc in doc('MyINFODRegistry.asc')//Associations,
$DVoc in doc('MyINFODRegistry.xml')//Vocabularies,
$PVocA1 in doc('MyINFODRegistry.xml')//PVocA1

Where

$PVocE1/infod:VocabularyInstanceEntityReference = Fn:GetEPR($Pub) and
$PVocE2/infod:VocabularyInstanceEntityReference = Fn:GetEPR($Pub) and
$Asc/infod:AssociationEntityIdentifier = Fn:GetEPR($Pub) and
$Asc/infod: AssociationVocabularyIdentifier = Fn:GetEPR($DVoc) and
$PVocA1/infod:VocabularyInstanceEntityReference = Fn:GetEPR($Asc)

Return
```

```
<PublisherView>

{$Pub, $PVoc1, $PVoc2, $Asc, $DVoc//wsinfod.VocabularyName, $AVoc1}

</PublisherView>
```

12.2 The Consumer View

The same idea can be applied to the consumer; there is, however, no reference to associations.

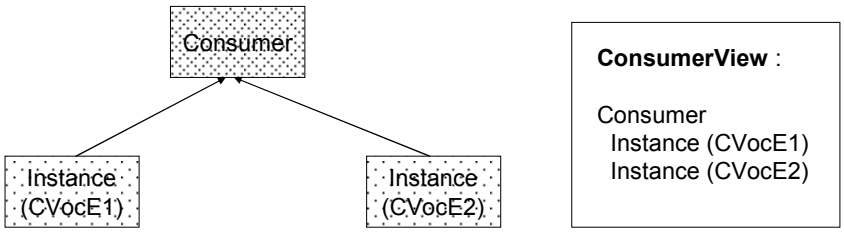


Figure 12-2: ConView

The XQuery statements for ConsumerView are constructed following the patterns of PublisherView. Please note that one arrow goes from instances of CVocE1 to optional instances of CVocE11.

```
For       $Con in doc('MyINFODRegistry.xml')//Consumers,
$CVocE1 in doc('MyINFODRegistry.xml')//CVocE1,
$CVocE2 in doc('MyINFODRegistry.xml')//CVocE2

Where

$CVocE1/infod:VocabularyInstanceEntityReference = Fn:GetEPR($Con) and
$CVocE2/infod:VocabularyInstanceEntityReference = Fn:GetEPR($Con) and

Return

<ConsumerView>

{$Con, $CVoc1, PVocE2,}

</ConsumerView>
```

12.3 The Subscription View

The next step is to include the subscription information. One could also discuss the subscriber view; but that view is generally not so important.

The same idea can be applied to the consumer; there is, however, no reference to associations.

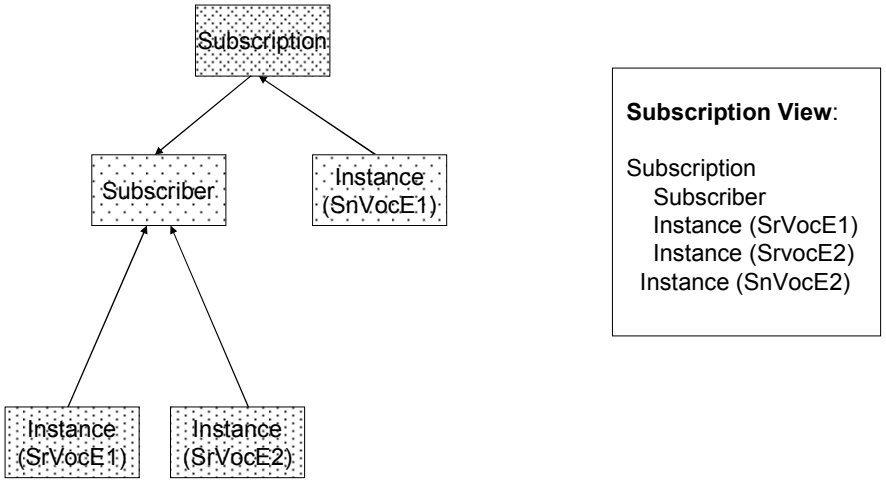


Figure 12-3: SubView

The XQuery statements for SubView are constructed following the patterns of PubView or ConView.

```
For      $Sub in doc('MyINFODRegistry.xml')//Subscriptions,
   $Subr in doc('MyINFODRegistry.xml')//Subscriber,
$SrVocE1 in doc('MyINFODRegistry.xml')//SrVocE1,
$SrVocE2 in doc('MyINFODRegistry.xml')//SrVocE2,
$SnVocE1 in doc('MyINFODRegistry.xml')//SnVocE1

Where

$Sub/infod:SubscriptionSubscriberIdentifyer = Fn:GetEPR($Subr) and
$SrVocE1/infod:VocabularyInstanceEntityReference = Fn:GetEP($Subr) and
$SrVocE2/infod:VocabularyInstanceEntityReference = Fn:GetEP($Subr) and
$SnVocE1/infod:VocabularyInstanceEntityReference = Fn:GetEP($Sub)

Return

<SubsriptionView>

{$Sub, $Subr, $SrVocE1, $ SrVocE2, SnVocE1}

</SubscriptionView>
```

12.4 The Consumer/Publisher View

The nest step is to create a special community to enable the interaction between publishers and consumers. This requires the inclusion of constraints.

Let us use assume the following (XPATH) constraints exist:

- PVocE1 referencing CVocE2 – limits publishers interest in consumers
- DVocA1 referencing CVocE1 – limits consumers access to data
- CVocE2 referencing PVocE1 – limits consumers interest in publishers

Applying these constraints would describe which consumer would be acceptable to which publisher.[4]

```
For $PView doc('MYINFODRegistry.xml')//PublisherView,
    $CView doc('MyINFODRegistry.xml')//ConsumerView,
Where
    fn:evaluate($PView/PVocE1/PropertyConstraint, $CView/CVocE2) = 1 and
    fn:evaluate($PView/DVocE1/PropertyConstraint, $CView/CVocE1) = 1 and
    fn:evaluate($PView/PVocE2/PropertyConstraint, $CView/CVocE1) = 1

return
    <PublisherConsumerView>
       {
        $PView/Publishers, $CView/Consumers
       }
    </PublisherConsumerView>
```

12.5 The Publisher/Consumer View

Changing the return clause would describe which publisher would be acceptable to which consumer

```
return

    <ConsumerPublisherView>
       {
        $CView/Consumers, $PView/Publishers
       }
    </ConsumerPublisherView>
```

Note that PublisherConsumerView and ConsumerPublisherView can be automatically generated once PublisherView and ConsumerView are generated.

12.6 Other Important Views

The following views/functions are of interest:

- **SubsriptionPublisherView** – publisher organized by subscription
- **SubsriptionConsumerView** – consumers organized by subscription
- **SubsriptionPublisherConsumerView** - consumers organized by subscription and publishers
- **SubsriptionConsumerPublisherView** – publishers organized by subscriptions and consumers

Note: Details will be added if reviews indicate a need.

[4] The evaluate function [EVAL} is one way to process multiple constraints as part of an XML statement.