



# A Data Sharing Middleware Prototype for Information Dissemination among Heterogeneous Data Sources

Technical Document - Volume I  
(June 2007- July 2009)

September 27 2009  
The University of Tennessee, Knoxville

# Purpose of the Report

This final project report is the final contract deliverable which summarizes the effort expended by the team in support of Department of Homeland Security-sponsored Southeast Region Research Initiative (SERRI) at the National Nuclear Security Administration's Y-12 National Security Complex through its partnership with the Department of Energy's Oak Ridge National Laboratory.

# Project Members

## Academia

- Hairong Qi (PI), University of Tennessee, hqi@utk.edu, 1508 Middle Dr., 319 Ferris Hall, EECS Department, Knoxville, TN 37996. Phone 865-974-8527
- Xiaorui Wang (Co-PI), University of Tennessee, xwang@ece.utk.edu
- Seddik Djouadi (Co-PI), University of Tennessee, djouadi@ece.utk.edu
- Raghul Gunasekaran (Research Associate), University of Tennessee, raghul@utk.edu
- Ming Chen, Ying Sun, Samir Sahyoun, Ben Taylor, UT Graduate Students

## Research Laboratories

- Frank DeNap, Oak Ridge National Laboratory, denapfa@ornl.gov
- Mallikarjun Shankar, Oak Ridge National Laboratory, shankarm@ornl.gov
- Steve Fisher, Rutherford Appleton Laboratory, UK, s.m.fisher@rl.ac.uk

## Industry, Private Sector

- Dieter Gawlick, Ronny Fehling, Aravind Yalamanchi, Oracle Corporation, dieter.gawlick, ronny.fehling, aravind.yalamanchi@oracle.com
- Vijay Dialani, Microsoft Research Center

# Acknowledgment

The team members would like to thank all our collaborators, Frank DeNap, Mallikarjun Shankar, Steve Fisher, Dieter Gawlick, Ronny Fehling, Aravind Yalamanchi, Vijay Dialani, for their persistent support throughout the project phase, without which, we would have never accomplished this much in such short period of time. We specifically would like to thank Dr. Mallikarjun Shankar from ORNL for his excellent mentorship, Dr. Dieter Gawlick from Oracle for sharing his vision on data dissemination and leading many insightful discussions over the weekly teleconference, and Mr. Aravind Yalamanchi from Oracle for providing world-class technical support to our development team.

We also would like to thank all the program managers, John Whittenburg, Romeo Morrissey, Mary Regan, Warren Edwards, and specifically, Ben Thomas, who have worked so closely with us and personally engaged in many rounds of discussions during the project phase. Thanks for your encouragement and strategic suggestions. They helped greatly in shaping the project toward the right direction.

Last but not the least, we thank Department of Homeland Security and the SERRI program for supporting our research and development effort such that we have the opportunity to contribute to the advance of science and technology.

# Contents

<b>1</b>	<b>Executive Summary</b>	<b>1</b>
1.1	Significance and Background . . . . .	1
1.2	A Unique Broad-based Alliance . . . . .	2
1.3	Brief Summary of Results . . . . .	3
1.3.1	Development of the INFOD Prototype . . . . .	3
1.3.2	Control-based Real-time Metadata Matching . . . . .	5
1.3.3	Source Localization, Boundary Tracking and Prediction of Chemical Plumes . . . . .	6
1.3.4	Dynamic Cyber-attack Detection and Classification (Cyber Defense)	8
1.4	Summary of Achievements . . . . .	9
1.4.1	Software and Demos . . . . .	9
1.4.2	Publications . . . . .	9
1.4.3	Students Advised and Graduated . . . . .	11
1.4.4	Collaboration with Other SERRI Teams . . . . .	11
<b>2</b>	<b>Motivation and Landscape Assessment</b>	<b>13</b>
2.1	Publish-Subscribe Models . . . . .	13
2.2	Literature Review . . . . .	15
2.3	Data Sharing Middleware . . . . .	18
<b>3</b>	<b>System Design</b>	<b>21</b>
3.1	The INFOD Model . . . . .	21
3.2	The INFOD Resources . . . . .	23

3.3	Constraints . . . . .	25
3.4	Mutual Filtering . . . . .	26
3.5	INFOD operations . . . . .	27
3.5.1	MetaData Queries . . . . .	27
3.5.2	Notification . . . . .	27
<b>4</b>	<b>System Implementation</b>	<b>29</b>
4.1	System Architecture . . . . .	30
4.2	Registry Structure . . . . .	31
4.3	Registry Operation . . . . .	33
4.4	Notification Messages . . . . .	36
4.5	Garbage Collection . . . . .	37
4.6	Mutual Filtering . . . . .	37
4.6.1	Constraints . . . . .	38
4.6.2	The Matching Procedure . . . . .	40
<b>5</b>	<b>System Evaluation</b>	<b>43</b>
5.1	Performance Metrics . . . . .	43
5.2	Evaluation Results . . . . .	44
<b>6</b>	<b>Use Case Description</b>	<b>48</b>
6.1	The First Responder Use Case . . . . .	48
6.2	LSS Use Case . . . . .	55
6.3	Extending the Use Case for DHS Applications . . . . .	57
<b>7</b>	<b>Conclusion</b>	<b>59</b>
7.1	Project Outcome . . . . .	59
7.2	Summary of Technical Outcome . . . . .	60
7.3	Discussions and Future Work . . . . .	61
7.3.1	Security Considerations . . . . .	62
7.3.2	Distributed Registry . . . . .	63

A	Appendix A	66
B	Appendix B	76

# List of Tables

5.1	Example of structural variation in XML - for property vocabulary instances	44
6.1	Property Vocabulary . . . . .	51
6.2	Data Vocabulary . . . . .	51
6.3	Property Constraints . . . . .	52
6.4	Subscription . . . . .	53

# List of Figures

1.1	Subscription Admission Controller . . . . .	6
1.2	Plume Modeling . . . . .	7
1.3	Cyber-Defense System . . . . .	8
2.1	Pub-sub models . . . . .	14
2.2	Pub-sub models . . . . .	18
3.1	The INFOD Registry . . . . .	23
3.2	Mutual Filtering in the INFOD registry . . . . .	26
4.1	INFOD System Architecture . . . . .	29
4.2	INFOD Table Structure . . . . .	31
4.3	INFOD Registry Structure . . . . .	32
4.4	Registry Operation flow Chart . . . . .	34
4.5	Resources to be compared in the Registry . . . . .	38
4.6	Evaluations for mutual filtering . . . . .	40
4.7	Mutual filtering in the INFOD registry . . . . .	41
5.1	System Scalability . . . . .	45
5.2	Effect of constraints on system performance . . . . .	46
5.3	Effect of schema complexity on system performance . . . . .	47
6.1	First Responder Use Case . . . . .	49
6.2	LSS Use Case . . . . .	55
6.3	LSS Application . . . . .	56

# Chapter 1

## Executive Summary

### 1.1 Significance and Background

The objective of this project is to develop a data sharing middleware that is able to handle multiple distributed data sources and dynamically changing items, and to assist in real-time information dissemination across multiple agencies for homeland security purposes. The increasing volume, diversity, and complexity of resources and data continue to raise challenges in sharing and retrieval of information. Getting the right information to the right person at the right time becomes the ultimate goal of the project. Although publish-subscribe systems have enabled communities to exchange information, we argue that these systems have been either restrictive or simplistic, relying on pre-defined channels for data sharing. The INFOrmation Dissemination (INFOD) model introduces a flexible and dynamic framework for brokering information among entities, including publishers who provide the information, consumers who are in need of information, and subscribers who are groups of consumers with common interests. These entities are characterized in terms of vocabularies, such as NIEM (National Information Exchange Model) and CAP (Common Alerting Protocol), and their interests as XQuery constraints within the INFOD registry. The registry matches publishers and consumers/subscribers dynamically based on their interests such that information can be routed from the publisher directly to the consumer. The matching mechanism is more effective in handling dynamic events and consumers would receive essential information even though they did not actively request it.

In a report published by DHS Science and Technology Directorate in June 2008, titled “High-Priority Technology Needs” [15], Information Sharing is identified as one of the twelve capstone IPTs (Integrated Product Teams) that provide capability to DHS operating components and first responders. We believe the objectives of INFOD are well aligned with DHS high priority tasks. We have demonstrated and evaluated the INFOD approach in a First Responder use case scenario which requires information dissemination based on changing event dynamics and varying requirements. Although the first phase development is oriented at building and evaluating the data-sharing prototype middleware, we anticipate that commercial systems will grow subsequently from this activity and emerge to meet the ultimate deployment needs of federal, state, county, and city entities and integrates into wide spectra of application domains, including fusion centers, emergency response system at various levels.

## **1.2 A Unique Broad-based Alliance**

The project represents a joint effort spanning academia, industry, and the federal sector. We are excited to have commercial partners Oracle Corporation, Microsoft Research, and the federal research laboratory Oak Ridge National Laboratory (ORNL) and Rutherford Appleton Laboratory (RAL) collaborate with the University of Tennessee in the creation of this product prototype. The specific complementary expertise and interests of the partners makes this team uniquely qualified to address the problems of data-sharing and dissemination. Specifically,

- University of Tennessee’s proposing faculty have in depth experience in related fields, including the development of collaborative information processing algorithms, the design of real-time embedded systems and adaptive middleware, and fault-tolerant control and multi-objective optimization.
- Oak Ridge National Laboratory has specific and unique experience in prototyping wide-area sensor networks at several sites (including Memphis, Charleston, Washington, DC). By being on several standards committees (including those sponsored by

the IEEE, Open Grid Forum, Open Geospatial Consortium, etc.), ORNL has been singularly positioned to work with academia and industry and has attracted the participation of commercial partners to solve nationally relevant problems.

- Oracle is acknowledged leader in the industry in producing robust, widely-adopted, and farseeing technology in the areas of data storage, data-mining, and information sharing.

Notable in this project is the in-kind support from Oracle and Microsoft. ORNL will be contributing in-kind (person hours and equipment) of about \$250K to received funding. Oracle and Microsoft will be contributing consulting hours and research software to the effort at no charge to the project. ORNL will also specifically use its understanding of deployed prototype systems to ensure that the research is focused on mission-oriented problems. The commercial partners will experiment and adopt the cutting edge developments from this research to potentially form their technology offerings aimed at adopters and buyers in the next five to ten years.

Over the course of the project period, a weekly teleconference sponsored by Oracle, has been held for team members to discuss progresses made, issues identified, and potential solutions. Through these weekly communications, team members are able to get problems resolved in a timely fashion, which greatly advanced the development work. Top-notch software engineers and high-rank architect from Oracle have provided real-time technical support and their vision to our project that has been invaluable to this work.

## **1.3 Brief Summary of Results**

This section highlights the research and developmental findings from project activities.

### **1.3.1 Development of the INFOD Prototype**

Compared to existing publish-subscribe systems that have improved the ability of communities to exchange information, the INFOD model largely advances the information sharing study by introducing a flexible and dynamic framework for brokering information between

entities. INFOD provides a structured information model where user communities are identified by property and data vocabularies. Property vocabularies characterize user entity properties or interests, while data vocabularies describe publisher data semantics or capabilities with available data. Data source entry associates publishers to multiple information sources. Publishers, consumers and subscribers are real-world entities characterized in terms of vocabularies and their interests as “constraints” within the INFOD registry. The Subscribers define subscriptions primarily as XQuery constraints, including which publisher sends information to which consumer, the event of interest at the publisher, and which message to send to which consumer. The entity description, the constraints, and the subscription comprise the metadata information that INFOD uses to associate and link entities within a community using the same vocabulary. We refer to the process of associating entities in INFOD as mutual filtering, which we realize with a three-way join across publishers, consumers, and subscriber entities. The INFOD registry matches publishers and consumers based on information needs expressed through subscriptions and limited by properties, after which it sends notification messages to user entities informing them of their associations. As user properties or subscriptions change, and/or new users/subscriptions are registered, the system reevaluates granting/limiting associations for information sharing.

A prototype of the INFOD model has been built. A simple first responder emergency alert use case with interactive web interfaces has been demonstrated in our recent SERRI review meeting. In this use case scenario, an accident has just occurred, and a bystander reports the event to the E911 center, which then requests for police, ambulance and/or fire truck to be dispatched based on the current description of the event. As the first officer arrives at the location, the officer reports more details of the accident to the E911 center and also updates on the states of the incident. If resources in a particular region are not sufficient, the E911 center needs to make a decision in calling for additional resources based on their capabilities and availabilities. In this example, INFOD helps 1) monitor resources, 2) associate services, 3) gain knowledge of the capability/availability of the resource, and 4) define the incident object, structure + policies/constraints which need to be enforced based on the event description.

In applications involving emergency responders or inter-agency data sharing across fusion centers it is critical that information be communicated to the right entity. With a number of policies/constraints to be enforced while sharing such critical information, the INFOD system provides the capability to efficiently communicate the right information to the right person at the right time. We are in discussion with emergency responders at UT and ORNL to understand their specific requirements and current state of the art systems being used.

### **1.3.2 Control-based Real-time Metadata Matching**

In time critical applications such as emergency alerting systems, making decision at the right time is as important as making the right decision. In many systems, data flowing from numerous sources to numerous sinks have to be channeled flexibly, efficiently and more importantly, in a real-time manner. This requirement has been generally described as Valuable Information at the Right Time (VIRT). Our work aims to explore effective algorithms based on control theory to guarantee the goal of VIRT.

A common problem faced by information dissemination systems running in dynamic environments is that the registered attributes and constraints may vary at runtime. For example, a firefighter may constantly update its location attribute. As a result, all subscriptions in the registry need to be continuously reevaluated by rerunning metadata matching to ensure that the firefighter receives information from the sensors in the right locations. However, given the large number of publishers and consumers, reevaluating all subscriptions in the registry may cause severe system overload and unacceptably long delays. Therefore, it is a common practice to guarantee that the average response time of reevaluating all high-priority subscriptions is within a real-time constraint, which is referred to as real-time metadata matching. In the meantime, the maximal possible number of low-priority subscriptions should also be reevaluated. The number of reevaluated low-priority subscriptions is defined as the quality of service (QoS) of the system. Real-time metadata matching faces two major challenges. First, execution time of reevaluating a subscription may vary significantly as the numbers of involved publishers, consumers and constraints are different for each subscription and are dynamically changing at runtime. Second, metadata updates

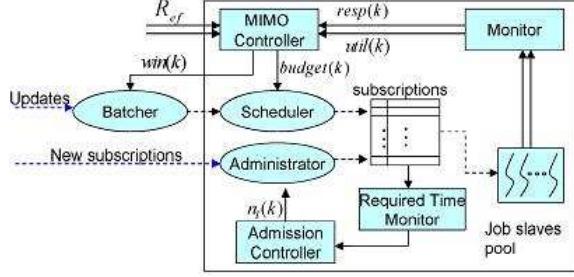


Figure 1.1: Subscription Admission Controller

may come either periodically or aperiodically with unpredictable time intervals. These uncertainties may lead to unpredictable delays for real-time metadata matching.

The INFOD model supports a control-based subscription matching mechanism for evaluating high priority policies within a time bound. We have explored novel feedback control architecture (Fig. 1.1) for INFOD to adaptively control the average response time of metadata matching. The system consists of two control loops, the integrated control loop that guarantees the metadata matching delay of high-priority information, as well as system throughputs, and the admission control loop that guarantees the metadata matching interval of low-priority information by applying some admission scheme on new arrival low-priority subscriptions. Further, we have proposed a Multiple-Input-Multiple-Output (MIMO) control algorithm based on optimal control theory to address the response time and the system throughput in an integrated manner. Empirical results show that our algorithm not only effectively guarantees the real-time requirement but also achieves the maximal system throughput.

### 1.3.3 Source Localization, Boundary Tracking and Prediction of Chemical Plumes

Plume localization and tracking is used as a potential use case scenario in this project. While this period of work focuses on the algorithm development and simulation, we hope to extend this work to testbed construction and field demonstration.

Accidental gas releases from industrial sites and biological terrorist attacks result in dangerous chemical plumes. To mitigate the effect of these plumes it is necessary to not only locate but also track their evolution, with the latter problem being more challenging.

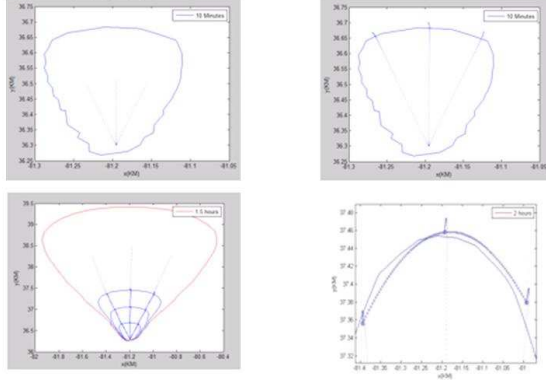


Figure 1.2: Illustration of mobile sensors tracking the plume boundary. Top-left: Sensors keep moving toward the boundary. Top-right: Sensors stop when the measurement is less than the boundary threshold concentration. Bottom-left: Sensors move to the predicted locations of the boundary. Bottom-right: Boundary interpolation.

We have proposed effective solutions to both problems using a group of fixed and mobile sensors, which measure the plume concentration at different locations and serve as both publishers and consumers within the INFOD framework.

For a grid of fixed sensors, the plume source is located using two methods, nonlinear Gauss-Newton least squares and Stochastic Approximation (SA) algorithms. It is shown that stochastic approximation methods give more accurate results than Least Square methods when dealing with noise corrupted data. An iterative algorithm that uses mobile sensors to estimate the plume source is also developed. It is shown that the sensors converge to the plume source after only a few iterations. The mobile-sensor-based iterative algorithm uses a smaller number of sensors than the first two methods but achieves the same accuracy.

Boundary tracking and prediction is accomplished using mobile sensors. A novel state space plume model is used in the estimating and prediction of the plume evolution. The predicted states of the plume progression are used as reference signals to deploy the sensors using optimal controllers. New measurements obtained from the sensors are then used to update the plume state estimates recursively at every time step. The estimates are further used in estimating the plume boundary thanks to spline interpolation. The process is repeated till the sensors converge to the plume boundary (Fig. 1.2).

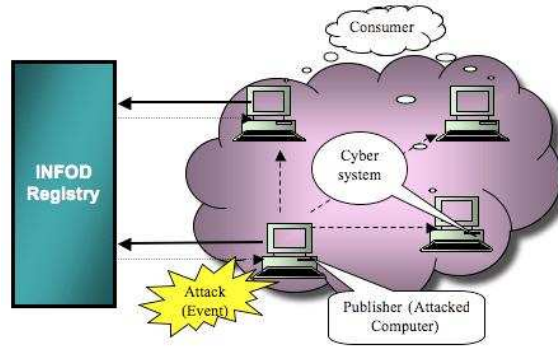


Figure 1.3: The cyber-defense system based on dynamic cyber-attack detection.

### 1.3.4 Dynamic Cyber-attack Detection and Classification (Cyber Defense)

Cyber-attack detection and classification is used as another use case scenario in this project. In this period of work, we have finished lab demonstration using data sets collected from multiple fixed sensors for target classification purpose.

The cyber system is composed of computers with different security levels, S1, S2, S3, and S4, with S4 having the highest security requirements, as shown in Fig. 1.3. For the intrusion type, currently our database can recognize four types of intrusion, probe, denial of service, remote-to-local, and user-to-root, which correspond to threat levels, T1, T2, T3, and T4, respectively, with T4 being the highest threat level. Computers with the highest security level are allergic to all types of intrusions, whereas computers with the lowest security level are only sensitive to intrusions with the highest threatening level. Dynamic classification is applied to each receiving packet from the network. Upon detection of intrusions of different threat level, only computers that are equipped with software that can handle these types of threats will be notified to save resources and confirm the detection. INFOD's main role here is to improve the quality of network monitoring with less resource consumed.

- If it is classified as a normal packet, no action will be taken;
- If classified as the T1 type intrusion, the attacked computer will send the alert information to all the other computers in the network which have the S4 security level;

- If classified as the T2 type intrusion, the attacked computer will send the alert information to all the other computers which have either the S3 or the S4 security level;
- Similarly, if it is classified as the T4 type intrusion, the attacked computer should send the alert information to all the other computers in the network which have S1 to S4 security levels.
- Each computer in the network behaves as both publisher and consumer, registering its security level in the INFOD registry using property constraints. Upon attack detection, the attacked computer (the publisher) will automatically notify computers (consumer) that are sensitive to this level of attacks. When one computer's security level changes, the INFOD registry will update its constraint, and mappings between publishers and consumers will be re-established.

## 1.4 Summary of Achievements

### 1.4.1 Software and Demos

With the support from a strong alliance of industry, research labs and academia partners, we are able to finish the development of the INFOD prototype one quarter ahead of schedule, which gives us more time to conduct thorough evaluations using the prototype. We have successfully developed a first responder use case scenario to show how the INFOD prototype works. All software and setup manuals have been posted at <http://panda.ece.utk.edu/wiki/InfoD> for public access.

### 1.4.2 Publications

#### **Papers Accepted and Presentations Made:**

- R. Gunasekaran, M. Shankar, D. Gawlick, S. Fisher, A. Yalamanchi, R. Fehling, H. Qi, "INFORMATION Dissemination middleware," Submitted to The Third ACM International Conference on Distributed Event-based Systems, Nashville, TN, July 6-9, 2009.

- M. Cheng, X. Wang, R. Gunasekaran, H. Qi, M. Shankar, “Control-based real-time metadata matching for information dissemination,” 14th IEEE Int. Conf. on Embedded and Real-Time Computing Sys and App, Taiwan, August 2008. (Acceptance rate: 26
- Y. Sun, H. Qi, “Dynamic target classification in wireless sensor networks,” Int. Conf. on Pattern Recognition (ICPR), Tampa, FL, December 8-11, 2008.
- S. Sahyoun, S. Djouadi, H. Qi, “Source localization using stochastic approximation and least squares methods,” 2nd Mediterranean Conference on Intelligent Systems and Automation (CISA’09).
- Raghul Gunasekaran, “INFOD Use Case Scenario and Demo,” Presentation at Open Grid Forum (OGF), Feb 2008, Boston
- Raghul Gunasekaran, “An INFOD Reference Implementation,” Presentation at Open Grid Forum (OGF), Oct 2007, Seattle

#### **Papers Under Review:**

- M. Chen, X. Wang, R. Gunasekaran, H. Qi, M. Shankar, “Adaptive response time control for metadata matching in information dissemination systems,” Submitted to IEEE Transactions on Computers.
- S. Sahyoun, S. Djouadi, H. Qi, “Dynamic plume tracking using mobile sensors,” Submitted to IEEE Conference on Decision and Control, 2009

#### **Student Poster Competition at the 3rd DHS University Network Summit**

The INFOD team has submitted three posters for the student poster competition held during the 3rd DHS University Network Summit. All three posters have been selected to enter the competition.

- R. Gunasekaran, M. Chen, S. Sahyoun, Y. Sun, B. Taylor, H. Qi, X. Wang, S. Djouadi, “Data Sharing Middleware for INFOrmation Dissemination (INFOD) among Heterogeneous Sources.”

- S. Sahyoun, S. Djouadi, H. Qi, “Source Localization & Boundary Tracking and Prediction of Chemical Plumes.”
- M. Chen, X. Wang, R. Gunasekaran, H. Qi, M. Shankar, “Control-based Real-time Metadata Matching for INFOrmation Dissemination (INFOD).”

### 1.4.3 Students Advised and Graduated

The UT team consists of three PIs (Hairong Qi, Xiaorui Wang, and Seddik Djouadi), one research associate (Raghul Gunasekaran), two Ph.D. students (Ming Chen and Samir Sahyoun), and two M.S. students (Ying Sun and Ben Taylor). We have graduated one MS student. Both Ph.D. students have passed the qualify exam and are actively working toward their dissertation research originated from this SERRI project.

- Y. Sun, Dynamic Target Classification in Wireless Sensor Networks, MS Thesis, Summer 2008.
- S. Sahyoun, Plume Source Localization and Boundary Tracking, *MS Thesis*, University of Tennessee, Fall 2008.

### 1.4.4 Collaboration with Other SERRI Teams

The INFOD project has kept close working relationship with other SERRI projects, including the Shelby County Sensor Fusion Center led by Dr. Hamilton Hunter at ORNL and the Information Sharing Policy Review Project led by Ms. Frances Bulter at Y-12 National Security Complex [16].

Dr. Qi has personally visited Shelby County Fusion Center, Memphis, with ORNL researchers to collect first-hand information on how INFOD can be possibly integrated within the fusion center. We learned that there are actually three fusion centers exist in geographically close proximity, the Shelby County Fusion Center, the Memphis Metropolitan Fusion Center, and the Tennessee Fusion Center located in Nashville. However, these three fusion centers are not sharing any information. In theory, fusion centers should welcome the exchange of data among different warehouses. It is not the case with existing fusion centers.

The lack of an efficient middleware to support information sharing and the lack of a policy to standardize information sharing both contribute to this situation.

INFOD provides the kind of middleware that would largely facilitate information sharing among different units. It also has flexible interfaces that are capable of seamlessly incorporating policies regarding security and privacy issues by the form of setting constraints to data.

## Chapter 2

# Motivation and Landscape

## Assessment

The increasing volume, diversity, and complexity of resources and data continues to raise challenges in sharing information. Suppose, we need to be able to channel the flow of data from thousands of sources to thousands of destinations dynamically. This is required in applications such as alerts from stock tickers, threat/monitoring sensors and actuators, or simply matching buyers and sellers in a retail application. We need a system that provides matching between the sources and destinations of streaming data in a flexible manner. The challenge is in handling these information sources or publishers which produce large volume of information and each individual publisher might express data in different semantics. Moreover, each consumer might have distinct information of interest which needs to be tracked and matched with the appropriate publisher. Apart from successfully matching publishers and consumers, the need is in sharing the right information with the right entity at the right time.

### 2.1 Publish-Subscribe Models

Communication through publish-subscribe (pub-sub) mechanisms has gained acceptance over the last decade as an effective mode for sharing of information. Historically originating in operating-system messaging mechanisms, pub-sub systems enable subscribers to

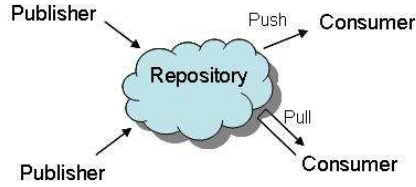


Figure 2.1: Traditional Publish/Subscribe model

express their interests for information (data, messages, etc.) and information is sent to the subscribers when it becomes available. Traditionally pub-sub systems, as illustrated in Fig. 6.3(a), consists of a central repository for publisher information. Subscribers, who are also the consumers of information, subscribe to information from the repository through a subscription. The subscription models the subscriber’s interest as a constraint. Matching the subscriber’s subscription to the information in the repository, the relevant information is delivered to the consumers. Information is received or retrieved by the consumer through *pull* or *push*. In the pull mode, the subscriber/consumer periodically or continuously queries the repository for information based on their needs. In the push mode, the subscription is stored at the repository and when the relevant information is available it is sent to the consumer. The modes of information dissemination catered to different application. The *pull* model served best for *service-oriented* applications and the *push* model served best for both *service-oriented* and *event-driven* applications.

The ubiquitous chat room is an example of a simple pub-sub system in which a user subscribes to a chat room and publishers post messages to that chat room. More sophisticated pub-sub system examples include stock-ticker and news report alert systems offered by web-based service providers. In these systems, a subscriber enters a range of data or type of story that interests them. The service provider polls the data source periodically, (re)publishing the data when it fits the subscription. The central repository in traditional models is a bottle neck for continuous information retrieval and since the information lookup is on the data, the task might be tedious and time consuming for large data volumes. Pub-sub systems in the literature generalize the enabling mechanisms and improve on the underlying infrastructure. The range of improvements focus on either the manner in which the subscriptions

are described and managed or on the infrastructure support to enable the messages to flow efficiently from source to sink.

## 2.2 Literature Review

The simplest pub-sub model in common use is the news group or mailing list, a *topic-based* pub-sub system. In this model the subscriber (also the consumer) subscribes for information on a topic of interest, in other words subscribes to a data channel. Email groups and news groups best describes a topic-based system. However, in these systems the subscriber has no option of limiting or applying constraints to the information being received through the data channel. This feature was supported by *type-based* pub-sub systems featuring subscriptions with simple constraints, where the subscriber can filter messages based on the structure or data type of an attribute based on the information flowing through the channel. RSS feed systems offer similar capabilities.

Further enhancement led to *content-based* pub-sub systems, where subscribers defined constraints on the attributes and content of the data to be received. The constraints are applied on the data or the meta-data of the information to be delivered. Content-based pub-sub models have contributed towards building systems for efficient sharing of resources in distributed or networked applications and as distributed computing paradigms. Message Oriented Middleware(MOM) architecture, a distributed computing paradigm, is based on the content-based pub-sub model. MOM is based on a queuing infrastructure, where consumers express their interests and publishers deliver information to a central message queue. A message transfer agent then delivers information to the consumers. MOM's asynchronous communication model maintains user anonymity, where publishers are unaware of their recipients and applicable constraints. JMS (Java Messaging Service) [8], provides a MOM API (Application Programming Interface) for the exchange of information between publishers and consumers. The JMS API provides two modes of information dissemination, a peer-to-peer model and a pub-sub model, and uses a simple language such as SQL to

define constraints. With asynchronous communication, scalability is not a problem, however, handling of queues by the message transfer agent determines the performance of the system.

A few other content-based pub-sub models address information dissemination in terms of a network/grid of information repositories with real time guarantees and optimized performance. **Gryphon** [4] is a content-based information dissemination model based on a redundant overlay network of brokers. Network brokers perform the function of matching, each broker partially matches events with subscriptions at each hop in the network and decides on forwarding to the next broker. The subscriptions in the network are organized in a hierarchical tree structure, exploiting the commonalities in multiple subscriptions. Matching is done from the top to bottom in the hierarchy of subscriptions and only one copy of the message is sent to the next broker. Consumers receive notification messages from the nearest broker. To avoid message hops in the network before it is dropped due to lack of relevance at bottom of the hierarchy, the match operation is done closer to the publisher end. Gryphon was implemented using standard JMS API and addressed security in terms of symmetric/asymmetric SSL authenticated connections.

**HERALD** [6]- an internet-scale event notification service. Herald system is an overlay network of servers (Herald servers) providing the pub-sub service and each server consists of multiple *rendezvous* points. The point acts as an abstraction of the pub-sub service, clients create these points, publish an event to the rendezvous point, subscriber's subscribe to the rendezvous point and the rendezvous point notifies the subscriber. A single herald server consists of multiple rendezvous points and each rendezvous point can hold multiple client information. The rendezvous points help accomplish a fault tolerant scalable system, points are replicated in other herald servers, points in a single server are distributed across multiple servers and also when the load in a rendezvous point increases the system splits work load among herald servers, resulting in an overlay distribution network. Herald provides a basic pub-sub model over a distributed network with support for simple queries.

**SIENA** is a wide-area event notification service. As described in [7], a distributed information dissemination model solves the problem of notification selection, i.e., which notification message should match with which subscription, and notification delivery, i.e.,

routing notification messages from publishers to consumers. The distributed pub-sub model is based on three design criteria: the interconnection topology, routing algorithm and the processing strategy. The model minimizes multiple notification messages sent to the network by looking at commonalities in the subscriptions. Thereby, when a subscription reaches a server, the subscription propagates only in the absence of similarities defined by previously propagated subscriptions. Attributes are defined in terms of tuples (*type, name, value*) and the constraints are defined as *type, name, operator and value*. This format of expressing information attributes supports the feature of extracting commonality in notification messages to be delivered, however achieving similar functionality using complex queries needs to be explored.

**Semantic Pub / Sub system**, SToPSS (Semantic Toronto Publish/Subscribe System) [5] addresses the problem of matching in pub-sub systems. Toronto pub-sub system, is a content-based model, matching publishers data to that of consumers interest. The model provides a solution for handling semantic problem in information dissemination systems, where most of the matching algorithms are based on the syntax of expressing constraints. Syntax defines the format, example if a constraint is a car of type sedan, syntactically it can be expressed as *car = "sedan"* or *car eq sedan* or *car is equal to sedan*. In a similar fashion the above constraint semantically needs to be expressed as *car eq sedan* or *automobile eq sedan* or *vehicle eq sedan*. The system translates all attributes with a single root attribute, which is done using a hierarchical structure of taxonomies of different words derived from ontology's. The semantic matching capability helps match multiple domains simultaneously, with a separate semantic stage followed by the matching stage. With generalized subscription the number of matches would be high, however with specialized subscription the matching process is fast with a minimal result set. The limitation of the system is in maintaining the taxonomy of the attributes.

Most of the above systems operate in linear fashion by evaluating constraints over publisher data, and focus on approaches to efficiently evaluate and reduce redundant constraints. As a key structural difference, the data sharing approach developed in this project proposes a framework on how individual user entities can express constraints and the mutual filtering process in evaluation of these constraints. Hence the constraints can be more appropriately

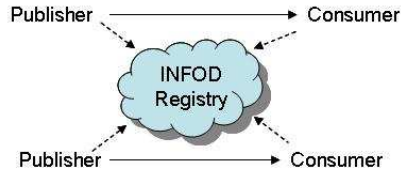


Figure 2.2: The INFOD model

represented at the data source, alleviating the central registry from data matching. The structured design also makes it possible for dynamically incorporating new users and events.

## 2.3 Data Sharing Middleware

There are two important structural shortcomings in existing systems that we address in our model. The first is that they typically restrict subscriptions to a fairly rigid set of options, e.g., a set of channels, or a pre-defined set of boolean operators on a limited set of entities, and the second is that pub-sub implementations almost inevitably concern themselves with the actual data contents and their relationship to the subscription - forcing the supporting infrastructure to be in the data path. In our model we describe a new approach to information sharing that allows senders and receivers to create their own vocabularies, create a flexible constraint structure, and have the system broker links between senders and receivers of information (letting them communicate directly). This approach called Information Dissemination (INFOD, pronounced *info-d*) has been developed and codified in a working group in the Open Grid Forum (OGF) - the complete API specification is available in [1]. INFOD addresses the gaps we find in current systems and supports a flexible pub-sub scheme that allows data sources and sinks to plug in and out of a system.

An important requirement INFOD addresses is to avoid having the infrastructure in the middle of the data-path because it raises scaling and security obstacles (impermissible in certain classes of systems). In contrast to traditional approaches, where a repository parses out the data from and to interested parties, described earlier, the INFOD approach allows the matched entities to communicate directly, as illustrated in Fig. 6.3(b). The model allows the plumbing of publishers and consumers. When the pub-sub system lies on the data-path the implicit assumption is that subscribers and consumers express interests

only against the constituent data and data-flows. In many cases, publishers and subscribers need to be matched based on their properties and the constraints (desirability requirements) they put on each other. These relationships depend on matching publishers and consumers independently of the real-time data. In such systems, there is a set of interests and capabilities that need to be matched to create the publisher-subscriber-consumer link. When event-based data becomes available at the publisher, the publisher further refines the target consumers. Thus, the constraints (or filters) and destination for the data itself is a run-time decision that is handed off to the publisher. INFOD reacts dynamically as the state of the entities and subscription changes. The registry reevaluates constraints breaking or forming new association between entities and notifies publishers of new sets of consumers it must communicate with.

Although a system with restricted subscription options is easy to use, an effective pub-sub infrastructure must support diverse interests and offer flexibility in terms of data and participant description. Users would like to be able to express constraints more powerfully on not just the data, but also on the metadata that describes the senders and receivers in the system. The INFOD infrastructure allows participants to register vocabularies, system entities (consumers, publishers, and subscribers), and subscriptions in a registry. The registry stores metadata characterizing this information in XML descriptors. The metadata may be added or deleted at any time and the registry accordingly updates the brokering status between publishers, subscribers, and consumers. INFOD allows subscription to define constraints associating publisher and consumer entities, define the event of interest at the publisher and also the messages that should be created in response to these events. The registry matches publishers and consumers based on the subscription, individual entities constraint and their metadata information. We refer to this process as mutual filtering. The registry then notifies entities of the match results. The publisher is notified of the consumers it is matched with and the subscription binding them.

INFOD may be distinguished from conventional pub-sub models in the following specific ways. First, *consumers* and *subscribers* are characterized as independent entities. In the model a subscriber is not always a consumer of information, rather it creates a subscription for a community of consumers. (We do not use the symmetric term producer because the

true relationship of the message sending action is from publisher to consumer. We use the term data source instead to refer to sources of information.) Second, communities of interest create and themselves define vocabularies for characterizing and expressing interests. Third, the middleware defines a constraint based mutual filtering technique operating on entity properties described in terms of the vocabularies as opposed to solely content-based matching found in conventional pub-sub models. Fourth, the model allows entities to define their own conditions of interest (constraints on whom they may communicate with) as distinct from subscriptions that ultimately connect entities within the community. Fifth, the INFOD model supports the handling of dynamic events filtered at the publisher, and allows publishers to determine valid consumers based on changing event states - the data path is, thus, directly from publishers to consumers.

The remainder of the report is organized as follows. Chapter 3 introduces and describes the INFOD middleware in detail. Chapter 4 explains the implementation and software architecture of the data sharing middleware. It also describes the mutual filtering procedure, which forms the core of the system. This is followed by chapter 5 discussing the system performance. Chapter 6 describes the use cases built for testing the INFOD and potential use-case where INFOD can be used. Chapter 7 concludes the report.

## Chapter 3

# System Design

The INFOD middleware is a repository accessible through SOAP based web services and capable of notifying entities through *Web Service* Notification. The middleware provides a general means to determine which messages are to be sent from which publishers to which consumers based upon information kept in the INFOD registry. The approach extends the traditional pub-sub paradigm by allowing consumers to be determined *dynamically* based on the message content.

### 3.1 The INFOD Model

The model captures users entities as *publishers*, *data sources*, *consumers* and *subscribers* in the INFOD registry. The user entities create and describe themselves in terms of vocabularies - *property vocabulary* and *data vocabulary*. A vocabulary can be thought of as a language template used by a community with common interest. The property vocabulary is used to describe or characterize the user entity and the data vocabulary describes the source data. In the INFOD model, the publisher represents a source of information associated with a data vocabulary. Publishers can also represent multiple or heterogeneous sources of information modeled as data sources, with each data source distinctively describing the source information in terms of a data vocabulary. The data source can also be viewed as a low-end publisher. A publisher which is not capable of communicating with the INFOD registry can be represented as a data source entry in the registry. For example, in a sensor network use

case, multiple sensors would communicate to a single base node, and each sensor could be unique and deliver information in different formats. In this use case, the base node is the publisher and the sensors are the data source entries in the registry. The model allows data source entries to create property vocabulary instances and each entry could be associated with a different data vocabulary.

The model differentiates between a consumer and subscriber. A consumer is simply a receiver of information whereas a subscriber creates subscriptions for a community of consumers. All entities are described as property vocabulary instances (XML documents that agree with a property vocabulary schema) and also express their interest for interaction with other entities as *constraints*. User entities express constraints not on the data, but the metadata that describes the senders and receivers in the system. Only the subscriber creates subscriptions that define constraints on the source data identifying the event of interest and the information to be disseminated in response to these events.

With all entity information captured in the registry, as shown in Fig. 3.1, the registry needs to identify which publisher and which consumer needs to be matched. The registry uses subscriptions, individual entities constraints and metadata information to identify the publisher-subscription-consumer event channel. The registry then notifies entities of the match results. Matching in the INFOD registry is strictly based on entity properties and constraints they put on each other. The constraints on the run-time event data is handled by the publisher. When event-based data becomes available at the publisher, the publisher further refines the target consumers and sends the appropriate message. The INFOD registry reacts dynamically if the state of an entity or the subscription changes. The registry re-evaluates constraints breaking or forming new association between entities and notifies publishers of the new set of consumers.

Information in the INFOD registry is represented as resources and each resource is uniquely identified by a URI (Uniform Resource Identifier). All resources in the registry are represented as XML documents associated to a schema. The following subsection details the various resources in the registry.

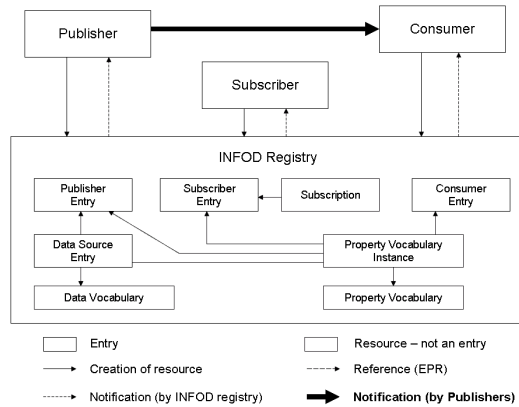


Figure 3.1: The INFOD Registry

### 3.2 The INFOD Resources

**Vocabulary**, facilitates a structured model for information exchange and the semantics for expressing constraints. The INFOD model supports two forms of vocabulary definitions, *property vocabulary* and *data vocabulary*. Property vocabulary defines the semantics for characterizing entities and expressing constraints on entities to be matched with in the registry. Data vocabulary defines the semantics of publisher data, for describing an event of interest and the message in response to the event. Defining vocabularies has its advantages for users within a community to express interest and simplify the process of matching. Many pub/sub models, described in [5], use ontology’s and a database of taxonomies solving the problem of semantic matching between publishers and consumer in the absence of a predefined semantics. However, INFOD vocabulary definitions significantly reduce the complexity in the process of matching and provide a more definitive match among entities.

Apart from the semantic benefits, vocabulary definitions provide a flexible and structured means of communication within a community. Vocabularies can be adopted from established standard such as NIEM (National Information Exchange Model) or could be custom XML schema definition (XSD) agreed by the community. Various interest groups can establish custom vocabularies in the registry for selective interaction or accept established vocabularies in registry. Vocabularies implicitly bind communities of common interest, which also helps a single entity to be a part of multiple communities by accepting those vocabularies and creating property vocabulary instances.

**Entry**, represents an entity as a publisher, consumer, subscriber, or data source in the registry, each being identified by a unique URI. An entry in the registry constitutes a *Web Service Address (WSA)*, name, description, property constraint and a boolean notification value. The WSA is the endpoint reference of the entity for receiving notification messages from the registry and other entities, while the name and description are for user comprehension. The boolean notification value indicates if the entity requires to be notified by the registry on the results of mutual filtering. A value of TRUE implies the entity is interested in messages from the registry. Property constraints expressed in entries model greater flexibility in matching entities of interest, which in generic pub-sub models are always a part of subscription. For example, a publisher entry in its property constraint can define specific consumers of interest, or also select subscribers of interest allowing or limiting the number of applicable subscriptions. Though subscriptions necessarily associate publishers to consumers, the property constraint defines an individual entities interest over the subscription associating them.

**Property Vocabulary Instance**, describes a publisher, consumer, subscriber, or data source entry in terms of the registered property vocabulary in the registry. An instance is an INFOD resource which references an entry and a property vocabulary in the registry by their URI's. The instance characterizes or describes the referenced entry in terms of the referenced vocabulary. An entity can have more than one instance and can be from different property vocabularies. This allows an entity to be a part of multiple communities.

**Subscription** is created by the subscriber. The subscription associates entities modeling constraints which identifying entities for interaction from the metadata information in the registry, defines the event of interest at the publisher, and the messages to be generated in response to the event for consumer. The constraints are modeled in the subscription as three types of constraints: property constraint, data constraint, and dynamic consumer constraint, which are detailed in the following section.

### 3.3 Constraints

Entities define constraints expressing conditions of interest thereby both granting and limiting the flow of information from publishers to consumers. INFOD model supports three types of constraints: *property constraints*, which is the basis of matching in the INFOD registry, *data constraints*, defines an event at the publisher and message that needs to be generated in response to the event by the publisher, and *dynamic consumer constraints*, determines a subset of consumers to be notified from the set of consumers identified by the registry based on specific consumer properties.

***Property Constraint***, are expressed by publishers, consumers, subscribers and data sources in their entry's, and in the subscription. Constraints by a specific entity represent the entity's interest in interacting with other entities in the registry. The constraints are expressed as an XPath or XQuery expressions and are evaluated as boolean expressions on resources in the registry. The constraint attributes are based on registered property vocabulary schemas and are only evaluated in the INFOD registry. Publishers/data sources express consumers and subscribers of interest. Subscribers define constraints on consumers, publishers, and data sources, while, consumers specify constraints on subscribers, publishers and data sources of interest.

***Data Constraint***, defines an event of interest at the publisher and the message that needs to be generated in response to an event. The constraint is defined as a part of the subscription and a publisher is notified of the data constraint. Constraints can be defined in any format, such as XPath, XQuery, it is in the capability of the publisher to evaluate the constraint. Data constraints should be expressed in terms of the registered data vocabulary attributes.

***Dynamic Consumer Constraint***, defines a constraint relating events to consumer and help the publisher identify consumers from a static set of consumers determined through the matching process in the registry. The dynamic consumer constraint can also be defined in any format, such as XPath, XQuery, it is in the capability of the publisher to evaluate the constraint. The constraint is expressed in terms of the registered property vocabulary attributes.

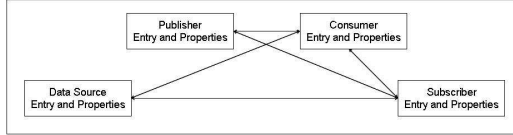


Figure 3.2: Mutual Filtering in the INFOD registry

### 3.4 Mutual Filtering

Mutual filtering refers to the process of matching entities based on the property constraint and entity’s property information in the registry. In general, constraints in pub-sub models are defined only in the subscription by the consumer/subscriber and are evaluated on the data or meta-data of the publisher. The INFOD model offers a flexible way for associating entities by allowing every entity to define a constraint specifying with whom it would interact with. Though the subscription constraint groups publishers and consumers, mutual mapping between entities is based on individual entity’s constraint. Allowing every entity to define constraints increases the complexity in matching but provides a greater level of flexibility in comparison to traditional pub-sub models. In general, subscription is the only means of expressing interests, however, in the INFOD model every entity is allowed to represent individual interests, thereby reducing the number of subscriptions.

A subscription triggers matching in the registry, however, only the property constraint of the subscription is used for matching entries in the registry. The data constraints and the dynamic consumer constraints are evaluated by the publisher, which determines the notification message the publisher sends to a specific consumer. The subscription’s property constraint associates publishers and consumers in the registry based on the property constraint. An entity with no property constraint is simply connected to other entities by the subscription’s property constraint. A subscription with no property constraint associates all publishers and consumers in the registry. Fig. 3.2 illustrates the mutual filtering operation across various entries in the registry which have constraints and property vocabulary instances. Also, any state change in the registry triggers the process of mutual filtering, by state change any create, replace or drop operation could trigger evaluation of constraints. A create or replace operation should result in the evaluation of relevant constraints in the registry, which might or might not change the mapping, nevertheless would need to be

checked. However, a drop operation does not necessarily trigger the matching process and depends on the implementation, if the registry is capable of storing match results any drop operation would result in sending new notification messages to previously mapped entities by ignoring references to the dropped resource.

## 3.5 INFOD operations

### 3.5.1 MetaData Queries

The registry is a repository for vocabulary definitions, entity information, property vocabulary instances and the subscription, and allows users to query for this information in the registry. For example, a user entity interested in joining a community would need access to the vocabulary definitions and would perform a metadata operation for the vocabulary definition. Entities can perform two operations, a *GetMetaData* operation with a query input and a *GetNotificationMessage* operation specifying a subscription URI. In the first operation, the query is formulated as an XPATH or XQuery expression referencing the information in the registry. For the second operation, the specific subscription would be evaluated and the corresponding notification message sent based on the requested entity being a publisher, consumer, or subscriber.

### 3.5.2 Notification

The INFOD registry sends notification message to publishers, consumers and subscribers as a result of mutual filtering and state change in the registry. The publisher then notifies consumers of the actual information. Publishers, consumers and subscribers are modeled as web services which are capable of receiving Web Service - Notification (WS-Notification) messages from the registry. Also publishers should be capable of sending notification message to the consumers. A publisher is notified of potential consumers bound by the particular subscription along with the data constraints and dynamic consumer constraints. Similarly the consumer is notified of the subscription URI and all publishers bound by the subscription. Also, the subscriber is notified of the subscription URI with the list of consumers and

publishers bound to the specific subscription. The registry notification messages are conditional based on the choice to receive notification messages or not, set by the entities while creating an entry in the registry. The publisher having received the notification message from the registry, sends notification message to consumers based on the event definition in the data constraints and specifically to certain consumers, as identified in the dynamic consumer constraint. The notification message sent by the registry and the publisher is in accordance with WS-Notification specification [10].

As stated earlier, all the information in the INFOD registry are stored as XML documents based on defined XML schemas. The various XML schemas which capture the information in the registry are listed in Appendix 1.

## Chapter 4

# System Implementation

The current implementation of INFOD is in Oracle Database 10g. However, it does not restrict the implementation of INFOD to a specific database. Our choice of INFOD implementation in oracle database was to benefit from its capabilities as an XML database and a relational database model, and also for some of the built-in capabilities such as expression filters for evaluating constraints.

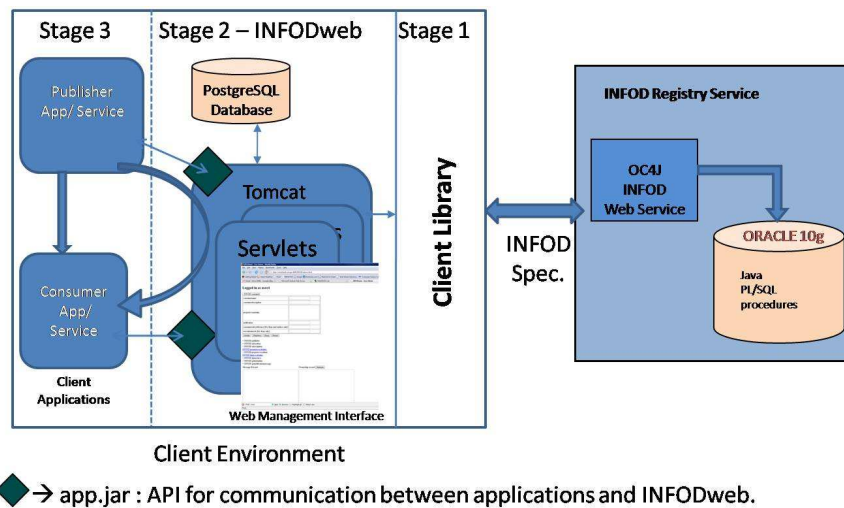


Figure 4.1: INFOD System Architecture

## 4.1 System Architecture

The system is developed as a client-server model which would enable testing, evaluating and building applications. Fig. 4.1 pictures the system architecture. The server being the INFOD registry accessible as a web server through SOAP requests and the client setup are built on Apache Tomcat. The server side system consists of the Oracle database and Oracle Application Server (OC4J - Oracle Containers for J2EE). All INFOD interfaces are available as web services capable of being communicated using SOAP messages, as defined in the INFOD specification. The Oracle Application Server OC4J is used to host the INFOD web services which connects and communicates with the database. The Oracle JDeveloper tool is used to generate INFOD web services from PL/SQL functions in the database and host it on OC4J. The PL/SQL functions define the various INFOD interfaces performing database operations (insert, update, delete) and triggering actions (matching, notification) in the database.

The client module is developed to interact with the INFOD registry. It contains three stages of implementation, as shown in the Fig. 4.1. The first stage is the client library coded in java for interaction with the INFOD registry service. The library defines all the operations as function calls that the user can use for interacting with the registry. The function call supports all interfaces that are defined in the INFOD specification document. Application developers who would like to interact with the INFOD system can use this software package for developing custom applications interacting with registry.

The second stage is a web interface for accessing the registry. Users can use this web interface to create, replace, and drop resources in the registry. First time users are allowed to create an account signing up for a username and password. The returning users would use the login information to view the resources the user created and receives all the notification messages when the user is not logged in. This interface helps users build applications and understand how the INFOD registry associates users based on the resources they have created in the registry. The user information and associated resource URI's are stored in a PostgreSQL database at the client side.

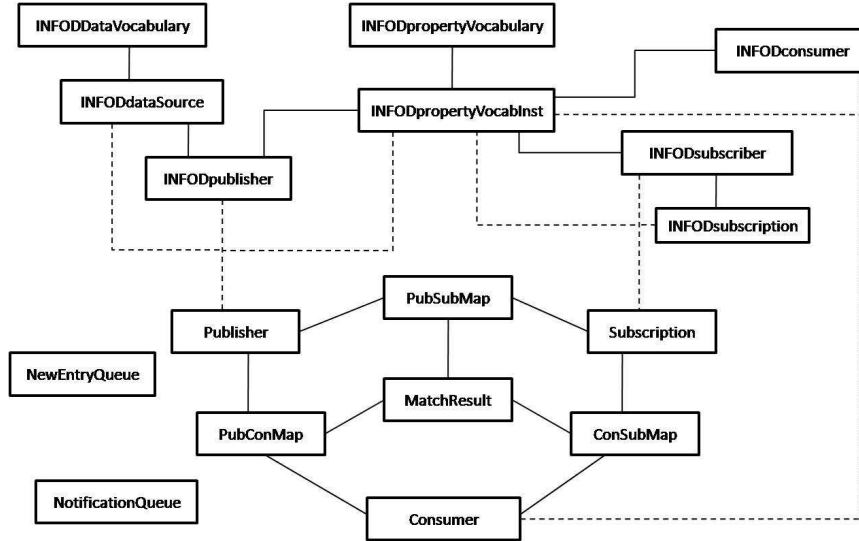


Figure 4.2: INFOD Table Structure

The third stage is for binding user applications or modules to the INFOD system. A standard set of Java interfaces have been identified which a user application can call for communicating with the existing client setup. The user, through the web interface, uploads a jar package while creating an entry in the registry through the web interface. The jar file for publishers could help forward the notification messages to the user applications and any notification messages generated by the application can be directed using the existing client module. Similarly, the jar file for subscribers and consumers helps direct notification messages to their respective applications.

## 4.2 Registry Structure

Fig. 4.2 illustrates the INFOD model in the database, including the tables and their structures. The tables prefixed with ‘INFOD’ store information on entities registered through the INFOD interfaces as XML documents - as defined in the specification document. The other tables in the database store information to enable efficient matching and notification of entities in the registry. Fig. 4.3 details on the tables prefixed with ‘INFOD’, while the remaining tables are described in the mutual filtering section. The table *publisher* in Fig. 4.2 is linked to tables *INFODpublisher*, *INFODdatasource* and *INFODpropVocabInst*. The table *publisher* is used for matching, where a publisher is represented with information

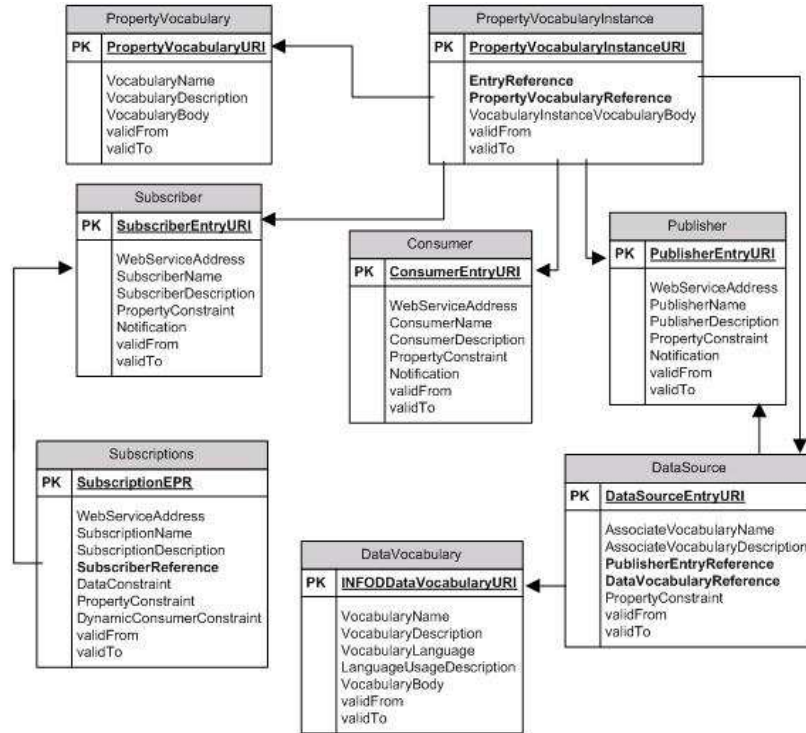


Figure 4.3: INFOD Registry Structure

from its entry and the referenced property vocabulary instance. If a single publisher entry has multiple property vocabulary instances then the publisher table would have multiple entries with each one referring to different instance. All these entries would have the same property constraints since they refer to a single publisher entry. For a data source entry, it is identified by a separate entry in the publisher table associated to its corresponding property vocabulary instance. The data source entry's property constraint would also include the property constraint from the publisher being references in addition to its property constraint. Similarly the table *consumer* combines the consumer entry and property vocabulary instance as a single entity to enable matching. And the table *textitsubscription* associates the subscriber's entry, subscription defined by the subscriber and the property vocabulary has a single entry. The benefits of associating data in this model would be appreciated when detailing on the mutual filtering section.

Fig. 4.3 details on the different tables and the contents of the table. The name for each table is indicated in the grey background. Each table in the figure has four columns; the first column is the primary key element, the validFrom and validTo fields are the third

and fourth columns. The rest of the elements in the corresponding tables are in a single XMLType column referring to an XML schema. For example, the consumer web service address (WSA), name, description, propertyconstraint and notification details are stored as an XML doc in an XMLType column. In the current implementation, this XML message is the body of the SOAP request messages from a consumer, when a user creates an entry in the INFOD registry. The primary key is a unique URI identifying the information in the registry. The validFrom and validTo are used for drop operations in the registry. When a new entry is created the validFrom field is set to the current time and when any drop operation is requested the validTo field is set.

The tables NewEntryQueue and NotificationQueue are formed from the queue tables handled by user defined processes. Every new entry in the registry is added to the NewEntryQueue for the job handler to match it with other entities in the registry, once the mutual filtering process is done the entry is removed from the NewEntryQueue table. After performing the mutual filtering, matched entities need to be notified of the results through web service notification. The entity to be notified with the relevant subscription causing the notification is logged in the NotificationQueue table. A user scheduled job handler reads from the notification queue table and sends a web service notification message based on the entities role as a publisher, consumer or subscriber.

### 4.3 Registry Operation

The registry supports three kinds of operation: create, replace and drop. The replace operation is restricted to only certain entity information in the registry. Fig. 4.4 illustrates a typical operational sequence for a consumer entry being created, replaced, and dropped.

When a new entry is created a unique URI (Universal Resource Identifier) is generated, which uniquely identifies every resource in the registry. When a consumer entry is created it triggers a java procedure call for reorganizing the information and parsing the constraints enabling efficient matching within the registry. As we recollect, the consumer can define constraints on publishers and subscribers of interest. In this case, the consumer's property constraint is parsed into two constraints one that is applicable on the publisher metadata

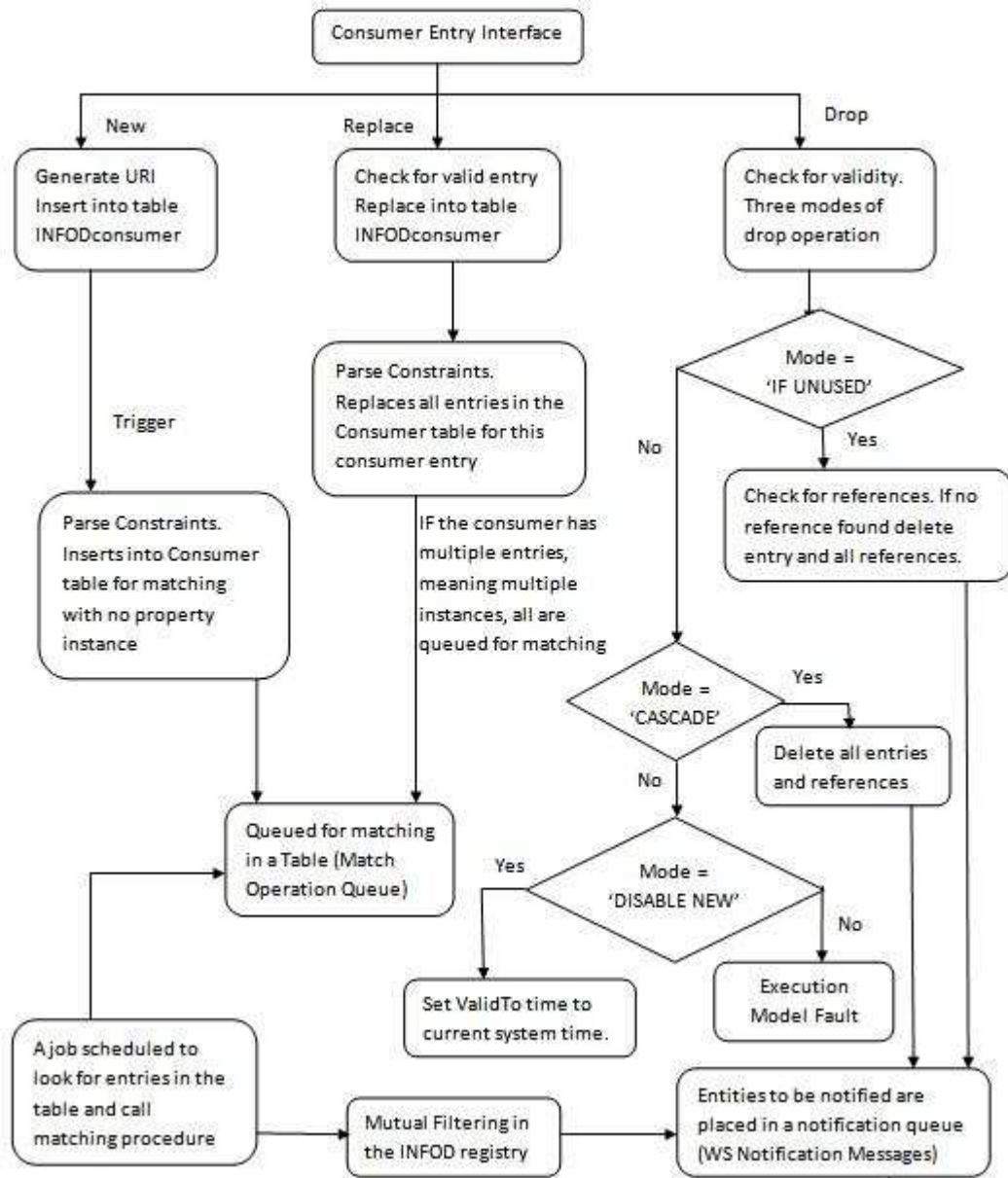


Figure 4.4: Registry Operation flow Chart

and the other constraint that is applicable on subscriber metadata. The constraints needs to be parsed such that they can be evaluated using expression filters. If the consumer does not have constraint on a particular entity, say the subscriber, the constraint is represented as  $1=1$ , which in expression filter evaluation method would be matched with all entities. Apart from the constraints other information in the entry is also extracted from the XML doc and are represented in columns of `varchar` type for easy retrieval. The information of the new entry created is updated in the `NewEntryQueue` table for mutual filtering in the registry. A job handler repeatedly checks for new entries in the `NewEntryQueue` table and performs the match operation, detailed in the next chapter. Once the consumer is matched with publishers and subscribers in the registry a notification message is sent to all the matched entities. The entities to be notified and the notification message that needs to be sent are queued up in the `NotificationQueue` table. Another job handler checks this table periodically and sends the web service notification messages. The need for having a separate table for notification is to avoid network delays inhibiting procedure runs in the registry.

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

The drop operation does not call for mutual filtering but entities needs to be notified when an association is no longer valid. Since, the results of mutual filtering is stored in the registry identifying which entity is no longer associated with the entry being dropped is a lookup on the results table. The registry first checks if the entity to be dropped is present in the system. The registry supports three kinds of drop operations,

- *IF UNUSED* mode, used when the user wants to drop an entry only when it is not associated with any other resource in the registry.
- *Disable New*, allows the user to restrict the entity from forming new associations with other entities in the system. This is done by updating the `validTo` field associated with the entities, which is checked whenever an entity is being matched.
- *Cascade* mode, deletes the entry and all the associations the system has in the registry.

## 4.4 Notification Messages

Publishers, consumers and subscribers are notified by the INFOD registry. The publisher is notified of the subscription and the consumers bound to that subscription. Similarly, the consumer is notified of the subscriptions and publishers, and the subscriber of the consumers and publishers bound by a specific subscription. The INFOD specification document details on the format of the notification messages.

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

The web service notification message to be sent to entities is created in a pl/sql procedure using DOM parser. The process of sending notification messages is handled by a java procedure called within the pl/sql procedure. As described earlier, the entities to receive the notification messages are queued in a NotificationQueue table and are handled individually

by a job handler. This is to avoid the delays in sending the notification message affecting the completion of the mutual filtering process and performing other registry functions. From the above example a change in C1's constraint results in sending a number of notification messages, which might be time consuming depending on the network connectivity.

## 4.5 Garbage Collection

This procedure is for removing resources in the registry which have been dropped by 'DISABLE NEW' execution mode. This a procedure which is run at periodic intervals checking for resources which are no longer associated with any other resources in the registry. The procedure looks for entries which are have a validTo field time stamped and looks for any references associated to the resource. Example, a consumer could be referenced in the property vocabulary instance and matched with resources in the registry. When a consumer is void of either references then the consumer entry would be removed through this procedure. The procedure scans for resources in the registry with validTo field time stamped and has no references in the registry, and deletes such resources in the registry.

## 4.6 Mutual Filtering

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

	<b>Publisher</b>	<b>Consumer</b>	<b>Subscriber</b>	<b>DataSource</b>
<b>Publisher</b>	X	✓	✓	X
<b>Consumer</b>	✓	X	✓	✓
<b>Subscriber/Subscription</b>	✓	✓	X	✓
<b>DataSource</b>	X	✓	✓	X

Figure 4.5: Resources to be compared in the Registry

#### 4.6.1 Constraints

The INFOD model allows publishers, data sources, consumers and subscribers to define constraints (property constraints), as a condition for triggering information flow between publishers and consumers. Publishers, data sources and consumers define constraints while creating an entry in the INFOD registry. Also, subscribers define constraints while creating an entry in the INFOD registry and also define subscriptions. The constraints specified by an entity are applicable on other entries and/or the property vocabulary instances of an entry. For example, a publisher could specify constraints on a consumer entry, consumer property vocabulary instance, a subscriber entry and a subscriber property vocabulary instance.

Fig. 4.5 details which entity can define constraint on which other entity in the INFOD registry and accordingly the matching procedure should enforce all these possible comparisons. The row headers indicate the constraints defined by the entities and the column headers indicate the instances of the entities in the registry. A publisher instance means the publisher entry, publisher property vocabulary instance and publisher data source entries. When a consumer defines a constraint it would be applicable on publishers, subscribers and data source instances in the registry.

In the following, we list certain functionalities of INFOD to be considered while matching

- A subscription binds publishers/data sources and consumers - meaning in the absence of a subscription, publishers/data sources and consumers are NOT associated for information flow, though they would express entities of interest as constraints.
- Any create/replace/drop in the INFOD registry would trigger the matching procedure and would require evaluation of all relevant constraints in the registry. Except for cases where it can be logically predetermined that any particular create/replace/drop operation would not require evaluation of a group of entries or constraints in the registry. For example, through prior evaluation or matching process it has been predetermined that a specific publisher and subscriber cannot be matched then any subscription created by the subscriber need not be evaluated against a set of publishers. However, if any of the publisher or subscriber properties change then all subscriptions need to be evaluated.
- ALL constraints relating a publisher and a consumer need to be satisfied for notification. Meaning if subscription relates a publisher and consumer; and the consumer satisfies the publisher's constraint but the publisher does not satisfy the consumer's constraint then publisher and consumer are not notified of each other.
- If a subscription does not specify any property constraint then all publishers and consumers in the registry are related through the subscription and they would be associated only based on their individual property constraints.
- A data source entry inherits all the property constraints of the publisher it references. For a data source to be matched with a consumer, the data source entry's property constraint and the referenced publisher's property constraint should be satisfied.
- Similarly, subscription inherits all the property constraints of the subscriber that created the particular subscription.

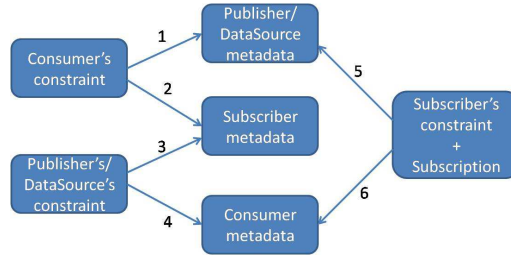


Figure 4.6: Evaluations for mutual filtering

#### 4.6.2 The Matching Procedure

Fig. 4.6 illustrates the computational flow for the mutual filtering. We consider the three metadata sets describing the publisher/datasource, consumer, and subscriber. We must constrain the entities with the three set of property constraints by the publisher/datasource, consumer, and subscriber/subscription. For a publisher and consumer to be matched, all the six computations must be evaluated. We use the Oracle database’s capability to evaluate XPath expressions using expression filters [3] and its indexing methods. The *evaluate* operator in expression filters evaluates XPath expressions on XML type data stored in the database tables.

The mutual filtering operation can be performed in a single SQL *select* statement by specifying all the six computations in the where clause. However, we found the scalability of such a system limiting. With 200 consumer and 200 subscriptions in the system, irrespective of the number of publishers, when a new publisher joins the registry, the system took nearly a minute to associate the publisher with consumer’s in the system. This is the time taken for the six computations to be done when the entries in the system have a instance schema of 500 elements. For a system setup with 1000 consumers and 1000 subscriptions the system took over four minutes to complete all computations. The non-linear performance degradation occurs because of the computations - cross product of the number of entities in the system. We optimized the mutual filtering process by creating intermediate steps - reducing the number of computations.

The optimization is based on the following observations

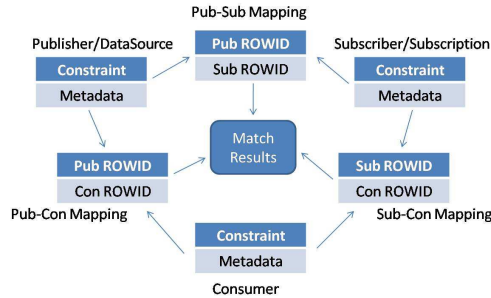


Figure 4.7: Mutual filtering in the INFOD registry

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

Fig. 4.7 illustrates the tables used in the database with the mapping tables representing the intermediate stage. The publisher and consumer are matched on their constraint and metadata, and the results are stored in the pub-con mapping table. Similarly, the sub-con mapping table has the list of the subscription and consumer pairs that can be associated and the pub-sub mapping table has the publisher and subscription pairs. With the intermediate steps, the mutual filtering function is a three step process. Say, a new publisher joins the INFOD system. The first step is in finding all the consumer's the new publisher can be matched with. For this we evaluate the new publisher's constraint on all consumer's metadata and all the consumer's constraint are evaluated on the new publisher's metadata. The result is the new publisher being matched with consumers and the results are updated in the pub-con mapping table. Similarly, the second step is in finding all the

subscriber/subscription the new publisher can be mapped with and updating the pub-sub mapping table. The final step is a three-way join across the mapping tables for determining the subscription-publisher-consumer tuples. This tuple gives us the desired information on which publisher and which consumer are being associated and through which subscription. However, the final step depends on the number of entries existing in the system, which determine the number of rows in the mapping table and correspondingly affect the computation time.

## Chapter 5

# System Evaluation

The INFOD registry is implemented on Oracle database 10g and the SOAP web service for interacting with the registry is hosted on OC4J (Oracle Containers for J2EE) application server. The implementation was carried out on a Linux (Red Hat Enterprise Linux 5) system with two Intel Core 2 Duo processor and with 4GB of RAM.

### 5.1 Performance Metrics

We evaluate the system performance for varying complexity of property vocabulary schemas and constraints, and for increasing number of entities in the registry. We evaluate performance in terms of *scalability* - here, a measure of the time taken for a new entry (publisher or consumer) to be mutually matched with existing entries and constraints in the INFOD registry. The performance of the registry depends on the complexity of the instances and constraints, and by the number of registered entities. The system was evaluated for increasing number of publishers, consumers and subscribers and subscriptions in the registry. In the simulation, entries either expressed constraints or did not express constraints which were then treated as NULL constraints - meaning that the entry would be matched with every other entity in the system.

Again, in the INFOD system, property vocabularies are the XML schemas and the property vocabulary instances are XML documents adhering to the schema. In general, the complexity of an XML document is determined by its structural and also the built-in

```

<node1>nodeValue</node1>
<node3>
  <node3_child1>nodeValue</node3_child1>
</node3>
<node5>
  <node5_child1>
    <node5_child2>nodeValue</node5_child2>
  </node5_child1>
</node5 >

```

Table 5.1: Example of structural variation in XML - for property vocabulary instances

functional components. The size of the XML document, meaning the number of elements, and the number of hierarchical elements - parent-child nodes or complex elements - influence the structural complexity of an XML document. The functional complexity is defined by the number of user-defined data types, enumeration types, lists and data type restrictions. For our tests, we consider complexity of the instances solely based on the structure of the XML. We generated XML documents representing property vocabulary instances, which was a single XML element, a node with one child element, or a complex element with one child and a sub-child element - See examples in Table 5.1.

The performance and the scalability of the system depend on the mutual filtering process in the registry. In our implementation for matching purposes, the publisher and data source are considered representing identical entities (since in the INFOD model their constraints have the same structure). The subscription has its property constraints and inherits the property constraint of the subscriber, and is represented as a single entity for evaluation purposes. In this section, whenever we mention subscription we imply that it includes the subscriber’s constraint.

## 5.2 Evaluation Results

With the mutual filtering optimization procedure, as detailed in the previous chapter, the system was evaluated for its scalability, the results are as shown in Fig. 5.1. For a registry setup with 1000 consumers, publishers, and subscribers, the system with the intermediate mapping tables took around two seconds to match a new entry, as compared to the direct evaluation which took more than four minutes. In the plot,  $y$  axis indicates the time taken

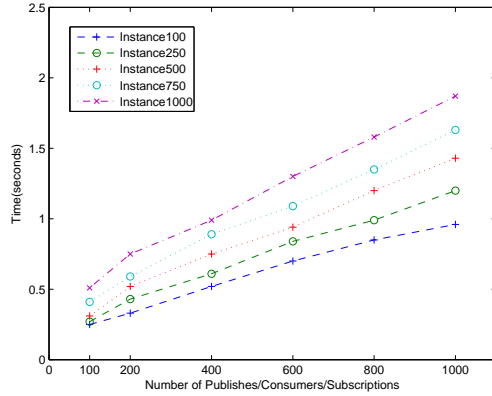


Figure 5.1: System Scalability

for a new entry to be matched with the existing entities in the INFOD registry, the  $x$  axis representing the number of entities in the registry. The legend ‘Instance1000’ refers to an entry’s property vocabulary instance of 1000 elements, and similarly ‘Instance500’ refers to an property vocabulary instance of 500 elements. With the intermediate step, the system scaled sub-linearly for increasing number of entities. The main reason being we had reduced the number of computations.

Fig. 5.2 shows the performance of the system for a fixed number of entities but the constraints defined by the entities vary. A consumer entry has constraints on subscribers and publishers, similarly a publisher has constraints on subscribers and consumers, and a subscription has constraint on publishers and consumers. M1 represents the system run where all entities defines constraints and M2 represents the run where publishers and consumers do not specify constraints against a subscriber. The size of the vocabulary instance is increased linearly along the  $x$  axis for a set number of publishers, consumers and subscriptions - in this case 400. The  $y$  axis denotes the time taken for matching a new consumer. It is intuitive to conclude that the matching would be faster with lesser constraints, as lesser number of constraints needs to be evaluated. However, the system responds faster with more constraints than with lesser constraints. We attribute this counter-intuitive result to the intermediate stage in the matching procedure. With fewer constraints, more entities are a candidates for a match - thus the intermediate mapping table increases in size. In the setup with lesser constraints, the three-way join across the mapping tables consumes more

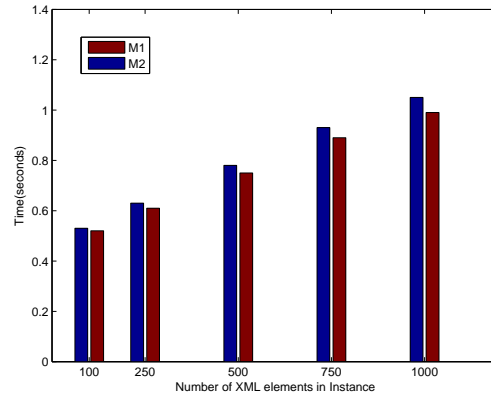


Figure 5.2: Effect of constraints on system performance

time, as the number of computations is the cross product of the number of evaluations. Say each mapping table has 10 entries, the number of evaluations for the join operation would be the product of the number of entries, requiring thousand runs. If the size of the intermediate mapping table is doubled to 20 entries, the number of evaluations required for the join is eight thousand computations. However, from the plot the time difference is in the order of milliseconds. This is because the three-way join is a simple ROWID match but not XPath evaluations. Nevertheless, the performance of the system with the intermediate step is far better than the registry implementation without the intermediate step.

The performance of the system was also evaluated for varying complexity of the XML schema (vocabulary instance). Three different types of vocabulary instances were adopted. In the simplest of schema, every XML tag is a simple element by itself and has no child nodes, this is denoted as t0 in the plot. The second schema, represented as t1, all the nodes have one child element and the XQuery/XPath constraint is defined on the value of the child element. The third schema supports a two level schema hierarchy and is represented as t2 in the plot. In this case, every element in the schema has one child node, and the child node has a single child element. Constraints are defined on the child element's value. The XPath constraint defined in this setup do not point to any parent nodes, they simply refer to the child element. The performance of the system is noted for different length of vocabulary instances, increased linearly along the  $x$  axis, as shown in Fig. 5.3. The  $y$  axis denotes the time taken for matching a new consumer into the system for different schema complexity.

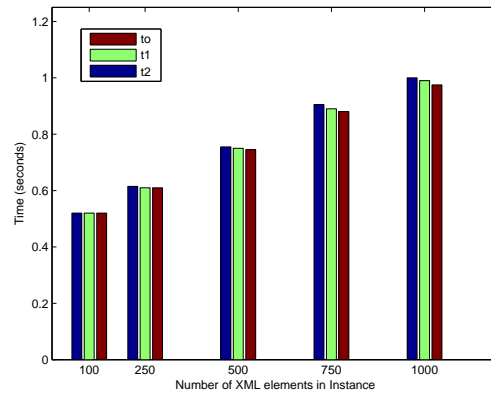


Figure 5.3: Effect of schema complexity on system performance

We found for short schema instances, the difference in time for evaluation of constraints with varying schema complexity was insignificant. For increasing schema length, the match time increased for greater structural schema complexity. For a 1000 element schema instance, the time difference is in the fraction of milliseconds.

## Chapter 6

# Use Case Description

Useability and extensibility are features an application developer or consumer looks towards using a system. The INFOD system provides the infrastructure for easy and quick establishment of a community for sharing information. Establishing a community through the INFOD system can be viewed as a four-step process.

Step 1: Identify and register vocabularies describing community members, *property vocabulary* and semantics of data disseminated by publishers, *data vocabulary*.

Step 2: Register entities with the registry by creating entry's and property vocabulary instance.

Step 3: Create subscription by subscribers. With the availability of many tools and freeware, like GME, XMLspy, Jdeveloper, a simple script can generate graphical interfaces of registered XML schemas (vocabularies), which enables easy creation of property instances and subscriptions coherent with schema definitions in the registry.

Step 4: Match the entities through mutual filtering and notify each entity on the results of mutual filtering. Publishers after receiving information on consumers and constraints binding events and messages, disseminate information to candidate consumers.

### 6.1 The First Responder Use Case

Emergency response systems are applications where the need for disseminating the right information to the right person within a critical time is of prime importance. The first

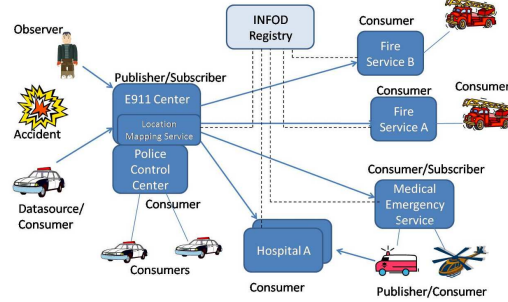


Figure 6.1: First Responder Use Case

responder use case introduces unique and complex requirements requiring an *event-driven* system. In emergency situations, it is best to know the resources and entities that can be requested prior to the incident. In other words, being prepared for a range of potential emergency situations. Most pub-sub systems act on the incident information, evaluating policies and constraints after the incident has occurred. With policies defined for every possible incident, it is the responsibility of an officer to familiarize with all these policies and act on them, which is not practical. In the INFOD model, the entities are matched before the event occurs, and only the information being sent to individual entities changes based on the incident. For example, in an industrial area, it would be beneficial if the local authorities knew what resources are required and available to respond to a specific industrial accident. The decision to request these resources is made only when an incident occurs. In this use case, the E911 center creates subscriptions which characterize events and the necessary action to be taken once an event unfolds. For example, if a fire occurs at a residential building, the services dispatched and the nature of the response differs from those, if the fire takes place at an industrial facility. Defining different type of events, the actions to be initiated, and the specific consumer to be alerted are defined in the subscription.

Fig. 6.1 illustrates a simplified first responder use case, where information needs to be disseminated among entities in the system. This simplified use case is sufficient to help us describe the working of the INFOD system. The E911 center is the first place where an accident or incident gets reported, possibly by a bystander who happens to be at the location at the time of the event. Based on the information received, the operator at the E911 center dispatches the required resources like police, fire service, and ambulances. The

operator follows a set of policies that outlines on what services to put into action for a particular type of event. When the first police officer reaches the event location, the officer reports more specific details and based on the changing event information new actions are taken. In the real-world, an officer at the scene or E911 assumes a role of *Incident Commander* (IC), and determines the action and resources to be enforced. In our example, we assume the incident commander is located at the E911 center and has the same function.

The effectiveness of managing an incident closely depends on the IC's familiarity with policies, access to information and the knowledge of the current state of resources and services. The current state of the art systems provide a location map of available resources and the IC makes the decision. For example, an E911 operator may need to dispatch a first responder nearest to the incident by looking at a location mapping system, however the shortest geographic distance between the responder's location and event location might not suggest the shortest travel time. Though polices and procedures are written anticipating emergency situation, only the well-directed implementation of the policies proves to be efficient. INFOD addresses this problem through its subscriptions, which efficiently model an "*incident object*" and help enforce policies based on the identified incident.

As shown in Fig. 6.1, every entity is identified as a publisher, consumer and/or subscriber based on their roles. The E911 center is a publisher, sending notification messages to first responder services. The police control center is associated with the E911 center and is a consumer of the information disseminated by the E911 center. The police officers (and patrol cars) are resources of the control center and are also consumers of messages from the E911 center. When an incident is reported by a bystander to the E911 center for the first time, an officer is dispatched to the location who checks and provides more information on the event. The first reporting police officer in this case acts as a data source, providing current event information to the E911 center. The fire and medical services are consumers of information acting based on the notification message from the publisher (E911 center). The fire and medical services have resources like fire engines, ambulances, helicopters, and are modeled as consumers as they may interact directly with the E911 center when in action. The ambulances and helicopters are also identified as publishers as they might publish critical patient information to hospitals. Hospitals are modeled as consumers. The

<i>Predicates</i>	<i>Description</i>
Organization	Name, ID, SubUnit details
Location	Physical location(State,County)
Contact Information	Name,Email,Phone,Fax
Associated Organization	Affiliations
Resource Information	Type,Classification,Availability

Table 6.1: Property Vocabulary

<i>Predicates</i>	<i>Description</i>
Activity	Description between time period
Event	Description at a specific time
Action	Response description
Alert Status	Actual, Exercise, System, Test
Urgency	Immediate, Expected, Future, Past
Severity	Extreme, Severe, Moderate, Minor
Certainty	Very likely, Likely, Possible, Unlikely
Category	Geo, Met, Safety, Env, Transport, CBRNE
Substance	Chemical, Bio ,Radioactive materials

Table 6.2: Data Vocabulary

E911 center and the medical services act as subscribers, creating subscriptions which are the policies modeled as constraints for information dissemination in this use case. The first responder entities in this use case may be mobile and capturing such dynamic information in the registry would trigger frequent computation in the registry, with no changes in mutual filtering results. To avoid this situation, we make a further simplification and assume there is a location mapping service which can track resource movement and location at all times. This assumption is reasonable because many E911 systems implement such a capability for their use.

As stated earlier, the first step towards using an INFOD system in identifying vocabularies. In this use case, we use NIEM (National Information Exchange Model) [14] and CAP (Common Alerting protocol) [13] schemas as vocabularies. NIEM is an XML schema description for sharing critical information in the Departments of Justice, and Homeland Security. CAP is an XML message structure for describing emergency and disaster alerts. NIEM and CAP are extensive schema documents - we use a small portion in this use case. Table 6.1 gives a property vocabulary to describe first responder services in this use case.

<b>Publisher</b>
<i>for</i> \$sub in fn:collection(\$\$infodsubscriber) <i>where</i> \$pub//OrganizationParent/ OrganizationName="KnoxFireDept" <i>for</i> \$con in fn:collection(\$\$infodconsumers) <i>where</i> \$con//OrganizationName="FireStation28"
<b>Consumer</b>
<i>for</i> \$pub in fn:collection(\$\$infodpublisher) <i>where</i> \$pub//Location/LocationCounty="CountyA" and \$pub//Location/LocationCounty="CountyC" and \$pub//Location/LocationCounty="CountyX"
<b>Subscriber</b>
<i>for</i> \$pub in fn:collection(\$\$infodpublisher) \$con in fn:collection(\$\$infodconsumers) <i>where</i> \$con//OrganizationParent/OrganizationName =\$pub//OrganizationParent/OrganizationName

Table 6.3: Property Constraints

Information on the service’s organization, location, resources, point of contact and affiliations describe first responder entities. Table 6.2 lists the data vocabulary for describing alert messages based on the types specified in the NIEM and CAP schemas. The terms in the table identify semantics for describing an event or activity in a region. The E911 center classifies or describes events in terms of severity, urgency, category, and certainty. The data vocabulary is also used in the subscription’s data constraint, described later in this section.

User entities create entries as publishers, datasources, consumers, and subscribers in the INFOD registry. As described earlier, every entity specifies its interest as a property constraint. Table 6.3 details property constraints defined by publishers, consumers, and subscribers. The publisher defines constraints on consumer’s and subscriber’s property vocabulary instances. The publisher’s constraint identifies consumers and subscribers belonging to a specific organization as a constraint. Thereby, subscriptions created by subscribers satisfying the constraint are only applicable to this particular publisher. Similarly, though subscriptions might associate this publisher to consumers belonging from various organizations, the publisher has limited it’s consumers to only those belonging to the specific organization. The consumer entry defines property constraints on publishers and subscribers of interest. From Table 6.3, the consumer is interested in receiving information from publisher located in specific counties only. In this example, the consumer has

<b>Property Constraint</b>
<pre> for \$pub in fn:collection(\$\$infodpublisher)   where \$pub//OrganizationSubUnitName="E911" for \$con in fn:collection(\$\$infodconsumers)   where \$con//OrganizationCategory ="FirstResponders" </pre>
<b>Data Constraint</b>
<pre> declare namespace \$data = http:// infod.firstresponder.net.com/AlertDataVocabulary; let \$msg1 :=   for \$firstresponders in fn:collection(\$\$infodconsumer)     where \$data:AlertStatus = Actual and           \$data:EventCategory = CBRNE and           \$data:EventSeverity &gt; Moderate     return \$data,\$data:Instruction=Evacuate people let \$msg2 :=   for \$firstresponders in fn:collection(\$\$infodconsumer)     where \$data:capAlertStatus = Actual and           \$data:EventCategory = Fire and           \$data:EventSeverity = Moderate     return \$data:Substance </pre>
<b>Dynamic Consumer Constraint</b>
<pre> for \$firstresponders in fn:collection(\$\$infodconsumer)   where \$firstresponders//SubUnitName=Police     and fn:distance(\$firstresponder) &lt; 20   return \$msg1 for \$firstresponders in fn:collection(\$\$infodconsumer)   where \$firstresponders//SubUnitName=FireService   return msg2 </pre>

Table 6.4: Subscription

no constraints on the subscriber, meaning the consumer can be mapped with all subscribers and their subscriptions. The subscriber identifies publishers and consumers on whom its subscriptions are applicable. In this case, the subscriber's property constraint identifies only publishers and consumer belonging to the same parent organization for evaluating its subscription.

The INFOD subscription consists of a property constraint, data constraint, and dynamic consumer constraint as shown in Table 6.4. The property constraint defined by the subscription is similar in structure to constraints defined in the subscriber entry. The data constraint defines events of interest at the publisher. In this case we are interested in two events: an actual CBRNE (*\$data:EventCategory = CBRNE*) event with severity greater than moderate (*\$data:EventSeverity > Moderate*) and a fire event (*\$data:EventCategory = Fire*) with a moderate severity. The data constraints also have a return type specifying what message needs to be disseminated if the described event occurs. Note that the XPath elements specified in the data constraint is from the data vocabulary elements. To whom the message should be sent at run-time is limited by the dynamic consumer constraint. In this case, the dynamic consumer constraint is to find the closest first responder within a twenty mile radius and the first responder should belong to a specific unit.

Having created entries, constraints, and subscriptions in the INFOD registry, the registry performs mutual filtering and notification messages are sent to the matched entities. Recall, that the mutual filtering operation in the INFOD registry is only based on property constraints. The registry in its notification message to the publisher identifies a set of consumers that it matches, and also the data constraint and dynamic consumer constraint. When an event is triggered at the publisher, the publisher checks if it is identified as an event of interest in the data constraint and then evaluates the dynamic consumer constraint to determine any specific consumers for notification. The publisher might require information from the registry to determine specific consumers - it obtains this information using the `getMetaData` function. As the event state changes, the publisher reevaluates the data constraint and dynamic consumer constraint, and notifies consumers of the event change. If the data constraint identifies an event and no specific consumer is referenced in the dynamic

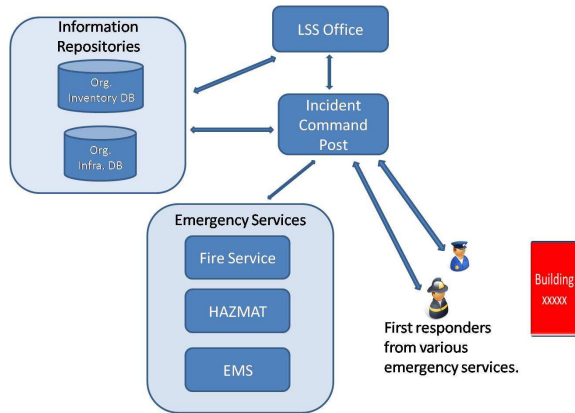


Figure 6.2: LSS Use Case

consumer constraint, then a notification message is sent to all consumers specified in the notification message received by the publisher from the INFOD registry.

This use case is developed as an example for demonstrating and understanding the functionalities of INFOD.

## 6.2 LSS Use Case

This use case is based on our interaction with the Laboratory Shift Superintendent's (LSS) Office at the Oak Ridge National Laboratory. We interacted with the LSS office to identify what are the critical needs when it comes to information sharing during an emergency. The end goal was in developing a system that would provide a better means of sharing information during critical times for the LSS office.

Fig. 6.2 illustrates the various entities and interaction between them during an emergency. The LSS office is the main command and control center for all emergency response actions initiated within the laboratory. The officers at the LSS office have access to all the necessary data required for handling an emergency. The data is stored across multiple data repositories, information such as floor plan, building occupancy, flammable and hazards material are a few information the emergency responders would require to execute a emergency response action. Also, the LSS office communicates to all the emergency response teams, such as the HAZMAT (hazards material), fire, and EMS (emergency medical services). At the time of an event, an incident command post is set up at close proximity to the location of

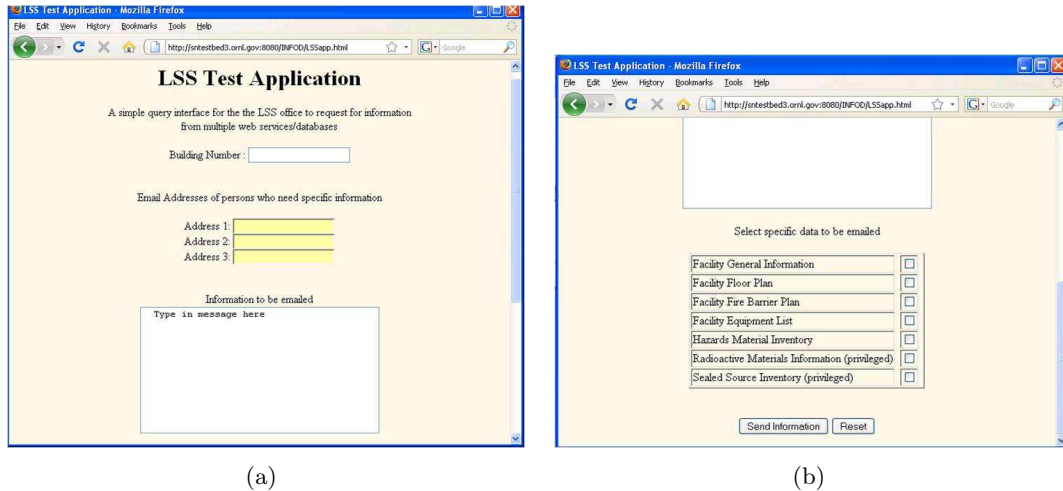


Figure 6.3: LSS Application

the incident. The officer at the incident command post, Incident Commander, coordinates activities among the various first responder teams. Also, each emergency service requires specific information, e.g., the fire service would require a layout of the building, the HAZMAT service might require inventory information and EMS service would want an estimate of the number people located in the building. The incident commander communicates with the LSS office and provides the necessary information to the emergency services. Also, as the first responders arrive at the location and as the event unfolds, the need for resources and equipment change which is also coordinated by the incident commander.

On discussing with the LSS office the main problem in executing plan was the access to information which is being stored across multiple repositories. The person in charge at the time of the event needs to query independently from every repository for such information. To solve their problem, we developed a solution of the LSS office using existing technologies for accessing information from multiple repositories. As shown in Fig. 6.3 we developed a simple application which would accept the building number, email address of the people to whom the information needs to be communicated and the specific details to be picked from multiple data repositories. Our application queries from multiple data repositories and sends all information packed into single email to the respective persons. We received great support and the LSS office acknowledged the need for such a simple system. The next

step is using the INFOD system and showing to them how well information can be directed to emergency people with little or no human intervention.

### **6.3 Extending the Use Case for DHS Applications**

The first responder use case was motivated by the information flow dynamics and the complexity in modeling information flow patterns as constraints, which would be the case in real world applications. The development of the LSS use case was towards demonstrating INFOD functionality in a smaller scale and validating its effectiveness as an event-based information dissemination tool. Through the first responder use case we were able to model multiple entities with different roles and specific requirements for sharing data, which is typical of DHS applications.

In the course of the current project, information sharing among fusion centers was considered as an ultimate target scenario where INFOD could be demonstrated. However, due to the lack of the availability in the policies modeling information sharing among fusion centers, which is itself a problem being addressed currently, the first responder use case turned out to be the near real-world scenario we could model with the expertise of first responders from the University of Tennessee and Oak Ridge National Laboratory. However, the field trip to the Shelby County Fusion Center helped us correlate between the first responder use case and the fusion center use case. One of the key differences between the two use cases was that the number of policies/constraints a fusion center would typically handle is higher in number and more varied because of the larger volume of data to be handled than the first responder use case. This was addressed through the scalability tests on INFOD, which validated our model to be effective for real-time operations. Another difference, the user dynamics in the first responder was more demanding than that of typically observed in a fusion center use case. However the INFOD approach is capable of handling such dynamics.

One of the challenges posed by the fusion center project with information sharing across state fusion centers was in the management of the INFOD registry. In our use case, we demonstrated using a single INFOD registry which is capable of handling thousands of user

entities and constraints. However, the challenge here was every state would like to control and own their registries, but registries would need to co-ordinate. The concept of distributed registries was proposed in the INFOD approach which would help solve the problem. The distributed registry concept will be discussed as one of the future works.

## Chapter 7

# Conclusion

The main contributions of INFOD to the area of information dissemination are in its vocabulary and constraint definitions. These form critical building blocks in applications such as inter-agency data sharing and information exchange where events and data flowing across entities are heavily burdened by factors such as terminology differences and access restrictions. Standard vocabularies such as NIEM could be adopted (even in part) as core vocabularies and flexible pub-sub communication can easily become a reality for a large community of users. By supporting a system of mutual filtering, publishers and subscribers express constraints against each other and not just on the data. More importantly, the INFOD approach makes the support infrastructure act only as a broker (match agent) and not as a data intermediary. This improves the security profile of the system significantly.

### 7.1 Project Outcome

The collaborative effort of academia, national laboratory and industry brought in mixed ideas and novelty towards building the INFOD approach. With Oracle contributing in software and technical guidance towards the development of the project, the members had an opportunity to picture what the real world needs are in such critical information sharing infrastructure and what components in the database might add to the strengths of such a system.

Two graduate students, Ms Ying Sun and Mr. Samir Sahyoun, have defended their contributions towards a Master's thesis, and Samir Sahyoun is continuing towards pursuing his Ph.D. Another Ph.D. candidate, Mr. Ming Chen, has included the contributions to INFOD development as a part of his dissertation.

We have made seven presentations at various conferences and two more papers are currently under review.

At the 2009 DHS University Network Summit, we have presented three student posters.

## 7.2 Summary of Technical Outcome

The INFOD model has largely advanced the information sharing study by introducing a flexible and dynamic framework for brokering information between entities. In the following, we summarize its technical merits.

First of all, a structured information model is built where user communities are identified by property and data vocabularies. Property vocabularies characterize user entity properties or interests, while data vocabularies describe publisher data semantics or capabilities with available data. The data source entry associates publishers to multiple information sources.

Secondly, publishers, consumers and subscribers are identified as real-world entities characterized in terms of vocabularies and their interests as “constraints” within the INFOD registry.

Third, a mutual filtering engine is developed that associates entities in INFOD. A three-way join across publishers, consumers, and subscriber entities is realized through which the INFOD registry matches publishers and consumers based on information needs expressed through subscriptions and limited by properties, after which it sends notification messages to user entities informing them of their associations.

Fourth, a dynamic reevaluation scheme is enabled such that when user properties or subscriptions change, and/or new users/subscriptions are registered, the system can reevaluate granting/limiting associations for information sharing.

Fifth, a prototype of the INFOD model has been built with the above capabilities.

Finally, a simple first responder emergency alert use case with interactive web interfaces has been demonstrated at SERRI review meetings and project quarterly review meetings.

In summary, with a number of policies/constraints to be enforced while sharing such critical information, the INFOD system provides the capability to efficiently communicate the right information to the right person at the right time.

### 7.3 Discussions and Future Work

In the course of modeling INFOD and the first responder use case, our team and the INFOD working group have identified a list of capabilities that INFOD needs to support. We have implemented the core capabilities during this period of project. In the following, we list a few recommendations which are proposed in the INFOD specification,

- A more structured vocabulary schema - Classify attributes as dynamic, transient, or static where dynamic refers to those attributes that change frequently, e.g., the latitude and longitude of a moving vehicle. A transient attribute changes over a certain time or in periodic intervals. Static attributes are ones that do not change or change only occasionally. It is best if dynamic attributes are always a part of the data vocabulary, while static attributes could be either in the property or data vocabulary depending on their function of either describing an entity or describing an event of interest. The handling of transient attributes depends upon the information dissemination needs and the capability of the INFOD registry to deal with updates at a certain frequency.
- A formal definition of constraint - Model constraints as XQuery expressions and evaluate these constraints.
- The concept of a disseminator entity - Provide a means of cohering information from multiple publishers and directing them towards consumer entities.
- The need for a distributed registry.
- The need of security policies dealing with authorization and authentication of user entities and the information being shared.

### 7.3.1 Security Considerations

INFOD operates in a web services environment with users across various domains accessing registry resources compelling a secure communication paradigm for *Authentication* and *Authorization* within the registry. Publishers, consumers, and subscribers access the INFOD registry through SOAP messages requiring a secure point-to-point

communication model, and also publishers exchange information with consumers demanding mutual authentication and establishing secure encrypted channels for information exchange. Publishers, consumers, and subscribers are modeled as users accessing INFOD services and a single user could be a publisher, consumer, and/or subscriber. In the INFOD model, *Authentication* deals with user certification in the registry and *Authorization* on administering access control policies for user privileges to resources and messages from the registry.

*Authentication*, the registry creates security tokens, associates every user to a security token and authenticates user in the registry, as in any public key infrastructure model. INFOD operating as a SOAP based web service, securing SOAP communication between entities is as described in the *WS Security* specification [9]. The specification handles secure point-to-point communication, maintains user tokens/certificates, and supports message encryption. *WS Secure Conversation* specification [11] provides the mechanism for establishing secure long-term point-to-point communication channels as required for communication between publishers and consumers.

*Authorization* defines firstly, user access policy to resources in the registry and secondly, messages that a user is entitled to receive as a result of mutual filtering in the registry. For example, user access policy determine if the consumer is allowed to view publisher information in the registry, however, this does not block the publisher and consumer from being matched. User access policy specifies user privileges to create, replace, and/or drop a resource in the registry, and the privilege for performing meta-data operations. User privileges are enforced through a Role-Based Access Control (RBAC) policy, where a community user with administrator access creates and assigns roles to users. Roles could be generic, such as a *publisher* role which would allow a user to create, replace, and drop

publisher entry in the registry, or could be application specific, a user role only to replace certain attributes in the property vocabulary instance and no create/drop privileges. The RBAC policy is enforced as described in the OASIS XML Access Control Markup Language (XACML) RBAC specification [12]. The second feature of limiting users to messages from the registry is enforced by assigning constraints to specific resources in the registry as a security constraint. This feature is for INFOD user to add additional constraints as a security measure and would be applied on the existing association among entities. If this was to be modeled as a part of the subscription, any change would trigger the mutual filtering process in the registry. However, adding it as a security constraint would apply the constraint on the existing matched result. Also removing the security constraint would mean reverting back to the previously matched results.

### **7.3.2 Distributed Registry**

The current model of INFOD is based on a centralized registry, the next stage would be in extending it to network of registries. Though centralized model acts as a single point of failure and a performance bottle-neck, the current INFOD approach is better than conventional centralized pub-sub models as INFOD does not handle data, unlike other model which are in the path of the data. Recollecting, INFOD only serves as a broker for information dissemination, and connects potential publishers and consumers. However, when information needs to be disseminated across multiple enterprises, such as across state fusion centers, where each enterprise would like to control and maintain the INFOD registry independently but are interested in sharing information between organizations.

With a working prototype of INFOD, the next development is in the real-time deployment of the INFOD system which would help understand and address problems in the real world.

# Bibliography

- [1] Information Dissemination Base Specification, Open Grid Forum (OGF), Jan 2008, <http://forge.gridforum.org/sf/projects/infod-wg>.
- [2] Information Dissemination Base Use Case Scenarios, Open Grid Forum (OGF), Nov 2007, <http://forge.gridforum.org/sf/projects/infod-wg>.
- [3] D. Gawlick, D. Lenkov, A. Yalamanchi, L. Chernobrod “Applications for Expression Data in Relational Database Systems,” in *Proceeding of the 20th International Conference on Data Engineering*, April 2004.
- [4] G. Banavar, Chandra, et al, “An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems,” in *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, 1999.
- [5] M. Petrovic, I. Burcea, H.-A. Jacobsen. “S-ToPSS: Semantic Toronto Publish/Subscribe System,” In *Proc. of Conf. on Very Large Data Bases*, September, 2003.
- [6] Luis F. Cabrera, Michael B. Jones, and Marvin Theimer. “Herald: Achieving a Global Event Notification Service.” In *HotOS VIII*, May 2001.
- [7] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. “Achieving Expressiveness and Scalability in an Internet-Scale Event Notification Service.” *Nineteenth ACM Symposium on Principles of Distributed Computing (PODC2000)*, Portland OR. July, 2000.
- [8] JMS Specification, <http://java.sun.com/products/jms/docs.html>
- [9] Web-Service Security Specification, <http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>

- [10] Web-Service Notification Specification, [http://docs.oasis-open.org/wsn/wsn-ws\\\_base\\\_notification-1.3-spec-os.pdf](http://docs.oasis-open.org/wsn/wsn-ws\_base\_notification-1.3-spec-os.pdf)
- [11] Web-Service Secure Conversation, <http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.3/ws-secureconversation.pdf>
- [12] XACML Profile for Role Based Access Control, [docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf](http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf)
- [13] CAP, Common Alerting Protocol, <http://www.oasis-open.org/committees/download.php/14759/emergency-CAPv1.1.pdf>
- [14] NIEM, National Information Exchange Model, <http://www.niem.gov/>
- [15] DHS Science and Technology Directorate, “High-Priority Technology Needs”, June 2008. [http://www.dhs.gov/xlibrary/assets/High\\_Priority\\_Technology\\_Needs.pdf](http://www.dhs.gov/xlibrary/assets/High_Priority_Technology_Needs.pdf)
- [16] SERRI Information Sharing & Management Projects, <http://www.serri.org/research/Pages/informationsharingmanagement.aspx>

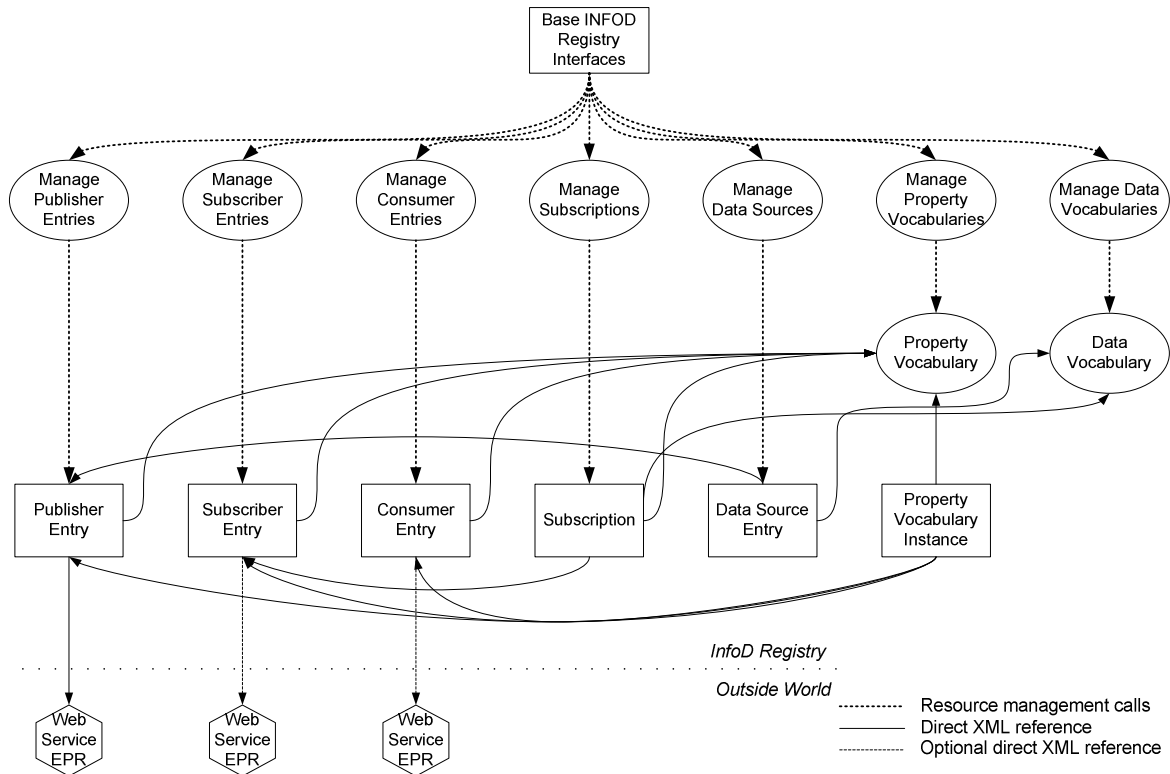
---

## Appendix A – XML Schema

This section includes the following xml schema:

- Publisher Entry (section 1.1)
- Subscriber Entry (section 1.2)
- Consumer Entry (section 1.3)
- Subscription (section 1.4)
- Property Vocabulary (section 1.5)
- Property Vocabulary Instance (section 1.6)
- Data Vocabulary (section 1.7)
- Data Source Entry (section 1.8)
- INFOD Error Messages (section 1.9)
- INFOD Notification (section 1.10)
- Publishers Notification (section 1.11)
- Subscriber Notification (section 1.12)
- Consumer Notification (section 1.13)

The following graphic depicts the XML schema relations for the 'www.ggf.org/INFOD/INFODRegistry' Namespace. The circles in the first row represent the operations of the INFOD registry. The circles in the second row show the vocabularies that are managed by the vocabulary operations. The boxes in the third row represent the resources, data entries and property vocabulary instances. Within the xml schema of those boxes, there are reference pointers to other entries or vocabularies, represented by URIs. The honeycombs represent the external web services URIs that are associated to the resources. Note that the same Web Service URI can be associated to multiple INFOD resources.



**Figure 1: XML schema relations of INFODRegistry namespace**

Figure 6, provides a schematic representation of internals of INFOD registry. Web Service URIs from publisher, subscriber and consumers are associated with publisher, subscriber and consumer entries respectively. Associations between entries with data and property vocabulary instances are also highlighted.

## 1.1 Publisher Entry

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ident="http://www.ggf.org/INFOD"
  targetNamespace="http://www.ggf.org/INFOD/INFODRegistry">

  <xsd:element name="infodPublisherEntry">
    <xsd:annotation>
      <xsd:documentation>
        Description of Publisher Entries
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="WSReference"
          type="wsa:EndpointReferenceType"
          minOccurs="0" maxOccurs="1"/>
        <xsd:element name="PublisherName" type="xsd:string"
          minOccurs="0" maxOccurs="1"/>
        <xsd:element name="PublisherDescription" type="xsd:string"
          minOccurs="0" maxOccurs="1"/>
        <xsd:element name="PropertyConstraint" type="xsd:any"
          minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="Notification" type="xsd:boolean"

```

```

nillable="true"
minOccurs="0" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>

```

## 1.2 Subscriber Entry

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.ggf.org/INFOD"
            targetNamespace="http://www.ggf.org/INFOD/INFODRegistry ">
  <xsd:element name="infodSubscriberEntry">
    <xsd:annotation>
      <xsd:documentation>
        Description of Subscriber Entries
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="WSReference"
                    type="wsa:EndpointReferenceType"
                    minOccurs="0" maxOccurs="1"/>
        <xsd:element name="SubscriberName" type="xsd:string"
                    minOccurs="0" maxOccurs="1"/>
        <xsd:element name="SubscriberDescription" type="xsd:string"
                    minOccurs="0" maxOccurs="1"/>
        <xsd:element name="PropertyConstraint" type="xsd:any"
                    minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="Notification" type="xsd:boolean"
                    nillable="true" minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

## 1.3 Consumer Entry

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.ggf.org/INFOD"
            targetNamespace="http://www.ggf.org/INFOD/INFODRegistry ">
  <xsd:element name="infodConsumerEntry">
    <xsd:annotation>
      <xsd:documentation>
        Description of Consumer Entries
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="WSReference"
                    type="wsa:EndpointReferenceType"
                    minOccurs="0" maxOccurs="1"/>
        <xsd:element name="ConsumerrName" type="xsd:string"
                    minOccurs="0" maxOccurs="1"/>
        <xsd:element name="ConsumerDescription" type="xsd:string"
                    minOccurs="0" maxOccurs="1"/>
        <xsd:element name="PropertyConstraint" type="xsd:any"
                    minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

```

    <xsd:element name="Notification" type="xsd:boolean"
      nillable="true" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>

```

## 1.4 Subscription

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ident="http://www.ggf.org/INFOD"
  targetNamespace="http://www.ggf.org/INFOD/INFODRegistry ">

  <xsd:element name="infodSubscription">
    <xsd:annotation>
      <xsd:documentation>
        Description of Subscriptions
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="SubscriptionName" type="xsd:string"
          minOccurs="0" maxOccurs="1"/>
        <xsd:element name="SubscriptionDescription" type="xsd:string"
          minOccurs="1" maxOccurs="1"/>
        <xsd:element name="SubscriberEntryReference"
          type="wsa:EndpointReferenceType"
          minOccurs="0" maxOccurs="1"/>
        <xsd:element name="DataConstraint" type="xsd:any"
          minOccurs="0" maxOccurs="1"/>
        <xsd:element name="PropertyConstraint" type="xsd:any"
          minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="DynamicConsumerConstraint" type="xsd:any"
          minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

## 1.5 Property Vocabulary

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ident="http://www.ggf.org/INFOD"
  targetNamespace="http://www.ggf.org/INFOD/INFODRegistry ">

  <xsd:element name="infodPropertyVocabulary">
    <xsd:annotation>
      <xsd:documentation>
        Description of a Property Vocabulary
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="PropertyVocabularyName" type="xsd:string"
          minOccurs="0" maxOccurs="1"/>
        <xsd:element name="PropertyVocabularyDescription" type="xsd:string"
          minOccurs="0" maxOccurs="1"/>
        <xsd:element name="PropertyVocabularyBody" type="xsd:any"
          minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

```

    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

## 1.6 Property Vocabulary Instance

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.ggf.org/INFOD"
            targetNamespace="http://www.ggf.org/INFOD/INFODRegistry ">

  <xsd:element name="infodPropertyVocabularyInstance">
    <xsd:annotation>
      <xsd:documentation>
        Description of Property Vocabulary Instance
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="EntryReference"
                    type="wsa:EndpointReferenceType"
                    minOccurs="1" maxOccurs="1"/>
        <xsd:element name="PropertyVocabularyReference"
                    type="wsa:EndpointReferenceType"
                    minOccurs="1" maxOccurs="1"/>
        <xsd:element name="PropertyVocabularyInstanceBody"
                    type="xsd:schema"
                    minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

## 1.7 Data Vocabulary

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.ggf.org/INFOD"
            targetNamespace="http://www.ggf.org/INFOD/INFODRegistry ">

  <xsd:element name="infodDataVocabulary">
    <xsd:annotation>
      <xsd:documentation>
        Description of Data Vocabulary
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="DataVocabularyName" type="xsd:string"
                    minOccurs="0" maxOccurs="1"/>
        <xsd:element name="DataVocabularyDescription" type="xsd:string"
                    minOccurs="0" maxOccurs="1"/>
        <xsd:element name="DataVocabularyLanguage" type="xsd:string"
                    minOccurs="1" maxOccurs="1"/>
        <xsd:element name="DataVocabularyBody" type="xsd:any"
                    minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

## 1.8 Data Source Entry

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.ggf.org/INFOD"
            targetNamespace="http://www.ggf.org/INFOD/INFODRegistry ">

  <xsd:element name="infodDataSourceEntry">
    <xsd:annotation>
      <xsd:documentation>
        Description of Data Source Entries
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="CreateDataSourceEntryName" type="xsd:string"
                    minOccurs="0" maxOccurs="1"/>
        <xsd:element name="DataSourceEntryDescription"
                    type="xsd:string"
                    minOccurs="0" maxOccurs="1"/>
        <xsd:element name="PublisherEntryReference"
                    type="wsa:EndpointReferenceType"
                    minOccurs="1" maxOccurs="1"/>
        <xsd:element name="DataVocabularyReference"
                    type="wsa:EndpointReferenceType"
                    minOccurs="1" maxOccurs="1"/>
        <xsd:element name="PropertyConstraint" type="xsd:any"
                    minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

## 1.9 Error Messages

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"

            xmlns:wsrf-bf="http://www.ggf.org/INFOD/fault"
            xmlns:ident="http://www.ggf.org/INFOD"
            targetNamespace="http://www.ggf.org/INFOD/INFODRegistry ">

  <xsd:complexType name="CreateResourceAuthorizationFaultType">
    <xsd:complexContent>
      <xsd:extension base="wsrf-bf:BaseFaultType"/>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="CreateResourceAuthorizationFault"
              type="infod:CreateResourceAuthorizationFaultType"/>

  <xsd:complexType name="ReplaceResourceAuthorizationFaultType">
    <xsd:complexContent>
      <xsd:extension base="wsrf-bf:BaseFaultType"/>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="ReplaceResourceAuthorizationFault"
              type="infod:ReplaceResourceAuthorizationFaultType"/>

  <xsd:complexType name="DropResourceAuthorizationFailureType">
    <xsd:complexContent>
      <xsd:extension base="wsrf-bf:BaseFaultType"/>
    </xsd:complexContent>
  </xsd:complexType>
```

```

</xsd:complexType>
<xsd:element name="DropResourceAuthorizationFailure"
             type="infod:DropResourceAuthorizationFailureType" />

<xsd:complexType name="InvalidExecutionModeType">
  <xsd:complexContent>
    <xsd:extension base="wsrf-bf:BaseFaultType" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="InvalidExecutionMode"
             type="infod:InvalidExecutionModeType" />

<xsd:complexType name="UnsupportedVocabularyFaultType">
  <xsd:complexContent>
    <xsd:extension base="wsrf-bf:BaseFaultType" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="UnsupportedVocabularyFault"
             type="infod:UnsupportedVocabularyFaultType" />

<xsd:complexType name="UnsupportedXQueryFaultType">
  <xsd:complexContent>
    <xsd:extension base="wsrf-bf:BaseFaultType" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="UnsupportedXQueryFault"
             type="infod:UnsupportedXQueryFaultType" />

<xsd:complexType name="GetMetaDataAuthorizationFailureType">
  <xsd:complexContent>
    <xsd:extension base="wsrf-bf:BaseFaultType" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="GetMetaDataAuthorizationFailure"
             type="infod:GetMetaDataAuthorizationFailureType" />

<xsd:complexType name="UnknownResourceReferenceFaultType">
  <xsd:complexContent>
    <xsd:extension base="wsrf-bf:BaseFaultType" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="UnknownResourceReferenceFault"
             type="infod:UnknownElementReferenceFaultType" />

<xsd:complexType name="MissingRequiredParameterFaultType">
  <xsd:complexContent>
    <xsd:extension base="wsrf-bf:BaseFaultType" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="MissingRequiredParameterFault"
             type="infod:MissingRequiredParameterFaultType" />

<xsd:complexType name="UnknownFaultType">
  <xsd:complexContent>
    <xsd:extension base="wsrf-bf:BaseFaultType" />
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="UnknownFault"
             type="infod:UnknownFaultType" />
</xsd:schema>

```

## 1.10 INFOD Notification

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.ggf.org/INFOD"
            targetNamespace="http://www.ggf.org/INFOD/INFODNotify ">
  <!-- ===== Notification Metadata ===== -->
  <xsd:element name="SubscriptionReference"
              type="wsa:EndpointReferenceType"/>
  <xsd:element name="Topic"
              type="infod:TopicExpressionType"/>
  <xsd:element name="PublisherReference"
              type="wsa:EndpointReferenceType"/>
  <!-- ===== Message Helper Types ===== -->
  <xsd:complexType name="TopicExpressionType" mixed="true">
    <xsd:sequence>
      <xsd:any minOccurs="0" maxOccurs="1" processContents="lax"/>
    </xsd:sequence>
    <xsd:attribute name="Dialect" type="xsd:anyURI" use="required"/>
    <xsd:anyAttribute/>
  </xsd:complexType>
  <xsd:complexType name="NotificationMessageHolderType">
    <xsd:sequence>
      <xsd:element ref="infod:SubscriptionReference"
                  minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="infod:Topic"
                  minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="infod:PublisherReference"
                  minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="infod:ConsumerReference"
                  minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="Message">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any namespace="##any" processContents="lax"
                minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="NotificationMessage"
              type="infod:NotificationMessageHolderType"/>
  <!-- ===== Message Types for Consumer Notification by Publishers ===== -->
  <xsd:element name="Notify">
    <xsd:annotation>
      <xsd:documentation> Notification of Consumers by Publishers
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="infod:NotificationMessage"
                    minOccurs="1" maxOccurs="unbounded"/>
        <xsd:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```

    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

</xsd:schema>

```

## 1.11 INFOD Publisher Notification

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"

    xmlns:ident="http://www.ggf.org/INFOD"
    targetNamespace="http://www.ggf.org/INFOD/INFODNotify">

  <xsd:element name="PublisherNotification">
    <xsd:annotation>
      <xsd:documentation>
        Notification of Publishers
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="SubscriptionReference"
          type="wsa:EndpointReferenceType"
          minOccurs="1" maxOccurs="1" />
        <xsd:choice minOccurs="1" maxOccurs="1">
          <xsd:sequence>
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="ConsumerEntryReference"
                  type="xsd:any"
                  minOccurs="1" maxOccurs="1" />
                <xsd:element name="ConsumerReference"
                  type="wsa:EndpointReferenceType"
                  minOccurs="1" maxOccurs="unbounded" />
              </xsd:sequence>
              minOccurs="0" maxOccurs="unbounded" />
            </xsd:complexType>
            <xsd:element name="DynamicConsumerConstraint"
              type="xsd:any"
              minOccurs="0" maxOccurs="unbounded" />
          </xsd:sequence>
          <xsd:sequence>
            <xsd:element name="ConsumerEntryReference"
              type="xsd:any"
              minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="DynamicConsumerConstraint"
              type="xsd:any"
              minOccurs="1" maxOccurs="unbounded" />
          </xsd:sequence>
        </xsd:choice>
        <xsd:element name="DataConstraint"
          type="wsa:EndpointReferenceType"
          minOccurs="1" maxOccurs="unbounded" />
        <xsd:element name="DataConstraint"
          type="xsd:any"
          minOccurs="1" maxOccurs="1" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

## 1.12 INFOD Subscriber Notification

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.ggf.org/INFOD"
            targetNamespace="http://www.ggf.org/INFOD/INFODNotify ">

  <xsd:element name="SubscriberNotification">
    <xsd:annotation>
      <xsd:documentation>
        Notification of Subscribers
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="SubscriptionReference"
                    type="wsa:EndpointReferenceType"
                    minOccurs="1" maxOccurs="1"/>
        <xsd:element name="ConsumerEntryReference"
                    type="wsa:EndpointReferenceType"
                    minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PublisherEntryReference"
                    type="wsa:EndpointReferenceType"
                    minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

## 1.13 INFOD Consumer Notification

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ident="http://www.ggf.org/INFOD"
            targetNamespace="http://www.ggf.org/INFOD/INFODNotify ">

  <xsd:element name="ConsumerNotification">
    <xsd:annotation>
      <xsd:documentation>
        Notification of Consumers
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="SubscriptionReference"
                    type="wsa:EndpointReferenceType"
                    minOccurs="1" maxOccurs="1"/>
        <xsd:element name="PublisherEntryReference"
                    type="wsa:EndpointReferenceType"
                    minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

---

## Appendix B – INFOD API: Emergency Response Use Case

Building a community and enabling information dissemination through the INFOD system may be viewed as a four step process. The first step is to identify and register vocabularies, the property vocabulary for describing community members and the data vocabulary for describing the structure of the publisher information. In the second step, users register with the INFOD registry creating entities and property vocabulary instances. The third step is the creation of subscriptions by subscribers. Finally, the INFOD registry notifies each entity of the results of mutual filtering. Once they have received notifications, it is up to the publishers to disseminate information to the relevant consumers based on the data constraints defining events and messages. The steps along with the INFOD API schemas for creating resources and sending notification messages are briefed below.

### Step 1: Register the vocabularies with the INFOD registry.

#### a) Registration of First Responder Property Vocabulary

Request message:

```
<infod:CreatePropertyVocabulary>
  <infod:VocabularyName>
    FirstResponderPropertyVocabulary
  </infod:VocabularyName>
  <infod:PropertyVocabularyDescription>
    Vocabulary for describing First Responders
  </infod:PropertyVocabularyDescription>
  <infod:VocabularyBody>
    <?xml version="1.0"?>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:ident="http://www.ggf.org/INFOD"
      targetNamespace="http://www.ggf.org/INFOD"
      xmlns:niem="http://www.niem.gov/neim">
      <xsd:element name = "Identification" type = "niem:OrganizationType"/>
      <xsd:element name = "Location" type = "niem:LocationType"/>
      <xsd:element name = "ContactInformation" type = "niem:ContactType"/>
      <xsd:element name= "AssociatedOrganization" type="niem:OrganizationType"/>
      <xsd:element name = "Resource" type = "xsd:string"/>
    </xsd:schema>
  </infod:VocabularyBody>
</infod:CreatePropertyVocabulary>
```

Response message (for success case):

```
<infod:CreatePropertyVocabularyResponse>
  <infod:PropertyVocabularyReference>
    <wsa:Address>
      http://www.ggf.org/INFOD/PropVocab/5687A2740287230587763FD05D32A4F
    </wsa:Address>
  </infod:INFODVocabularyIdentifier>
</infod:CreatePropertyVocabularyResponse>
```

#### b) Registration of Data Vocabulary

Request message:

```
<infod:CreateDataVocabulary>
  <infod:VocabularyName>
    EmergencyResponseDataVocabulary
  </infod:VocabularyName>
  <infod:DataVocabularyDescription>
    Vocabulary for describing events
  </infod:DataVocabularyDescription>
```

```
<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"
    xmlns:niem="http://www.niem.gov/neim"
    xmlns:cap="urn:oasis:names:tc:emergency:cap:1.1">
    <xsd:element name = "Activity" type = "niem:ActivityType"/>
    <xsd:element name = "Event" type = "niem:EventType"/>
    <xsd:element name = "Action" type = "xsd:string"/>
    <xsd:element name = "AlertStatus" type = "cap:AlertType"/>
    <xsd:element name = "Urgency" type = "cap:UrgencyType"/>
    <xsd:element name = "Severity" type = "cap:SeverityType"/>
    <xsd:element name = "Certainty" type = "cap:CertaintyType"/>
    <xsd:element name = "Category" type = "cap:CategoryType"/>
    <xsd:element name = "Substance" type = "niem:SubstanceType"/>
  </xsd:schema>
</infod:VocabularyBody>
</infod:CreateDataVocabulary>
```

**Response message (for success case):**

```
<infod:CreateDataPropertyVocabularyResponse>
  <infod:DataVocabularyReference>
    <wsa:Address>
      http://www.ggf.org/INFOD/DataVocab/5687A2740287245D87763FD05D32A4F
    </wsa:Address>
  </infod:DataVocabularyReference>
</infod:CreateDataPropertyVocabularyResponse>
```

**Step 2: User entities register with the INFOD Registry**

User entities register as publishers, consumers, subscribers, and data sources.

**a) Registration of E911 as Publisher**

**Request message:**

```
<infod:CreatePublisherEntry>
  <infod:WSReference>
    <wsa:address>http://www.sntestbed3.ornl.gov/911services</wsa:address>
  </infod:WSReference>
  <infod:PublisherName>
    E911 center
  </infod:PublisherName>
  <infod:PublisherDescription>
    Control center were all incidents get reported
  </infod:PublisherDescription>
  <infod:PropertyConstraint>
    for $sub in fn:collection($$infodsubscriber)
      where $pub//OrganizationParent/OrganizationName="KnoxFireDept"
      for $con in fn:collection($$infodconsumers)
        where $con//OrganizationName="FireStation28"
  </infod:PropertyConstraint>
  <infod:Notification>True</infod:Notification>
</infod:CreatePublisherEntry>
```

**Returned URI:**

http://www.ggf.org/INFOD/publisher/5687A2740287245D87763FF75D32A4F

## b) Registration of Fire Service as a Consumer

Request message:

```
<infod:CreateConsumerEntry>
  <infod:WSReference>
    <wsa:address>http://www.sntestbed3.ornl.gov/Fireservices</wsa:address>
  </infod:WSReference>
  <infod:WSReference>
  </infod:WSReference>
  <infod:ConsumerName>Knox Fire Station</infod:ConsumerName>
  <infod:ConsumerDescription>Fire Service</infod:ConsumerDescription>
  <infod:PropertyConstraint>
    for $pub in fn:collection($$infodpublisher)
      where $pub//Location/LocationCounty="CountyA" and
            $pub//Location/LocationCounty="CountyC" and
            $pub//Location/LocationCounty="CountyX"
  </infod:PropertyConstraint>
  <infod:Notification>True</infod:Notification>
</infod:CreateConsumerEntry>
```

Returned URI:

<http://www.ggf.org/INFOD/consumer/5687A2740EA7245D87763FD05D32A4F>

## c) Consumer creates a Property Vocabulary Instance

Request message:

```
<infod:CreatePropertyVocabularyInstance>
  <infod:EntryReference>
    <wsa:Address>
      http://www.ggf.org/INFOD/consumer/5687A2740EA7245D87763FD05D32A4F
    </wsa:Address>
  </infod:EntryReference>
  <infod:PropertyVocabularyReference>
    <wsa:Address>
      http://www.ggf.org/INFOD/PropVocab/5687A2740287230587763FD05D32A4F
    </wsa:Address>
  </infod:PropertyVocabularyReference>
  <infod:PropertyVocabularyInstanceBody>
  <?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"
    xmlns:niem="http://www.niem.gov/niem">
    <Identification>
      <OrganizationName>FireStation29</OrganizationName>
      <OrganizationID>H-30330-56</OrganizationID>
    </Identification>
    <Location>
      <LocationAddress>
        <Street>Avenue</Street>
        <Number>10945</Number>
        <County>ABS</County>
        <Zip>98643</Zip>
        <City>London</City>
      </LocationAddress>
      <LocationCoordinates>
        <Latitude></Latitude>
        <Longitude></Longitude>
      </LocationCoordinates>
    </Location>
    <ContactInformation>
      <ContactPersonName>bob</ContactPersonName>
      <ContactPersonContactNumber>
```

```
<AreaCode>888</AreaCode>
  <PhoneNumber>789-9879</PhoneNumber>
</ContactPersonContactNumber>
</ContactInformation>
</xsd:schema>
</infod:VocabularyInstanceVocabularyBody>
</infod:PropertyVocabularyInstanceBody>
```

Returned URI:

<http://www.ggf.org/INFOD/propVocabInst/5687A2740E47245D87763FD05D32A4F>

#### d) The Publisher creates a Data Source Entry

Request message:

```
<infod:DataSourceEntry>
  <infod:DatSourceEntryName>
    EventDescriptionSource
  </infod:DataSourceEntryName>
  <infod:DataSourceEntryDescription>
    Emergency Event Description Source
  </infod:DataSourceEntryDescription>
  <infod:PublisherEntryReference>
    <wsa:Address>
      http://www.ggf.org/INFOD/publisher/5687A2740287245D87763FF75D32A4F
    </wsa:Address>
  </infod:PublisherEntryReference>
  <infod:DataVocabularyReference>
    <wsa:Address>
      http://www.ggf.org/INFOD/DataVocab/5687A2740287245D87763FD05D32A4F
    </wsa:Address>
  </infod:DataVocabularyReference>
</infod:DatSourceEntry>
```

Returned URI:

<http://www.ggf.org/INFOD/datasourceentry/5687A2740788245D87763FD05D32A4F>

#### e) E911 Center registers a Subscriber

Request message:

```
<infod:CreateSubscriberEntry>
  <infod:WSReference>
    <wsa:address>http://www.sntestbed3.ornl.gov/911services</wsa:address>
  </infod:WSReference>
  <infod:SubscriberName>E911 CENTER</infod:SubscriberName>
  <infod:SubscriberDescription>
    Decides on policies for all emergency events.
  </infod:SubscriberDescription>
  <infod:Notification>True</infod:Notification>
</infod:CreateSubscriberEntry>
```

Returned URI:

<http://www.ggf.org/INFOD/subscriber/5697A2740287245D87763FD05D32A4E>

### Step 3: E911 Center Creates Subscription

Request message:

```
<infod:CreateSubscription>
  <infod:SubscriptionName>
    Rule1
```

```

</infod:SubscriptionName>
<infod:SubscriptionDescription>
  Rules for fire service and police services
</infod:SubscriptionDescription>
<infod:SubscriberReference>
  <wsa:Address>
    http://www.ggf.org/INFOD/subscriber/5697A2740287245D87763FD05D32A4E
  </wsa:Address>
</infod:SubscriberReference>
<infod:PropertyConstraint>
  for $pub in fn:collection($$infodpublisher)
    where $pub//OrganizationSubUnitName="E911"
  for $con in fn:collection($$infodconsumers)
    where $con//OrganizationCategory="FirstResponders"
<infod>DataConstraint>
  declare namespace $data =
    http://www.ggf.org/INFOD/DataVocab/5687A2740287245D87763FD05D32A4F;
  let $msg1 := for $firstresponders in fn:collection($$infodconsumer)
    where $data:AlertStatus = Actual and
      $data:EventCategory = CBRNE and
      $data:EventSeverity > Moderate
    return $data,$data:Instruction=Evacuate people
  let $msg2 := for $firstresponders in fn:collection($$infodconsumer)
    where $data:capAlertStatus = Actual and
      $data:EventCategory = Fire and
      $data:EventSeverity = Moderate
    return $data:Substance
</infod>DataConstraint>
<infod:DynamicConsumerConstraint>
  for $firstresponders in fn:collection($$infodconsumer)
    where $firstresponders//SubUnitName=Police
      and fn:distance($firstresponder//Location,20)
    return $msg1
  for $firstresponders in fn:collection($$infodconsumer)
    where $firstresponders//SubUnitName=FireService
    return $msg2
</infod:DynamicConsumerConstraint>
</infod>CreateSubscription>

```

Returned URI:

<http://www.ggf.org/INFOD/subscription/5697A2740287245D87763FD05D32A4E>

#### Step 4: Notification Messages

##### a) Registry sends Notification Message to Publisher

```

<infod:PublisherNotification>
  <infod:SubscriptionReference>
    <wsa:Address>
      http://www.ggf.org/INFOD/subscription/5697A2740287245D87763FD05D32A4E
    </wsa:Address>
  </infod:SubscriptionReference>
  <infod:ConsumerEntryReference>
    <wsa:Address>
      http://www.ggf.org/INFOD/consumer/5687A2740EA7245D87763FD05D32A4F
    </wsa:Address>
  </infod:ConsumerEntryReference>
  <infod>DataConstraint>
    declare namespace $data =
      http://www.ggf.org/INFOD/DataVocab/5687A2740287245D87763FD05D32A4F;
    let $msg1 := for $firstresponders in fn:collection($$infodconsumer)
      where $data:AlertStatus = Actual and

```

```

        $data:EventCategory = CBRNE and
        $data:EventSeverity > Moderate
        return $data,$data:Instruction=Evacuate people
    let $msg2 := for $firstresponders in fn:collection($$infodconsumer)
        where $data:capAlertStatus = Actual and
            $data:EventCategory = Fire and
            $data:EventSeverity = Moderate
        return $data:Substance
</infod:DataConstraint>
    <infod:DynamicConsumerConstraint>
        for $firstresponders in fn:collection($$infodconsumer)
            where $firstresponders//SubUnitName=Police
                and fn:distance($firstresponder//Location,20)
            return $msg1
        for $firstresponders in fn:collection($$infodconsumer)
            where $firstresponders//SubUnitName=FireService
            return $msg2
    </infod:DynamicConsumerConstraint>
</infod:PublisherNotification>

```

#### b) Registry sends Notification Message to Consumer

```

<infod:ConsumerNotification>
    <infod:SubscriptionReference>
        <wsa:Address>
            http://www.ggf.org/INFOD/subscription/5697A2740287245D87763FD05D32A4E
        <\wsa:Address>
    </infod:SubscriptionReference>
    <infod:PublisherEntryReference>
        <wsa:Address>
            http://www.ggf.org/INFOD/publisher/5687A2740287245D87763FF75D32A4F
        <wsa:Address>
    </infod:PublisherEntryReference>
</infod:ConsumerNotification>

```

#### c) Registry sends Notification Message to Subscriber

```

<infod:SubscriberNotification>
    <infod:SubscriptionReference>
        <wsa:Address>
            http://www.ggf.org/INFOD/subscription/5697A2740287245D87763FD05D32A4E
        <\wsa:Address>
    </infod:SubscriptionReference>
    <infod:PublisherEntryReference>
        <wsa:Address>
            http://www.ggf.org/INFOD/publisher/5687A2740287245D87763FF75D32A4F
        <wsa:Address>
    </infod:PublisherEntryReference>
    <infod:ConsumerEntryReference>
        <wsa:Address>
            http://www.ggf.org/INFOD/consumer/5687A2740EA7245D87763FD05D32A4F
        <wsa:Address>
    </infod:ConsumerEntryReference>
</infod:SubscriberNotification>

```

#### d) Publisher sends Notification Message to Consumer

The publisher sends the notification message only when an event of interest, as specified in the constraints, has occurred.

```
<infod:Notify>
  <infod:NotificationMessage>
    <infod:SubscriptionReference>
      <wsa:Address>
        http://www.ggf.org/INFOD/subscription/5697A2740287245D87763FD05D32A4E
      <\wsa:Address>
    </infod:SubscriptionReference>
    <infod:PublisherEntryReference>
      <wsa:Address>
        http://www.ggf.org/INFOD/publisher/5687A2740287245D87763FF75D32A4F
      <wsa:Address>
    </infod:PubliherEntryReference>
    <infod:ConsumerEntryReference>
      <wsa:Address>
        http://www.ggf.org/INFOD/consumer/5687A2740EA7245D87763FD05D32A4F
      <wsa:Address>
    </infod:ConsumerEntryReference>
    <infod:Message>
      Information about the event.
    </infod:Message>
    </infod:NotificationMessage>
  </infod:Notify>
```