A Service-Oriented Framework for Controlling Invasive Species in Agriculture

Brahim Medjahed, Shamil Hadi, and William I. Grosky
Department of Computer and Information Science
University of Michigan – Dearborn, USA
E-mail: brahim@umich.edu E-mail: shadi@umich.edu
E-mail: wgrosky@umich.edu

Abstract: The Emerald Ash Borer (EAB) has killed or infested millions of ash trees in the United States. It has the potential to decimate Ash as a component of North American forests, which will have dramatic ecological and economic effects. Government agencies have joined forces to implement a long-term program to contain and eventually eradicate EAB from North America. One key requirement for the success of this program is the underlying information sharing infrastructure. In this paper, we propose a novel framework, called Sentinel, to automate the dissemination of EAB-related information among EAB partners. We propose an ontology-based service-oriented dissemination model. We introduce three (3) topic-based notification protocols: naïve push notification (NPN), centralized push notification (CPN), and distributed push notification (DPN). Finally, we present experiment results to compare the proposed protocols.

Keywords: Invasive Species – Agriculture – Emerald Ash Borer – Service-Oriented Architecture – Ontology – Notification.

Reference to this paper should be made as follows: Medjahed B., S. Hadi and W. I. Grosky (2009) 'A Service-Oriented Framework for Controlling Invasive Species in Agriculture'. *Int. J Metadata, Semantics and Ontologies*, Vol. X, No. Y, pp. aa-bc.

Biographical Notes: Brahim Medjahed is Assistant Professor in the Department of Computer and Information Science, University of Michigan – Dearborn. He received his Ph.D. in Computer Science from Virginia Tech in May 2004. His research interests include service-oriented computing, semantic Web, ontology, and workflow.

Shamil Hadi received his B.S. and M.S. both from the Department of Computer and Information Science, University of Michigan – Dearborn. His research interests include service-oriented computing and change management.

William I. Grosky is Professor and Chair of the Department of Computer and Information Science at University of Michigan - Dearborn. Prior to joining the University of Michigan in 2001, he was Professor and Chair of the Department of Computer Science at Wayne State University. Before joining Wayne State University in 1976, he was an Assistant Professor in the Department of Information and Computer Science at Georgia Tech. He received his B.S. in Mathematics from MIT in 1965, his M.S. in Applied Mathematics from Brown University in 1968, and his Ph.D. in Engineering and Applied Science from Yale University in 1971.

1 INTRODUCTION

The Emerald Ash Borer (EAB) is a shiny and invasive green beetle known for killing Ash trees in the United States (US) (Schuster 2005, Smitley 2002). It has the potential to decimate Ash as a component of North American forests, which will have dramatic ecological and economic effects. EAB has already killed or infested ten million ash trees in Michigan alone and fast spreading to adjoining states. The USDA (US Department of Agriculture) estimates that at the national level, if the EAB went unchecked, the loss to the nation could range from 20 to 60 billion dollars (Schuster 2005, Smitley 2002). The USDA Animal and Plant Health Inspection Service (APHIS), the USDA Forest Service, and the Canadian Food Inspection Agency, in cooperation with

State Departments of Agriculture and Natural Resources, have joined forces to implement a long-term program to contain and eventually eradicate EAB from North America. The plan, which is in the early stages of implementation, combines efforts at the research, prevention, detection, and legal levels. At the research level, scientists are working round the clock to understand the beetle's life cycle, find ways to detect new infestations, control EAB adults and larvae, and produce new insecticides. At the prevention level, mass awareness campaigns (e.g., publicity on TV/radio) are regularly launched to spread the word about EAB and the dangers posed by transporting firewood. At the detection level, federal, state, and local agencies promptly locate and eradicate outlier infestations (e.g., by cutting ash trees in infected areas). At the legal level, aggressive enforcement of state and federal quarantines is implemented.

Information sharing is a key requirement for the success of the aforementioned EAB containment programs. EAB partners (e.g., governments, universities, and media) are continuously publishing information (e.g., quarantine procedures and areas, need for cutting Ash trees, research results) on their Web sites about the borer. Multi-state efforts are being made to bring the latest information about the insect (e.g., http://www.emeraldashborer.info). However, the current process for disseminating EAB information is *ad hoc* and requires intensive human intervention which may hinder EAB containment plans. For example, Web sites include lists of phones numbers that can be used to report infestations. The called agencies generally process such information in a manual fashion; for instance, they need to figure out w*h*ich other EAB partners need to be notified, *how* (fax, letter, etc.), and *when*.

The dissemination of EAB-related information involves two types of participants: *producers* (send messages) and *consumers* (receive messages). The same participant may act as a producer and consumer. We identify two classes of dissemination techniques: *pull* and *push*. In the *pull* class, information is delivered by producers as a reply to specific requests submitted by consumers. This class corresponds to the well-known publish/subscribe interaction scheme defined for distributed systems (Eugster et al. 2003): consumers explicitly register their interest in receiving messages from certain producers. The *push* class (proposed in this paper) enables the sharing of information with EAB partners that did not request or subscribe to it, but would benefit from receiving it. Combining the concepts of Web service and ontology, a.k.a. semantic Web services (McIlraith et al. 2001), is at the core of push notification. In a nutshell, a Web service is an application accessible on the Web via programmatic means (Alonso et al. 2003, Papazoglou 2007, Qi et al. 2008). Web services provide support for automating interactions among EAB partners. Ontology is a shared and common understanding of a domain that can be communicated between people and application systems (Berners-Lee et al. 2001, Fensel 2003). The concept of ontology provides support for modelling the different patterns of interaction that exist between EAB partners.

In this paper, we propose a push-based framework, called Sentinel, to automate the dissemination of EAB-related information among EAB partners. A preliminary version of this paper appeared in (Medjahed and Grosky 2008). The main contributions of this paper are summarized below:

- We propose an ontology-based service-oriented model for disseminating information among EAB partners.
- We introduce three (3) topic-based dissemination protocols: naïve push notification (NPN), centralized push notification (CPN), and distributed push notification (DPN).
- We present experiment results to compare the proposed protocols.

The rest of this paper is organized as follows. In Section 2, we describe the dissemination model introduced in Sentinel. In Section 3, we propose three (3) ontology-based service-oriented protocols for EAB information dissemination. In Section 4, we describe our experimental study. In Section 5, we overview related work. We present concluding remarks in Section 6.

2 THE DISSEMINATION MODEL

In this section, we describe the dissemination model proposed in Sentinel. We first introduce the concept of EAB ontology to model interaction patterns among EAB partners. Then we illustrate the way EAB partners exchange EAB-related information through administrative services called EAB notifiers.

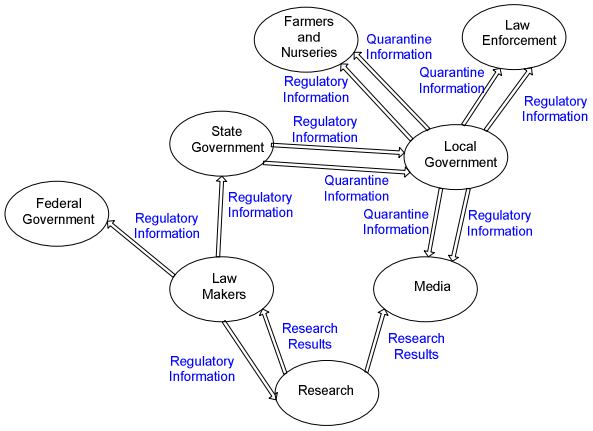
2.1 The EAB Ontology

We model the different patterns through which EAB partners exchange EAB-related information through the *EAB ontology*. The concept of ontology (Berners-Lee et al. 2001, Fensel 2003) was initially developed in Artificial Intelligence to facilitate knowledge sharing and reuse. It has since then been recognized as a popular research topic in various research communities such as knowledge engineering, e-commerce, natural language processing, cooperative information systems, and information integration. Ontology is a formal and explicit specification of a shared conceptualization. "Conceptualization" refers to an abstraction of a domain that identifies the relevant concepts in that domain. "Shared" means that the ontology captures consensual knowledge. "Explicit" means that the concepts used in the ontology and the constraints on their use are explicitly defined. "Formal" intends that the ontology should be machine understandable and described using a well-defined model or language called ontology language (such as OWL and RDF (Fensel 2003)).

At an abstract level, we model the EAB ontology as a labelled directed graph where nodes represent concepts in the application domain and labelled edges represent relationships between concepts. The graph can easily be specified using major ontology languages such as RDF and

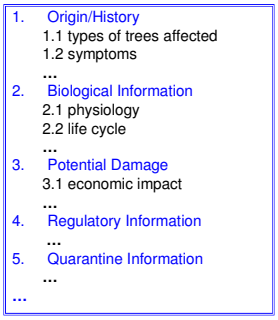
OWL. Each concept in the graph refers to an EAB partner category (i.e., domain of interest) from the application domain. An edge $C_i \rightarrow C_j$ labelled with T means that partners that belong to category C_i share information of topic T with partners of category C_j. Figure 1 depicts part of the EAB ontology. It states that:

Figure 1 Part of the EAB Ontology.



- Law makers send regulatory information to research institutions and state/federal governments. Each state government forwards that information to local governments. Local governments then notify farmers and nurseries, law enforcements, and media.
- State governments send quarantine information to local governments. These forward such information to farmers and nurseries, law enforcements, and media.
- Research institutions publish their research result with media and law makers.

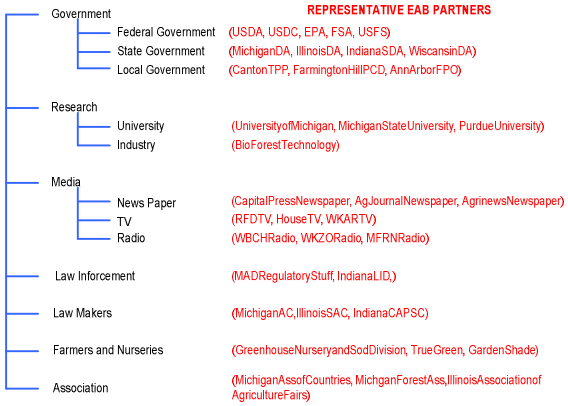
Figure 2 EAB Info Taxonomy



The categories (i.e., nodes) and topics (edges) of the EAB ontology are defined according to two taxonomies: EAB Info and EAB Partners. The EAB Info taxonomy gives the various topics of messages that may be exchanged among EAB partners. Examples of topics include "Origin/History", "Biological Information", "Potential Damage", "Regulatory Information", and "Quarantine Information" (Figure 2). An EAB message M is defined by the couple (T,D) where T is a topic from the EAB Info Taxonomy and D is the actual data to be sent. Topics may be recursively organized into subtopics. For instance, "Origin/History" has subtopics "types of trees affected" and "symptoms". We use the notation Subtopics(T) to refer to all subtopics under T (children and descendents) within the EAB Info taxonomy.

The EAB Partners taxonomy gives the categories of partners that may need to exchange EAB messages (Figure 3). An EAB partner may belong to more than one category. Examples of categories include "government", "research", "media", "law enforcement", "law makers", "farmers and nurseries", and "association". A category may be recursively organized into subcategories. For example, "government" includes three subcategories: "federal government", "state government", and "local government". We use the function Subcategories(C) to refer to all subcategories under C (children and descendents).

Figure 3 EAB Partners Taxonomy

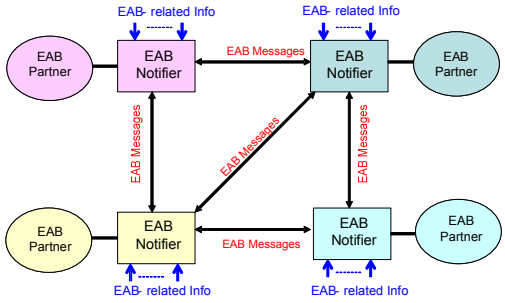


2.2 EAB Notifiers

One important feature of Sentinel is the automatic interaction among EAB partners to share EAB-related information. We propose a *service-oriented architecture* (SOA) to enable such interactions. SOAs utilize services as the building blocks for developing software systems distributed within and across organizations (Alonso et al. 2003, Qi et al. 2008). The most common realization of SOAs is based on Web services. Web services are network-accessible entities that provide pre-defined capabilities via the exchange of messages. Two factors are promoting Web services as the technology of choice for enabling cross-organizational interactions: (i) the use of standard technologies and (ii) the loose coupling of cross-organizational services. Three key XML-based standards have been defined to support Web services (Papazoglou 2007, Qi et al. 2008): SOAP, WSDL, and UDDI. SOAP defines a communication protocol for Web services. WSDL enables providers to describe the interface (e.g., operations, messages) of their services. UDDI offers a registry service that allows the advertisement and discovery of Web services.

In Sentinel, each EAB partner has a Web service, called *EAB notifier*, associated to it (Figure 4). All EAB notifiers are registered in the service registry (UDDI in our case) under the category "Emerald Ash Borer". Notifiers are further categorized according to the type of partner they are associate to (as defined in the EAB Partners taxonomy). Providing notifiers as Web services has two major advantages. First, it concords with recent SOA trends (e.g., Open Grid Services Architecture (Foster and Kesselman 2004)) which define management tasks as Web services. Second, EAB partners can join the system by merely downloading the trusted codes of EAB notifiers, deploying them as Web services, and registering them in UDDI.

Figure 4 Peer-to-Peer Topology of EAB Notifiers



Notifiers define a peer-to-peer network for sharing information among EAB partners. Each partner publishes EAB-related information via its associated notifier. The notifier formats the published information into an EAB message and sends it to other peers using one of the protocols described in Section 3. At the reception of an EAB message, a notifier applies the same protocol to forward the message to other notifiers. The WSDL document of each notifier WS_i includes two operations: Publish() and Notify(). Publish() allows partner-to-notifier interactions. It is invoked by the EAB partner corresponding to WS_i to disseminate an EAB message. We say that WS_i is the *root notifier* of that message. Notify() allows notifier-to-notifier interactions. It enables the exchange of EAB messages among notifiers.

3 DISSEMINATION PROTOCOLS IN SENTINEL

In this section, we propose three push-based dissemination protocols: naïve push notification (NPN), centralized push notification (CPN), and distributed push notification (DPN).

3.1 Naïve Push Notification (NPN)

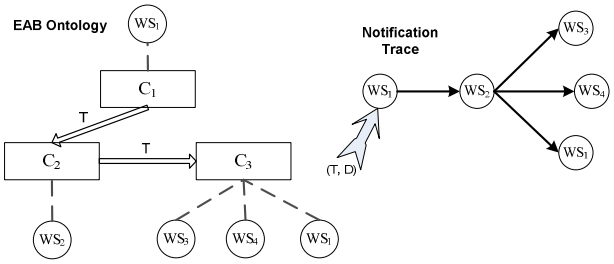
Figure 5 depicts the NPN algorithm executed by a notifier WS_i. At the invocation of Publish(T,D) or Notifiy(T,D), WS_i calls a utility procedure to process the message (line 2). The way this procedure is defined depends on the internal business logic of the EAB partner and is out of the scope of this paper. WS_i then identifies the services that need to be notified about M(T,D) (lines 3-10). For that purpose, WS_i first gets its category C_p; this is done by accessing WS_i entry in UDDI. WS_i then determines all edges $C_p \rightarrow C_q$ (from the EAB ontology graph) labelled with T or a subtopic of T. The reason behind including subtopics is the following: if a service is interested in a topic T, then it is interested in any subtopic of T. WS_i finally sends M(T,D) to each notifier WS_k that belongs to category C_q. M(T,D) is also forwarded to services that belong to subcategories of C_q.

Figure 5 NPN Protocol Executed by EAB Notifiers

```
(1) At Invocation of Publish(T,D) or Notify(T, D)
(2)  Process(T,D)
(3)  Determine Category Cp of WSi
(4)  τ = {T} ∪ Subtopics(T)
(5)  For each link Cp → Cq labeled with a topic from τ  Do
(6)        Ϲ = {Cq} ∪ Subcategories(Cq)
(7)        For each WSk of a category from Ϲ Do
(8)              Invoke Notify(T,D) of WSk
(9)        End For
(10)  End For
(11) End
```

The NPN protocol has two major drawbacks. First, a message may be sent indefinitely in the system (notification loops). Second, a message may be received repeatedly by a notifier (double notifications). To illustrate these drawbacks, let us consider the scenario depicted in Figure 6. WS1 is registered under C_1 and C_3 categories; WS_2 is registered under C_2; WS_3 and WS_4 are registered under C_3. Let us assume that M(T,D) is published to WS_1. Based on the EAB ontology graph, WS_1 (under C_1) sends M(T,D) to WS_2 (under C_2). Then, WS_2 send M(T,D) to services under C_3 (i.e., WS_3, WS_4, and WS_1). WS_1 will again forward M(T,D) to WS_2, hence creating an infinite notification loop. In the rest of this section, we propose two protocols that deal with notification loops and double notifications.

Figure 6 NPN Protocol - Example



3.2 Centralized Push Notification (CPN)

The CPN protocol uses a centralized history to keep track of all messages along with their consumers and producers. The use of a centralized history ensures that each notifier receives a message at most once. This allows avoiding double notifications and notification loops.

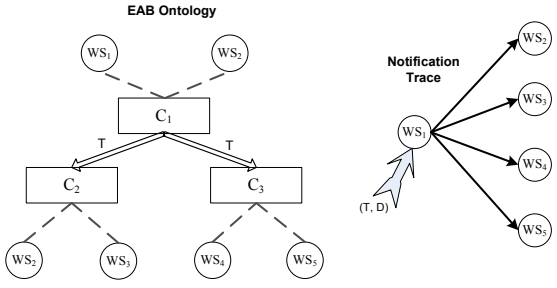
Figure 7 depicts the CPN algorithm executed by a notifier WS_i. At the invocation of Publish(T,D), WS_i (the root notifier) processes the message (line 2) and generates a unique message ID (lines 3-6). We use UUIDs (Universal

Unique Identifier) as message IDs; UUIDs are 128-bit numbers used to uniquely identify an object or entity on the Internet. IDs are carried by messages throughout the notification process. Since WS_i is the producer of M(ID,T,D), it stores (ID,*, WS_i) in the history. WS_i then determines all services that need to be notified about M(ID,T,D) (lines 7-19). The same statements (lines 7-19) are executed by WS_i at the invocation of Notify(ID,T,D). WS_i first gets all edges $C_p \rightarrow C_q$ labelled with T or a subtopic of T (C_p is the category of WS_i) from the EAB ontology graph. WS_i then determines the list of all services with C_q or a subcategory of C_q as a category. WS_i forwards M(ID,T,D) to WS_k (by calling the operation WS_k.Notify(ID,T,D)) only if WS_k has not received this message yet (lines 12-17); this is done by checking that neither (ID,*,WS_k) nor (ID,WS_k,*) belongs to the history. WS_i finally inserts (ID,WS_k,*) in the history.

Figure 7 CPN Protocol Executed by EAB Notifiers

```
(1) At Invocation of Publish(T,D) or Notify(ID, T, D)
(2)   Process(ID,T,D)
(3)   If the operation invoked is Publish(T,D)
(4)       Then Generate a unique message ID
(5)            Insert (ID,*,WSi) in History
(6)   End If
(7)   Determine Category Cp of WSi
(8)   τ = {T} ∪ Subtopics(T)
(9)       For each link Cp → Cq labeled with a topic from τ Do
(10)          C = {Cq} ∪ Subcategories(Cq)
(11)          For each WSk of a category from C Do
(12)              If (ID,WSk,*) ∈ History ∨ (ID,*,WSk) ∈ History
(13)               Then Ignore WSk
(14)                 Else
(15)                     Insert (ID,WSk,*) in History
(16)                     Invoke Notify(ID,T,D) of WSk
(17)              End If
(18)          End For
(19)      End For
(20) End
```

Figure 8 CPN Protocol - Example



History

Message ID	Consumer	Producer
	*	WS_1
1	WS_2	*
1	WS_3	*
1	WS_4	*
1	WS_5	*

To illustrate CPN protocol, let us consider the scenario depicted in Figure 8. WS_1 is registered under C_1 category; WS_2 is registered under C_1 and C_2; WS_3 is registered under C_2; WS_4 and WS_5 are registered under C_3. Let us assume that the information (T,D) is published to WS_1. WS_1 generates a unique message ID equals to 1 and inserts (1,*,WS_1) in the history. Based on the EAB ontology, WS_1 figures out that WS_1, WS_2, WS_3, WS_4, and WS_5 are candidates to receive M(1,T,D). However, the history shows that WS_1 has already received the message as a producer. Hence, WS_1 sends M to WS_2, WS_3, WS_4, and WS_5 only. It also inserts (1,WS_2,*), (1,WS_3,*), (1,WS_4,*), and (1,WS_5,*) in the History.

3.3 Distributed Push Notification (DPN)

The use of a centralized history in CPN suffers from two major drawbacks. First the history table node constitutes a single point of failure. Second, each notifier needs to remotely access the history table; this may increase the time required to send notifications. To address these problems, we propose a distributed version of NPN (Figure 9). Each notifier WS_i maintains a local history called $History_i$. $History_i$ contains the IDs of all messages received by WS_i (as a producer or consumer). At the invocation of Notify(ID,T,D), WS_i checks if ID belongs to $History_i$. If so, WS_i ignores M(ID,T,D). Otherwise, WS_i inserts ID in $History_i$ and forwards M(ID,T,D) using the same algorithm as NPN (lines 10-17).

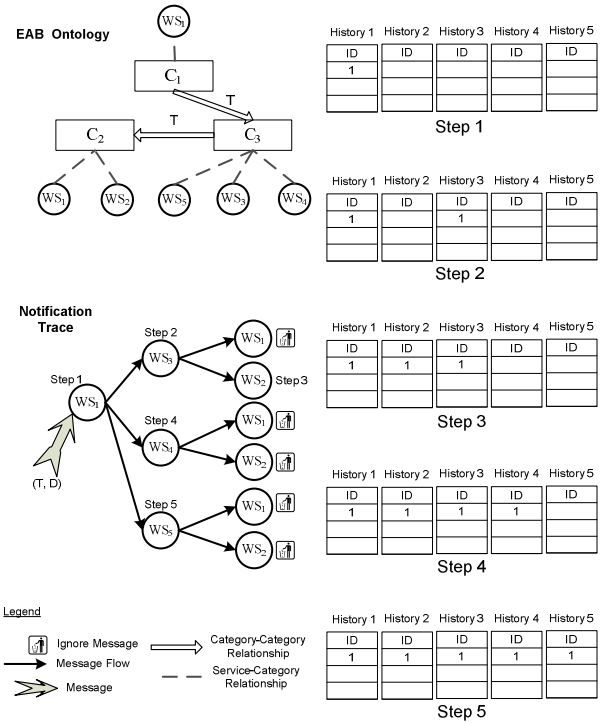
Figure 9 DPN Protocol Executed by EAB Notifiers

```
(1) At Invocation of Publish(T,D) or Notify(ID, T, D)
(2)   Process(ID,T,D)
(3)   If the operation invoked is Publish(T,D)
(4)       Then Generate a unique message ID
(6)   End If
(7)   If ID ∈ Historyi
(8)      Then ignore M(ID, T, D)
(9)      Else Insert(ID) into Historyi
(10)          Determine Category Cp of WSi
(11)          τ = {T} ∪ Subtopics(T)
(12)          For each link Cp → Cq labeled with a topic from τ Do
(13)              C = {Cq} ∪ Subcategories(Cq)
(14)              For each WSk of a category from C Do
(15)                  Invoke Notify(ID,T,D) of WSk
(16)              EndFor
(17)          EndFor
(18)   EndIf
(19) End
```

To illustrate DPN protocol, let us consider the scenario depicted in Figure 10. This scenario shows that repeated notifications are still possible in DPN. However, such repetitions are detected by notifiers; this allows avoiding notification loops. Since WS_1 is the first to receive the information defined by T and D, it generated a unique ID (say 1), inserts ID in $History_1$ and forwards M(1, T, D) to services that belong to C_1 (i.e., WS_1) and C_3 (i.e., WS_3, WS_4, and WS_5). At the reception of M, WS_3 inserts ID into $History_3$ since ID $\notin$ $History_3$. Then, it forwards M to services that belong to C_2 (i.e., WS_1 and WS_2). Since ID $\in$

History$_1$, WS$_1$ considers M as a repeated notification and hence ignores the message. Since ID ∉ History$_2$, WS$_2$ inserts ID into History$_2$. At the reception of M, WS$_4$ inserts ID into History$_4$ since ID ∉ History$_4$. Then, it forwards M to services that belong to C$_2$ (i.e., WS$_1$ and WS$_2$). Since ID ∈ History$_1$ and ID ∈ History$_2$, WS$_1$ and WS$_2$ ignore M. At the reception of M, WS$_5$ forwards M to WS$_1$ and WS$_2$. Similarly, the message is ignored by both WS$_1$ and WS$_2$.

Figure 10 DPN Protocol - Example



4 EXPERIMENTS

We conducted experiments to compare the performance of CPN and DPN protocols. We used Microsoft Windows Server 2003 (operating system), Microsoft Visual Studio 8 (development kit), UDDI Server, IIS Server, and SQL Server (for history tables in CPN and DPN). We defined the EAB Partners and Info taxonomies as UDDI categorizations. We deployed twenty (20) representative EAB notifiers (implemented in C#) and registered them in UDDI under the corresponding categories (see Fig 3 for the list of EAB partners). All notifiers are deployed in the same machine (Intel(R) processor, 1500MHz, and 512MB of RAM). The centralized history is stored in a separate machine within our local area network to simulate delays in accessing a remotely located history. We considered both synchronous and asynchronous service invocations. We designed ten (10) scenarios and ran each scenario using CPN synchronous, CPN asynchronous, DPN synchronous, and DPN asynchronous.

Figure 11 Notification Time for CPN and DPN

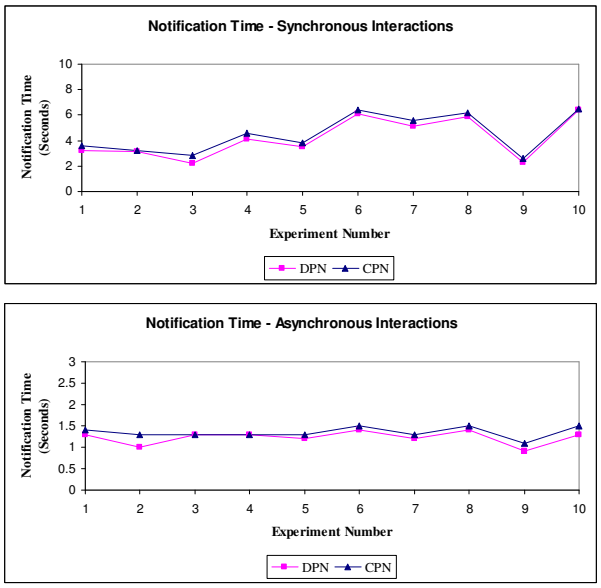


Figure 11 (top) compares the global notification time (in seconds) of CPN and DPN synchronous. Notification time is bigger in the case of CPN since all notifiers need to access a remotely located history table. In DPN, each notifier accesses a local history table. Figure 11 (bottom) compares the global notification time of CPN and DPN asynchronous. As in the synchronous case, DPN is faster than CPN. It is interesting to note that notification time is smaller in the case of asynchronous invocations. The reason is that notifications are performed in parallel in asynchronous invocations while they are performed sequentially in synchronous invocations.

Figure 12 Number of Notifications for CPN and DPN

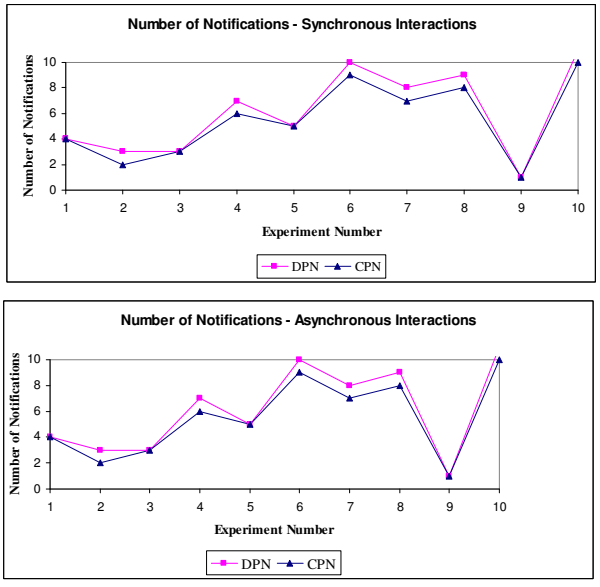


Figure 12 (top) compares the total number of notifications sent in CPN and DPN synchronous. Experiments show that DPN generates more notifications that CPN. As mentioned earlier, the use of a centralized history in CPN allows avoiding double notifications and notification loops. However, double notifications are still possible in DPN. Figure 12 (bottom) compares the total number of notifications sent in CPN and DPN asynchronous. The total number of notifications is similar to the one in the synchronous case. The interaction mode (synchronous vs. asynchronous) affects notification time not the number of notifications.

5 RELATED WORK

Publish-subscribe has received considerable attention over recent years (Bertino and Ramamritham 2004). It provides a loosely coupled form of interaction that is well suitable for large-scale distributed systems. Publish-subscribe systems are generally categorized as topic-based or content-based (Eugster et al. 2004). In topic-based systems, each event belongs to one of a fixed set of subjects (also called topics); publishers are required to label each event with a topic name. In content-based systems, events are no longer divided into different subjects. The subscriber defines a subscription condition according to the internal structure of events; all events that meet the condition will be sent to the subscriber. Our focus in this paper is on topic-based publish-subscribe in service-oriented architectures.

WS-Notification is a family of specifications that define a standard Web services approach to notification using a topic-based publish/subscribe systems (Graham 2004, Vinoski 2004). It includes a standard message exchanges to be implemented by service providers that wish to participate in Notifications, standard message exchanges for a notification broker service provider (allowing publication of messages from entities that are not themselves service providers), operational requirements expected of service providers and requestors that participate in notifications, and an XML model that describes topics. WS-Notification follows the pull notification approach (see Section 1); Consumers explicitly register their interest (via subscription) in receiving messages that belong to certain topics with potential producers. The notification approach proposed in this paper is push-based; it uses the concept of ontology to model possible notification patterns that exist among Web services. Web services may receive a notification even if they did not explicitly subscribe to it.

Web Services Resource Framework (WSRF) is a family of OASIS-published specifications that allow Web services to become stateful (Foster and Kesselman 2004). Major contributors of WSRF include the Globus Alliance and IBM. Web service clients communicate in WSRF with resource services which allow data to be stored and retrieved. When clients interact with Web services they include the identifier of the specific resource that should be used inside the request, encapsulated within the WS-Addressing endpoint reference. WSRF uses WS-Notification (i.e., pull notification) to enable information dissemination among Web services.

Several techniques have adopted the concept of ontology in publish-subscribe systems. (Wang et al. 2004a) proposes an ontology-based matching algorithm for content-based publish-subscribe systems. (Skovronski and Chiu 2006) presents a publish-subscribe system which utilizes ontology to classify and query published data. (Wang et al. 2004b) uses ontology to model events, topics, and subscriptions in content-based publish-subscribe systems. Our work is different in that it uses ontology to model notification patterns among producers and consumers.

6 CONCLUSION

In this paper, we propose an ontology-based service-oriented model for disseminating information among EAB partners. We introduce three (3) topic-based publish-subscribe protocols: naïve push notification (NPN), centralized push notification (CPN), and distributed push notification (DPN). Finally, we describe a prototype Sentinel implementation and conduct experiments to compare the proposed dissemination protocols.

Experiment results show that CPN minimizes the number of notifications. However, it increases the notification time since all notifiers access a centralized table. On the hand, DPN decreases notification time (each notifier accesses a local history) and increases the number of notifications (because of the possibility of double notifications). As future research we will explore other protocols to minimize both notification time and the number of notifications. For instance, we will investigate the use of distributed hash tables (Naor and Wieder 2003) to reduce double notifications. We will also define protocols that embed the (local) list of notified services in the SOAP headers of notification messages to minimize notification time and the number of notifications.

Sharing information among EAB partners generally triggers a number of internal decisions and actions (besides forwarding information to other partners) within each partner. For example, a government agency may notice that, during the past two major holidays, a substantial number of people have been fined for moving firewood outside a certain quarantine area. It may decide to improve its publicity campaign in that area in the coming holiday. As future work, we are planning to develop machine learning techniques to empower EAB partners with decision making rules.

Acknowledgements

The Work of Brahim Medjahed is supported by grants from the University of Michigan's OVPR (Office of the Vice-President for Research) and the University of Michigan - Dearborn's CEEP (Centre for Engineering Education and Practice).

REFERENCES

Alonso G., Casati F., Kuno H. and Machiraj V. (2003): Web Services: Concepts, Architecture, and Applications, Springer Verlag (ISBN: 3540440089).

Berners-Lee T., Hendler J. and Lassila O. (2001): The Semantic Web. Scientific American, 284(5):34-43.

Bertino E. and Ramamritham K. (2004): Guest Editors' Introduction: Data Dissemination on the Web. IEEE Internet Computing 8(3).

Eugster P. T., Felber P. A., Guerraoui R. and Kermarrec A.-M. (2003): The Many Faces of Publish/Subscribe, ACM Computing Surveys, 35(2):114–131.

Fensel D. (2003): Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce, Springer Verlag (ISBN: 3540003029).

Foster I. and Kesselman C. (2004): The Grid: Blueprint for a New Computing Infrastructure, 2nd Edition. Morgan Kaufmann (ISBN: 1-55860-933-4).

Graham S., Niblett P., Chappell D., Lewis A., Nagaratnam N., Parikh J., Patil S., Samdarshi S., Sedukhin I., Snelling D., Tuecke S., Vambenepe W. and Weihl B. (2004): Publish-Subscribe Notification for Web services, White paper.

Medjahed B. and Grosky W. I (2008): A Notification Infrastructure for Semantic Agricultural Web Services. In Metadata and Semantics, M. A. Sicilia and M. D. Lytras (Eds),, Springer-Verlag.

McIlraith S. A., Son T. C. and Zeng H. (2001): Semantic Web Services. IEEE Intelligent Systems, 16(2).

Naor M. and Wieder U. (2003): A Simple Fault Tolerant Distributed Hash Table, Second International Workshop on Peer-To-Peer Systems.

Papazoglou M. P. (2007): Web Services: Principles and Technology, Prentice Hall (ISBN: 9780321155559).

Qi Y., Liu X., Bouguettaya A. and Medjahed B. (2008): Deploying Web Services on the Semantic Web. VLDB Journal, 17(3).

Schuster C. (2005): Invasive Insects. Maryland Cooperative Extension. Newsletter. 7(1):3-5.

Skovronski J. and Chiu K. (2006): Ontology Based Publish Subscribe Framework. The 8th Int. Conf. on Information Integration and Web-based Applications Services.

Smitley D. (2002): Emerald Ash Borer: Late Summer and Fall Management Strategies, Landscape Alert, Michigan State University. 8(15).

Vinoski S. (2004): Web Services Notifications. IEEE Internet Computing 8(2).

Wang J., Jin B. and Li J. (2004a): An Ontology-Based Publish/Subscribe System. Middleware 2004, ACM/IFIP/USENIX International Middleware Conference.

Wang J., Jin B., Li J. and Shao D. (2004b): A Semantic-Aware Publish/Subscribe System with RDF Patterns, COMPSAC.