Incorporating Update Semantics Within Geographical Ontologies

Xuan Gu and Richard T. Pascoe

Department of Computer Science and Software Engineering (CSSE),
University of Canterbury, Canterbury, NZ
mark.guxuan@gmail.com,
richard.pascoe@canterbury.ac.nz
http://www.cosc.canterbury.ac.nz/richard.pascoe

Abstract. In this paper is described a systematic technique by which geographical ontologies, descriptions of concepts and relationships that exist for geographical domains of interest, may incorporate *update policies*, knowledge that governs the updating of data described by these ontologies. Of particular interest are those ontologies describing distributed geographical data where different components are maintained by separate organizations. As provider organizations change their individual contributions to the distributed data set, the efficacy of a local copy of this distributed data will decline. The incorporated update policy of the associated ontology for this local copy will be used to determine when an accumulation of changes, described by *update notifications*, justifies updating the local copy. Update policies and update notifications are assumed to have a common ontological basis. Ontologies are described using the Unified Modelling Language (UML) [4] with the semantics of an update policy being expressed using a UML profile described in this paper. The intent is to implement software agents that will execute the update policy and when justified will generate a plan by which the local copy can be updated to reflect the distributed data currently available.

1 Introduction

Geographic Information Systems (GISs) are used by many organizations, governments, research institutes and other bodies for tasks such as gathering, transforming, manipulating, analyzing, and producing information related to spatial data. For example, police and fire departments may use GISs to locate landmarks and hazards, plot destinations, and design emergency routes. The tasks are often further complicated by organizations using shared or distributed data sets in their analysis to reduce costs and improve consistency across related data sets.

The environment being considered in this research is analogous to the notion of a Geospatial Information Community (GIC) initially described by McKee and Buehler [1] and refined by Bishr [2] where organizations share data sets within the context of a common ontology. In such an environment. individual

M.A. Rodríguez et al. (Eds.): GeoS 2005, LNCS 3799, pp. 211–226, 2005.

organizations using this data may also be responsible for maintaining elements of this common data to reflect changes in the real world phenomena represented. Thus, an organization may fulfill either or both of the following roles:

a data provider P_n where this organization provides others with access to at least one data set λ. For a given system, there is a collection of data sets $\Theta = \{\lambda_g\},\ 1 \leq g \leq G$ provided by N providers ($G \geq N$). In the particular environment being considered, Θ will typically be large and transmitting large parts of this data will be resource intensive and potentially slow.

a data consumer C_m where this organization at some point in time t creates a local copy $\Lambda_{p,m,t}$ of a distributed data set $\Lambda_p \subseteq \Theta_t$. The local copy $\Lambda_{p,m,t}$ is needed to satisfy accessibility and performance requirements.

The focal point of this research is the ease with which changes to Θ over time can be selectively propagated to $\Lambda_{p,m,t}$. That is, over a period of time $t \ldots t',\ t < t'$, any provider P_n may apply changes $\Delta\lambda_{g,t'}$ to their data set $\lambda_{g,t}$ to form an updated data set $\lambda_{g,t'} = \lambda_{g,t} + \Delta\lambda_{g,t'}$. In doing so, any consumer C_m of the distributed data set $\Lambda_p : \lambda_g \in \Lambda_p$ may update $\Lambda_{p,m,t}$ to form $\Lambda_{p,m,t'}$. The conditions under which C_m updates $\Lambda_{p,m,t}$ is determined by comparing a description of $\Delta\lambda_{g,t'}$, provided by P_n, with the update policy for $\Lambda_{p,m}$. The notation being used throughout this paper is summarized in Table 1.

Table 1. Summary of notation used to refer to elements of the proposed system

Symbol	Description	
P_n	Provider n	$1 \leq n \leq N$
C_m	Consumer m	$1 \leq m \leq M$
$\lambda_{g,t}$	Data set g provided by P_n at time t	
Θ_t	All data provided within the system at time t.	$\Theta_t = \{\lambda_{g,t}\}\ 1 \leq g \leq G,\ G \geq N$
$\Delta\lambda_{g,t'}$	Changes to $\lambda_{g,t}$ occurring between t and t'	$t < t'$
Λ_p	Distributed data set p	$\Lambda_p \subseteq \Theta$
$\Lambda_{p,m,t}$	The local copy of a distributed data set $\Lambda_p \subseteq \Theta$ created at some point in time t by C_m	
$\Delta\Lambda_{p,t'}$	Changes to $\Lambda_{p,t}$ occurring between t and t'	

Notes:

1. For any given point in time t:
   (a) Any data set λ contributed by P_n is uniquely identified by g;
   (b) Any distributed data set Λ is uniquely identified by p.
2. For any given two points in time $t,\ t' :\ t < t'$:
   (a) $\lambda_{g,t'} = \lambda_{g,t} + \Delta\lambda_{g,t'}$
   (b) $\Lambda_{p,t'} = \Lambda_{p,t} + \Delta\Lambda_{p,t'}$ where $\Delta\Lambda_{p,t'} = \{\Delta\lambda_{i,t'} : \lambda_i \in \Lambda_p\}$

Although the problem of synchronizing a local copy of a distributed data set could be solved by having consumers access the distributed data set directly

rather than indirectly through a local copy $\Lambda_{p,m,t}$, such an ideal situation is unlikely, because of the typically large quantities of data being shared, security and reliability concerns, cost and flexibility, and the different performance and business requirements of individual consumers.

The proposed approach uses software agents to synchronize $\Lambda_{p,m,t}$ according to the associated update policy embedded in the associated ontology. While these agents may act independently of each other, the expectation is that they will collaborate where the agents have common goals within their update policies thereby optimizing the process by which $\Lambda_{p,m,t}$ can be synchronized. The remainder of this paper contains descriptions of our progress towards achieving the following research objectives:

1. To determine what characteristics of a change in data value or structure need to be described in an update notification $\Delta\lambda_g$ to facilitate research objective 3, described below.
   To be effective, these update notifications must contain detailed descriptions of the changes so that each software agent can better evaluate the significance of $\Delta\lambda_g$ in the context of the update policy for $\Lambda_{p,m}$.
2. To provide a means by which an organization's update policy can be easily incorporated into the ontology for $\Lambda_{p,m}$. The approach adopted here involves expressing the ontology using UML that has been extended by a UML Profile for clearly expressing the semantics of an update policy.
3. To implement
   (a) agents that can individually or collectively
      i. determine the importance of a particular update notification in the context of each agent's update policy. In essence, addressing issues such as:
         A. computing $\Delta\Lambda_{p,m,t'}$ by accumulating and merging update notifications $\Delta\lambda_g$ from the providers of $\lambda_g \in \Lambda_p$.
         B. evaluating $\Delta\Lambda_{p,m,t'}$ within the context of the associated update policy to determine whether the difference is significant thereby justifying the update of $\Lambda_{p,m,t}$.
      ii. create an *update execution plan* that details the steps by which to synchronize $\Lambda_{p,m,t}$ with $\Lambda_{p,m,t'}$.
   (b) a system that will execute the update execution plan

Objectives 1 and 2 are discussed in Sections 3 and 4 respectively while some preliminary ideas for objective 3 are briefly discussed in Section **??**. The general approach adopted for this research is described next.

2 Overall Approach

Geographical ontologies are expressed as UML models [6] encoded as XML Metadata Interchange (XMI) documents [13] in a manner compatible with the notion of application schemas that conform to ISO 19109 [7]. This standard is one of

many being defined by Technical Committee 211 (TC211) of the International Organization for Standardization to facilitate the interoperability of GISs.

The intent is to draw upon the many ISO TC211 standards and the associated harmonized UML model underlying these standards as the basis for geographical ontologies that are analogous to the notion of a Platform Independent Model (PIM) within the Model Driven Architecture (MDA) defined by the Object Management Group (OMG) [8].

At least one such PIM or ontology is expected to be defined for the entire distributed data set Θ available to organizations participating by either providing components of the distributed data set λ_g, consuming components of this data set as a local copy $\Lambda_{p,m,t}$, or both. Each organization will elaborate upon the distributed PIM or ontology to form another that is also platform independent, but which is restricted to only those components of the PIM that are of interest to the organization. This restricted ontology, for the local copy of the relevant components of the distributed data set, is further elaborated to form a Platform Specific Model (PSM) that introduces platform specific semantics reflecting the particular storage techniques being employed. While there is likely to be only one PIM describing the entire distributed data set, there will be at least one pair of restricted PIM and corresponding PSM for each organization.

Use of the MDA in this way clearly distinguishes between heterogeneity arising from conceptual differences in the way each organization views the shared geographical data and heterogeneity arising from the each organization using different implementation specific technologies for managing and processing this data. This distinction facilitates the development of flexible, scalable, loosely coupled systems. Furthermore, use of the MDA and the UML for expressing ontology partially addresses the problem that F. Fonseca mentioned in his paper about the gap between ontologies and the software components [5].

Seth and Larson's notion of a five-level schema framework for distributed systems [9] is combined with the OMG's notion of PIMs and PSMs elements of the MDA to form the four-level ontology framework shown in Figure 1. Within this framework, an ontology may fulfill one of five roles:

local PSM In this role, an ontology describes an organization's geographic domain of interest and reflects the specific platform managing the data. Each organization is likely to have quite different ontologies at this level of abstraction.

local PIM In this role, an ontology also reflects an organization's geographic domain of interest but in a platform neutral manner ideally using appropriate elements of the ISO 19100 harmonized model. In this role, an ontology is analogous to what Sheth and Larson [9] refer to as a component schema.

export PIM In this role, an ontology defines those parts of the local PIM for which each organization m is contributing data (λ_g) to form the distributed data set Θ. Such an ontology will be a subset of the local PIM ontology.

import PIM In this role, an ontology will define the local copy of the distributed data set $\Lambda_{p,m}$ that is of interest to organization m. Such an

ontology may be a (part of a) of an ontology generically referred to as distributed PIM.

distributed PIM In this role, an ontology will define data being shared by participating organizations.

When fulfilling any of the export, import, and distributed, PIM roles, the ontology is expressed using a vocabulary common to all organizations and consistently uses elements of the ISO TC211 harmonized model. An ontology fulfilling one of these three roles corresponds to Sheth and Larson's notion of an export, external, and federated schemas respectively. The export and import PIMs roles are regarded as being at the same level of abstraction and will be defined by an organization acting as a provider or a consumer respectively, while each the three remaining roles are regarded as being defined at a distinct levels of abstraction: therefore, there are four levels of abstraction in the proposed framework.

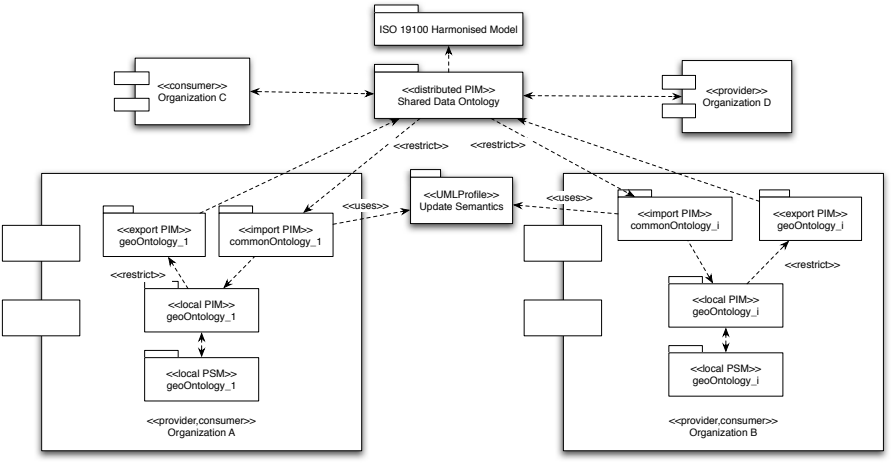


Fig. 1. Proposed four-level ontology framework where UML component symbols in the diagram represent organizations acting in the role of provider, consumer or both as indicated by the assigned stereotypes; UML package symbols are used in the diagram to depict ontologies with each being stereotyped to indicate the role the ontology fulfills in the framework

Of particular interest to this research is the import PIM ontology describing $\Lambda_{p,m}$ since this ontology is to incorporate the semantics of the organization's desired update policy. These semantics are to be incorporated by applying a UML Profile containing extensions (stereotypes, tagged values and constraints) [4–page 711] to the UML that will allow the individual characteristics of the organization's update policy to be specified consistently across all organizations.

An organization's local copy, Λ_p, of Θ conform to an ontology in the role of an import PIM. This ontology is expressed using UML that has been extended

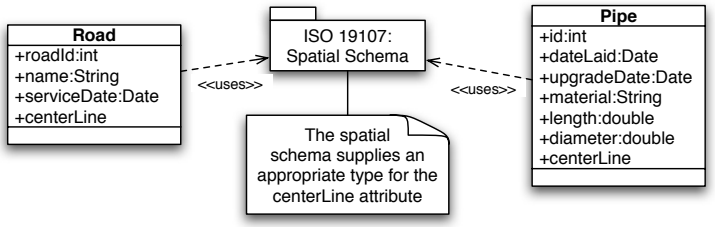


Fig. 2. The ontology for $\Lambda_{1,3,t}$. This ontology fulfills the role of an import PIM.

by the UML Profile for update semantics described in Section 4.1 thereby incorporating an update policy reflecting the organization's business priorities for data synchronization.

Once an update policy has been defined by organization m, execution of this policy involves analyzing the update notifications, $\Delta\lambda_g$, either broadcast by, or requested from, the various organizations contributing components of $\Lambda_{p,m}$ to determine when, within the context of the update policy, a significant difference exists between the local copy $\Lambda_{p,m,t}$ and $\Lambda p, m, t'$. When such a significant difference exists, an update execution plan is formulated to govern the steps by which $\Lambda_{p,m,t}$ is updated. The proposed system is viewed as an Ontology Driven Information System [3] since an ontology with the embedded update policy "plays a central role in the system's lifecycle" [*op cite*, page 16].

To illustrate the system in practice, a use case is now described to demonstrate update notifications and policies in the envisaged distributed environment.

Two local government authorities, P_1 and P_2, each provide a data set: P_1 provides λ_1 containing information about roads; and P_2 provides λ_2, containing information about underground water pipes. At time t a consumer organization, C_3, consumes both P_1 and P_2 to form a local data set $\Lambda_{1,3,t}$ which provides mappings between roads and water pipes based on their geospatial locations.

The concepts of update notifications and update policies are examined in more detail in Sections 3 and 4 respectively. In each, scenarios associated with the above use case are given to further illustrate these concepts.

3 Update Notifications

An update notification $\Delta\lambda_{g,t'}$ is a message from a data provider P_n to any consumer C_i that describes changes to data set $\lambda_{g,t'}$. In simple terms the following tasks are of interest:

- when the data is changed, create a description of this change (Sections 3.1); and
- provide consumers with access to this description (Section 3.2).

To illustrate the concept of an update notification consider the following scenarios within the context of the use case described earlier.

Scenario 1

Organization P_1 modifies λ_1 to:
- introduce two new road centre lines as a consequence of a new subdivision. This introduction also requires an existing road centre line to be altered;
- modify an existing centre line for a segment of road that has been realigned to remove a bend that was causing serious traffic accidents.

By prior arrangement, P_1 immediately sends $\Delta\lambda_1$, a description of these changes, to C_3.

Scenario 2

Organization P_2 modifies λ_2 to:
- alter the export PIM ontology defining λ_2 by deleting the attribute called 'diameter' and introducing a new attribute called 'comment';
- modifying the name of an existing attribute called 'id' to become 'pipeId'.

C_3 retrieves $\Delta\lambda_{2,t'}$ and $\Delta\lambda_{1,t'}$ from each provider's 'Blackboard' (see Section 3.2) as part of the periodic review process in place for updating $\Lambda_{1,3,t}$.

In each scenario, the description of the changes form the content of an update notification and only document characteristics that will be needed to determine whether the changes are significant in terms of the various update policies defined for the relevant distributed data sets. Inaccurate or incomplete update notifications will lead to poor decisions about when to synchronize the relevant Λ_p. Furthermore, subsequent planning and execution of the updates to $\Lambda_{p,t}$ may be inhibited because consumers lack sufficient information to determine what they need to update and how to retrieve and possibly transform the changed data to form $\Lambda_{p,t'}$.

In the case of Scenario 1, content of the update notification would include, for example: the spatial bounding box for each of the new roads; and the identifiers of existing roads that have modified values. The specific details (such as the spatial location of the changed road centre lines) are supplied when the update execution plan is being implemented. Further explanation of the content of update notifications is provided in Section 3.1.

As illustrated by the two Scenarios, update notifications are available either directly by sending the description to one or more consumers (Scenario 1), or indirectly by posting the description in a storage location that all consumers can access when they are interested (Scenario 2). Further explanation of the content of update notifications is provided in Section 3.2.

3.1 Content of Update Notifications

Update notifications describe changes to a particular data set. As shown in Figure 3, information contained within the notification are grouped into those

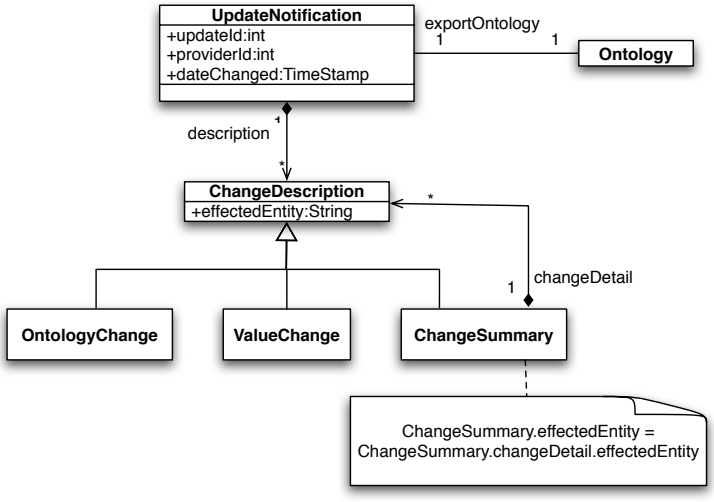


Fig. 3. UML Class diagram of Update Notifications

elements concerned with administrating the notifications themselves and those concerned with describing the changes. Administrative elements of a notification are:

Provider ID which is a unique identifier for each provider within the system so that the consumer knows the source of the notification.

Update ID is specified in sequence by data provider, and is unique to each update generated by this provider. This ID can also be treated as data set version number.

Export PIM Ontology describes provider data set in order to aid the consumer to further identify the provider and subscribed data sets.

Date Changed specifies the date and time when these changes were applied to the data.

Each change description describes one modification to an entity. This description is focussed on either

a value change, described by a statistic that summarizes the changes to values of this class of entity within $\Delta\lambda_{g,t'}$. A value change may involve the insertion, deletion, or modification of a value as illustrated by Scenario 1; or

an ontological change involving the insertion, deletion, or modification of elements (classes, attributes, associations) of the ontology as illustrated by Scenario 2.

More than one kind of change affecting the same entity is aggregated by a ChangeSummary description.

Describing Value Updates. A value update notification describes the content of $\Delta\lambda_{g,t'}$, the changed values since the last update notification at time t. This description is a list of statistics that are calculated for $\Delta\lambda_{g,t'}$ and only these statistics are included in the update notification. Examples of statistics that may be included are the number of insertions, deletions, or modifications, for different classes of entity, and such like. By transmitting only the statistics of the changed values, the transmission of a large amount of data across the network may be avoided if consumers decide to ignore such changes.

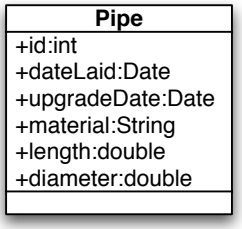
Describing Ontology Updates. Insertions, deletions and modifications to an existing ontology are expressed as an XMI document [13– page 1-31]. All differences described by XMI document must be applied in order so that the model integrity can be maintained. Thus, when a consumer receives an ontology update notification, the consumer can either ignore and discard that notification or address all differences described by the notification and any earlier notifications that were ignored.

Ontology updates may have serious consequences and need to be considered with care. Changes to the ontology correspond to changes to the database, which in turn may impact upon existing database processing software developed by each organization. One strategy being considered involves providers intending to make an ontological change sending consumers an ontological update proposal. This proposal is analyzed by each data consumer in order to assess the desirability of the proposed ontological update. For example, adapting an software application to the attribute name change would be much easier than adapting the same application to the changes that involve attribute deletion. After analysing the update proposal, each consumer sends feedback to the provider indicating whether such a change is desirable. The provider evaluates this feedback while deciding whether to make the ontological change.

3.2 Notifying Consumers of Updates

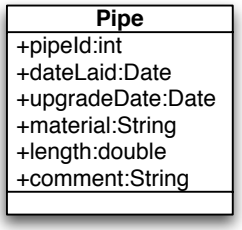
A local copy is updated when the consumer becomes aware that this data differs significantly from that which is available from the provider(s) as they change the available data to reflect modifications to the relevant real world phenomena. Consumer awareness typically occurs in one of two ways: either a consumer periodically checks with the provider(s) for updates, or provider(s) broadcasts update notifications to all relevant consumers whenever the available data has been updated. These two methods are referred to as *Pull* and *Push* respectively with both methods having advantages and disadvantages.

The Push method allows consumer to receive update notifications as soon as they are published by the providers. As the number of consumers increases, however, this method is likely to become impractical. Software such as Microsoft Windows and Norton Antivirus are used by a great many people around the world and having the providers of this software broadcast update notifications to every consumer may be impractical. The situation is exacerbated by providers




```
<xmi:XMI version="2.0">
  ...
  <UML:Class xmi.id="S.11" name="Pipe">
    <UML:Attribute xmi.id="S.19" name="id">
      ...
    </UML:Attribute>
    <UML:Attribute xmi.id="S.22" name="diameter">
      ...
    </UML:Attribute>
  </UML:Class>
  ...
</xmi:XMI>
```

(a) oldPipe.xmi




```
<xmi:XMI version="2.0" ...>
  ...
  <UML:Class xmi.id="S.11" name="Pipe" ...>
    <UML:Attribute xmi.id="S.19" name="pipeID">
      ...
    </UML:Attribute>
    <UML:Attribute xmi.id="S.25" name="comment">
      ...
    </UML:Attribute>
  </UML:Class>
  ...
</xmi:XMI>
```

(b) newPipe.xmi

```
<xmi:XMI version="2.0" xmlns:UML="org.omg/UML"
                       xmlns:xmi="http://www.omg.org/XMI">
    <!-- deletion of "diameter" attribute -->
    <difference xmi:type="xmi:Delete">
        <target href="oldPipe.xmi#S.22"/>
    </difference>

    <!-- addition of "comment" attribute -->
    <difference xmi:type="xmi:Add" addition="S.25">
        <target href="oldPipe.xmi#S.11"/>
    </difference>
    <UML:Attribute xmi.id="S.25" name="comment" ...>
        ...
    </UML:Attribute>

    <!-- change of "id" attribute -->
    <difference xmi:type="xmi:Replace" replacement="S.19">
        <target href="oldPipe.xmi#S.11"/>
    </difference>
    <UML:Attribute xmi.id="S.19" name="pipeID" ...>
        ...
    </UML:Attribute>
</xmi:XMI>
```

(c) diffPipe.xmi

Fig. 4. An example of an ontological update. The initial ontology for a pipe feature (a) is modified to become (b) by the following operations listed in (c): 1. the attribute 'diameter' is removed; 2. the attribute 'comment' is added; 3. the name of the attribute 'id' has been updated to 'pipeID'. Note that for illustrative purposes only the relevant XMI is shown.

who frequently publish update notifications. The Pull method solves these problems, since this method allows each consumer to configure a strategy of checking for update notifications which reflects the organization's unique business rules for data synchronization. However, the drawback to the Pull method is that consumers may not get critical updates in a timely fashion.

In this paper, both Pull and Push methods are considered, and the proposed system architecture allows both to work cooperatively to achieve the best result. Those consumers with low to moderate demands for up to date data may use a Pull method while organizations with high demands will be prepared to pay providers to be included in a Push method of receiving notifications.

Even though update notifications may be broadcasted to all subscribers, these updates might not be executed or might even be discarded because of the update policies (see Section 4) defined at the consumer end. In order for these consumers to retrieve for non-updated updates in a later stage, all update notifications need to be stored and maintained by both data providers and consumers for a configurable period of time. The storage space is analogue to *Blackboard*, which allows interaction and information can be exchanged indirectly and asynchronously between different organizations. Figure 5 shows the whole process described above.

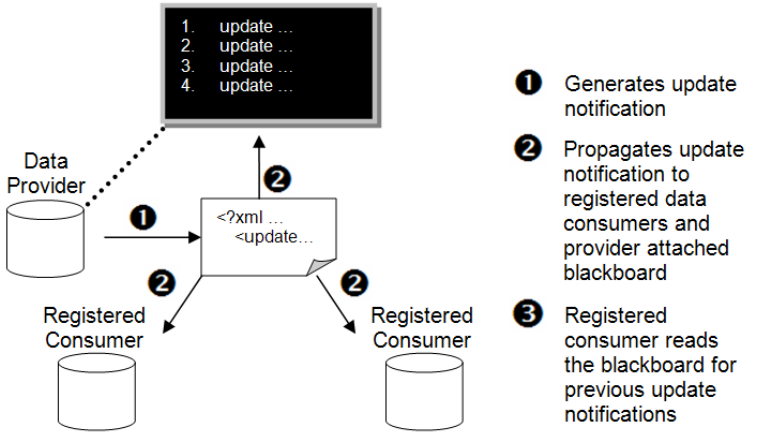


Fig. 5. Propagation of update notification

However, if more and more update notifications are displayed on the Blackboard, the storage space will eventually run out. In order to avoid this problem, a *Version control* mechanism is used to control the version of all subscribers' data sets by using the "Update ID" field in the notification message to maintain the information about which update has been successfully extracted and applied by which subscriber. Once an update has been extracted by all subscribed consumers, or has been displayed for a certain period of time, both the

update notification and $\Delta\lambda_{g,t'}$, the changed data values, will be removed from the Blackboard.

If a notification has been removed and not all consumers have executed that update, then those consumers need to either compute the changes by themselves by comparing their local copies $\Lambda_{p,m,t}$ with $\Theta_{t'}$ or simply update $\Lambda_{p,m,t}$ regardless of the quantity and nature of the changes made in the period of time between t and t'.

4 Update Policy

Consumer defined update policies consist of a series of rules that determine the circumstances under which a data consumer C_m will update its local data set $\Lambda_{p,m,t}$. These rules primarily reflect the higher level business requirements and constraints unique to each consumer.

Once update policies are defined by consumers, they are incorporated into primarily import PIM ontologies using UML extensions defined by a profile for update semantics described in Section 4.1. Providers may wish to indicate immutable elements (for example identifiers) of an export PIM ontology: therefore, such a stereotype is also provided within the profile for Update Semantics.

An update policy is used to evaluate the significance of a collection of update notifications to a particular consumer. When deemed significant, notification of these updates will initiate the updating of $\Lambda_{p,m,t}$. Informally this may be described by the following expression:

$$updatePolicyFunction(\Lambda_{p,m,t}, \Delta\Lambda_{p,m,t'}) \mapsto \{true, false\}$$

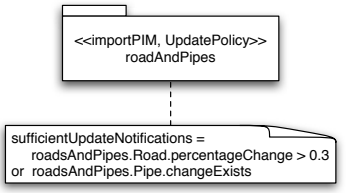
where the $updatePolicyFunction()$ comprises an OCL Expression involving tagged values defined by the UML profile for update semantics.

To illustrate the concept of an update policy, consider Scenario 3 within the context of the use case described earlier in Section 2.

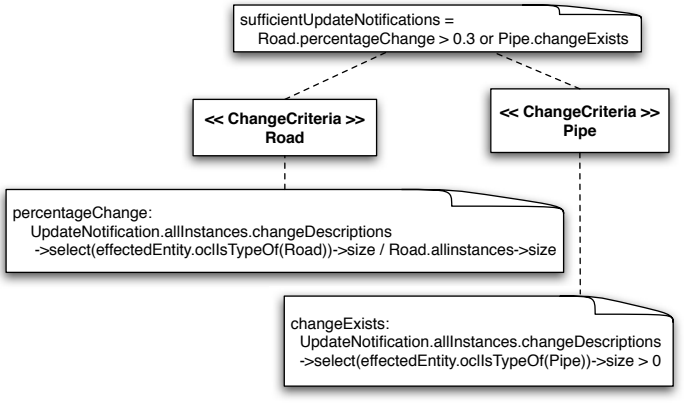
Scenario 3

> Based on the associated cost for update and the importance for each data set, C_3 only updates $\Lambda_{1,3,t}$ when either at least 30% of the data values in $\lambda_{1,t}$ have been changed or $\lambda_{2,t}$ is changed in any way. The import PIM ontology shown in Figure 2 is enhanced to incorporate this update policy as shown in Figure 6.

Sometimes, C_3 can neglect some minor updates in order to save network bandwidth and time or reduce the associated cost while the degree of inconsistency between $\Lambda_{1,3,t}$ and $\lambda_{1,t}$ or $\lambda_{2,t}$ is acceptable. However, under this situation, it needs extra care due to the accumulation effect of minor updates. For example, in this case, if 25% of data has been updated in $\lambda_{1,t+\Delta t}$ at $t + \Delta t$ and another 10% of data has been updated at t', then C_3 should update $\Lambda_{1,3,t}$ when the second update notification generated by P_1 has been received, since 35% of data in total has been updated.



(a) The package enclosing the ontology with the UpdatePolicy stereotype applied.



(b) The content of the package with stereotypes and tagged values

Fig. 6. The import PIM ontology shown earlier in Figure 2 enhanced to incorporate the update policy described in Scenario 3

4.1 UML Profile for Update Semantics

A complete description of the UML profile for Update Semantics is beyond the scope of this paper: therefore, selected stereotypes and tagged values defined by this profile are described to illustrate the intent.

«**UpatePolicy**»

This stereotype is applied once to a UML package symbol within which is defined an ontology (typically either an export or an import, PIM ontology). Associated with this stereotype are the following tagged values:

sufficientUpdateNotifications is an OCL expression which when true indicates that the available Update Notifications $\Delta\Lambda_{p,m,t'}$ justifies the updating of $\Lambda_{p,m,t}$.

updateSchedule which is assigned a value indicating the frequency with which $\Lambda_{p,m,t}$ is to be updated regardless of what has changed. Values that may be assigned to this tag are: `none, daily, weekly, monthly,` and `yearly`.

significantUpdate which is assigned the value of the following OCL expression: sufficientUpdateNotifications or (updateSchedule <> 'none'). When true, $\Lambda_{p,m,t}$ is to be updated.

Use of the updateSchedule tag indicates the consumer adopts a *pull* method while use of the sufficientUpdateNotification indicates the adoption of a *push* method for update notification.

«**ChangeCriteria**»

This stereotype is applied to classes for which any change may initiate the updating of the data set. This stereotype has the following tagged values to characterize the changes:

numberOfChanges an integer value indicating the number of change descriptions applicable to instances of this class or to this class itself.

percentageChange a real value between 0 and 1 indicating what percentage of the instances of this class in $\Lambda_{p,m,t}$ have been changed in $\Lambda_{p,m,t'}$.

changeExists a boolean value indicating that this class or instances of this class have been changed.

priority an integer value between 1 and 9 indicating the importance of the instance of this class. The lower the priority value, the higher the importance.

«**SpatialExtentCriteria**»

Like «ChangeCriteria», this stereotype is also applied to classes for which any change may initiate the updating of the data set: however, these changes are characterized by occurring to values within some specified spatial extent as defined by one of the following tagged values:

boundingBox the minimum bounding rectangular extent within which any change notifications or part thereof are regarded as potentially significant.

spatialBuffer For example all changes to pipes within 10 kilometers of a specified road centerLine.

«**Immutable**»

This stereotype will be applied to classes and or attributes of a class within an export PIM ontology to convey that instances of this class or values of this attribute will never be changed by the provider of λ_g.

The tagged values are to be used within OCL constraints to convey update policies unique to each consumer.

5 Conclusions and Future Research

In this paper is described ongoing research into facilitating the synchronization of a consumer's local copy of a distributed data set. The proposed solution involves each consumer incorporating into their platform independent import ontology of the local data set, defined using UML, a policy that governs when the data set is to be updated. The semantics of the update policy are documented within the ontology as OCL constraints expressed in terms of the stereotypes and tags defined by the proposed UML Profile. The platform independent import ontology

is one of four in the proposed four level framework which results from merging the OMG's MDA [8] with Seth and Larson's [9] notion of a five-level schema framework for distributed systems.

The platform independent import ontology with an embedded update policy is represented within an XMI document which is processed by software, an update agent, that implements the embedded update policy. Over time, this update agent listens for relevant update notifications from providers of the distributed data and evaluates the changes described either in isolation of, or in collaboration with, other update agents to determine when the local copy of a distributed data set is to be synchronized.

As the research continues, the UML profile for documenting the update policy will be refined to address issues that arise from implementation of this system as briefly described in the next section. Preliminary results suggest that the approach is feasible: however, much more experimentation is necessary to evaluate the efficacy of the solution proposed.

The system described in the paper is currently being implemented using Agent technologies such as OPAL [10], which incorporates an implementation of the Java Agent Services (JAS) specification [11], and two implementations of the Open GIS Consortium Web Feature Service (WFS) Specification [12] (Intergraph and geoserver, an Open Source project). Different versions of various data sets have been kindly made available by the Christchurch City Council to be used to create meaningful and realistic sequences of update notifications from two implementations of the WFS specification. Using this evolving implementation as a testbed for the research described here is the next phase.

An issue yet to be fully explored is the use of spatial operators in the definition of update policies. Consider, for example, the following scenario.

> C_3 would like to update $\Lambda_{1,3,t}$ when there are at least 5 changes to pipes within 10 kilometers of a specified road center line.

Creating an update policy with such a constraint expressed using OCL is difficult because, by default, OCL does not support spatial operations, such as 'Contains' and 'Overlaps'. Expressing such operators within OCL remains an open problem.

Acknowledgments

The research described here has been funded by the Foundation for Research, Science, and Technology subcontract number UOOX0208.

References

1. L. McKee, K. Buehler. The Open GIS Guide, *Open GIS Consortium, Inc*, Wayland, MA, 1996.
2. Y.A. Bishr, H. Pundt, W. Kuhn, M. Rdwan. Probing the concepts of information communities - A first step toward semantic interoperability. *Interoperating Geographic Information Systems*, pp. 55-70, Kluwer, Norwell, MA, 1999.

3. Frederico T. Fonseca and Max J. Egenhofer. Ontology-driven geographic information systems. In *Proceedings of GIS '99: Proceedings of the 7th ACM international symposium on Advances in geographic information systems*, ACM Press, New York, NY, 1999.
4. Object Management Group (OMG). UML 2.0 Superstructure Specification. October 8, 2004 http://www.omg.org/technology/documents/modeling_spec_catalog.htmi, 2004.
5. F.T. Fonseca. Users, Ontologies and Information Sharing in Urban GIS. *ASPRS Anual Conference*, Washington D.C., 2000.
6. S. Cranefield, M. Purvis. UML as an ontology modelling language. In *Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence* (IJCAI-99), 1999.
7. International Standards Organisation / Technical Committee 211 Geographic Information - Rules for application schema, ISO, 2005
8. Object Management Group (OMG) MDA Specifications. Available from: http://www.omg.org/mda/specs.htm.
9. A. Sheth, J.A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computer Surveys*, 22(3), 1990.
10. M. Purvis, S. Cranefield, G. Bush, D. Carter, B. McKinlay, M. Nowostawski and R. Ward. The NZDIS Project: an Agent-based Distributed Information Systems Architecture, In R.H. Sprague, Jr. (ed.) *CDROM Proceedings of the Hawaii International Conference on System Sciences (HICSS-33)*, IEEE Computer Society Press, 2000.
11. Francis G. McCabe. Java Specification Request 87: Java Agent Services Available from: http://www.jcp.org/en/jsr/detail?id=087.
12. Panagiotis A. Vretanos Web Feature Service Implementation Specification OpenGIS project document number OGC 04-094, version 1.1.0 3 May 2005. Available from: https://portal.opengeospatial.org/files/?artifact_id=8339.
13. Object Management Group. XML Metadata Interchange (XMI) Specification, Version 2.0. Available from: http://www.omg.org/mda/specs.htm, Document: formal/05-05-01.