

Topic Map Objects

Abstract. This paper presents a framework that provides domain specific classes for accessing and updating topic map data structures in a distributed environment. While work on Topic Map Web Services (TMWS) [1][2][3][4][5] have begun to provide developers with SOA type patterns for interacting with Topic Map data there is still a need for developers to be able to work with higher level views of the topic map data. TMAPI [6] has been the de-facto mechanism by which developers have accessed and updated topic map data. Topic Map Objects marries the advances in TMWS development with advances in modern language features of Java and C#. The result is a framework that allows developers to work with classes and objects that are relevant to the problem domain at hand. The expected contribution from this development is that developers can more easily use and work with topic maps with only a limited amount of knowledge about the full Topic Map Data Model (TMDM)[7]. In addition, developers will spend less time writing specific access code in order to construct domain specific applications.

Keywords: Topic Maps, TMDM, Object Framework, Persistence Models, Topic Map Web Services, Semantic Web.

1 Introduction

The Topic Map Data Model is a complex and heavy piece of machinery for developers to understand and use. In addition, TMAPI provides very little support for higher level abstractions and behavior. We consider these higher levels as aspects that are less generic than TMAPI and closer to the problem being addressed.

In the absence of these higher level abstractions developers are forced to build these themselves on top of TMAPI in order to help them complete their tasks. Topic Map Objects is a way to provide developers with this capability with minimal amounts of effort on their part. With a firm understanding of topic maps, TMAPI and also an understanding of the kinds of patterns developers will use in developing applications we are able to build the Topic Map Objects Framework.

The Topic Map Objects provides a framework that allows developers to create a set of classes and then using declarative attributes define how this set of classes map onto instances of a given topic map ontology. In addition to this data binding feature the framework also supports the construction of new topics and the updating of existing associations, names and occurrences by making changes to the objects used as representation proxies.

The approach is based on several key components; a topic map web service that supports high granularity of update, declarative property decorations on language

classes, the use of subject identifiers for topic map ontology topics, and that the topic map system will provide persistent and long lived item identifiers for all topics.

The paper first presents the framework from a high level architectural perspective. The role of each component of the architecture is described as are its dependencies with other components. Subsequently there are sections on each of the components along with examples of use where relevant.

We conclude with future directions this work can take and the upcoming key challenges.

2 Topic Map Objects Framework

2.1 Overview

The Topic Map Objects framework consists of two key services. We use the term “service” to mean a SOAP or HTTP service that exposes a number of named capabilities. One service is the Topic Map Web Service, a generic service that supports transactional updates to a topic map. The other is the Object Manager Service, which is able to create domain specific objects based on topic map data and return them serialized to a calling application.

The Object Manager Service is able to build domain objects because of information passed to it as part of the client request, but also from declarative information contained in class definitions. These declarations describe how a given instance of the class should be constructed from topic map data.

All domain specific classes inherit from a common base class or mix-in that provides support for making changes and additions to the localized model. These changes are recorded in such a way that they can be posted to the update capability of the Topic Map Web Service. This effectively closes the loop from initial request, to constructed object, to serialized object, to object on client, to changes to local model and then changes posted back to the update capability of the Topic Map Web Service.

2.2 The Object Manager Service

The object manager service is a web service that provides a capability that given a topic identifier and a class name it will return a serialized collection of objects. The reason for a collection of objects to be returned is that the initially constructed object can be related to other objects and these may also become instantiated and returned by the service.

The abstract definition of the service capability is as follows:

```
capability: conjureobject // the operation
parameter : topicidentifier // a topic identifier
parameter : classname // name of class to bind
```

An example of invoking the service using a HTTP service implementation is as follows:

```
GET
/tmobjs/conjureobject.aspx?topicidentifier=2324&classname=D.Person
```

What is returned from this service is a serialized form of the object(s) that are instantiated as a result of the request. This framework does not impose any restrictions on the format or organization of this representation. This provides flexibility and openness for developers to choose how the objects are serialized and de-serialized. There is an implicit expectation that the collection of classes that comprise the domain model is available to both the Object Manager Service and the client application. It is required on the client in order to deserialize and reconstitute the objects provided by the Object Manager Service capability.

The main semantics of the Object Manager Service are the interpretation of the different annotation types. The object construction process consists of first locating the topic specified in the request, constructing a single instance of the class requested and then proceeding to introspect all of the properties of the object. If the property is to be populated from the topic map it will have been decorated using one of the decoration types described. Decoration is a generic term for the annotation of class level properties in languages such as Java[8] and C#[9].

Annotation based applications, such as Topic Map Objects, define a set of Annotation classes. An annotation class is just as any other except the context in which it is used. An instance of an annotation class is used to 'say more' or add metadata about a property of another class.

Below, as an example, is the definition of one of the annotation classes from the Topic Map Objects library.

```
public class TopicPSIAttribute :TopicMapObjectAttribute
{
}
```

The thing to notice is that these annotation classes (in C# called Attribute classes) are many cases simply markers. They mark or decorate a property in a class such that some other process can know which properties to operate on and in what way. However, some annotation classes also have additional properties that can be populated when used. The TopicNameAttribute, shown below, is an example of this.

```
public class TopicNameAttribute : TopicMapObjectAttribute
{
    private string _scopePSI;
    public TopicNameAttribute(string scopePSI)
    {
        _scopePSI = scopePSI;
        if (_scopePSI==null) _scopePSI = "";
    }
}
```

```

public TopicNameAttribute()
{
    _scopePSI = "";
}

public string ScopePSI
{
    get {return _scopePSI;}
    set {_scopePSI = value; }
}
}

```

The following section on Annotation Types describes all of the annotation types that form the Topic Map Objects library. These annotation classes are, in a way, a description of the different ways that an object can be constructed from topic map data.

One the topic map service has constructed the object in line with the semantics associated with each Annotation type it serializes the structure and returns it to the calling application.

2.3 Annotation Types

The annotation types are the set of classes that allow the mapping of topic map data into a domain specific object. This section describes all of the annotation classes that comprise the Topic Map Objects library, and specifically, the interpretation of each class by the Topic Map Objects Service ‘conjure’ capability.

For each of the annotation types we provide its name, a description of any properties it has the interpretation of it as made by the Object Manager service.

TopicObjectIdentityAttribute

This attribute class is used to convey that a given class property should assume the value of the *ObjectID* of the topic object it is being bound to. The type of the class property must be a string. This attribute is rarely used explicitly by a solution developer as a base class called *TopicMapObjectBase* automatically maps this property using this decoration.

This attribute class has no additional properties.

TopicPSIAttribute

This attribute class is used to convey that a given class property should assume the value of the PSI (subject identifier in TMDM) that belongs to the topic object it is being bound to. The type of the class property must be a string. If more than one PSI exists then one is selected at random.

This attribute class has one additional property called `RegExp`. If this property is defined then the selected PSI must match the regular expression provided. If more than one PSI matches then one is chosen at random. The random selection in the case of more than one may seem dangerous, but subject identifiers are used for identity and all of them are equal, therefore it is not an issue to pick one from a bunch as they all resolve to the same subject.

TopicNameAttribute

This attribute class is used to convey that a given class property should assume the value of a name of the topic object it is being bound to. If the scope property is defined then the name selected will be the topic name value that is valid in that scope. The property type of the target property must be a string. If there is more than one name value then one is chosen at random.

If the class property type is an Array then all names that match the criteria will be added into the array.

TopicOccurrenceAttribute

This attribute class is used to convey that a given class property should assume the value of the occurrence that belongs to the topic object it is being bound to. The type and scope parameters are used to decide which occurrence value should be used.

If the class property type is defined as being an Array then all matching properties are added into the Array. If it is a singular property then one of the matching values is selected at random.

It has been seen in many applications that quite often there is only one property of a given type, i.e. age, height, weight. Thus the default of mapping just one property is an intuitive outcome.

TopicAssociationAttribute

This attribute class defines that the `IList` property belonging to the class will contain a list of objects. Each object is bound to one topic in the set of related topics as defined by the source role type, association role type and target role type PSIs properties of the annotation class. The name of the class defines which `C#` class is to be used in the binding process. The `IsLazy` property indicates that this collection should be constituted only when accessed on the client.

This annotation class contains the full definition of the relationship between the focus topic (this object) and the related topics. This is to enable the client to create an update that can add and remove associations based on changes to the local model.

There is scope, in a read only mode to enable the related topics to be defined via a TMQL query. The reason this would need to be read-only is that it would be semantically ambiguous what to do if a new object were to be added to the collection or removed.

When the Object Manager Service traverses an association then it will construct a new object based on the class name and then re-apply the same semantics for building that instance. The IsLazy property allows developers to ensure that circular models, long lists or deep trees are not pulled down in one large lump. However, there may be certain times when a client application would want to load the entire model into memory.

Example Annotation

The following example shows how two classes are annotated. The classes involved are Person and Skill. The model that is to be utilized is the fact that a Person will have several skills.

```
[TopicTypeAttribute("http://www.networkedplanet.com/person")]
public class Person : TopicMapObjectBase
{
    private string m_name;
    private string m_age;
    private string m_homepage;
    private IList m_skills;

    [TopicNameAttribute()]
    public string Name {
        get { return m_name; }
        set {
            OccurrenceSet(this, "Name", value);
            m_name = value;
        }
    }

    [TopicOccurrenceAttribute("http://www.networkedplanet.com/octypes/age")]
    public string Age {
        get { return m_age; }
        set {
            OccurrenceSet(this, "Age", value);
            m_age = value;
        }
    }

    [TopicOccurrenceAttribute("http://www.networkedplanet.com/octypes/homepage")]
    public string HomePage {
        get { return m_homepage; }
        set {
            OccurrenceSet(this, "HomePage", value);
            m_homepage = value;
        }
    }
}
```

```

[TopicAssociationAttribute("http://www.networkedplanet.com/atyp
es/personhasskill",

"http://www.networkedplanet.com/roletypes/person",

"http://www.networkedplanet.com/roletypes/skill",

"NetworkedPlanet.TopicMapObjects.TestData.Skill,TopicMapObjectTe
stData", true)]
    public IList Skills {
        get { return m_skills; }
        set { m_skills = value; }
    }
}

// The Class Skill
[TopicTypeAttribute("http://www.networkedplanet.com/skill")]
public class Skill : TopicMapObjectBase {
    protected string m_name;

[TopicNameAttribute("http://www.networkedplanet.com/scopes/engli
sh_name_scope")]
    public string Name {
        get { return m_name; }
        set {
            OccurrenceSet(this, "Name", value);
            m_name = value;
        }
    }
}

```

Variants

Variants are generally considered to be a complex and messy aspect of the topic map model. This is important as variants do not map easily to a single object property, but instead have a dependency on another value. To this end they are not included in the Topic Map Objects framework, although creating an annotation class for them would be possible.

2.4 Topic Map Objects Client

The a client of the Topic Map Object service in general doesn't need to do anything different from using a normal set of objects. When actions occur on the object model internally the TopicMapObjectBase class creates topic map update events. When an occurrence is changed a delete and create occurrence event are created. When a new topic is added to a collection of topic map managed objects then a new object is

created and in the topic map transaction, a new topic with a GUID along with a new association is defined.

When a client application has made a collection of changes then it can instruct the client object to post the update transaction back to the server.

Topic Map Objects has been engineered to make use of optimistic locking. To this end, if a change has occurred to any topic since the client made a read, then any changes by the client will result in a concurrent modification exception. The version information is stored in the TopicMapObjectBase object. This means that it is hidden from developers.

If a collection is defined as 'IsLazy' then upon a user accessing such a collection a new request is sent to the Topic Map Object Service to construct the required objects.

2.5 Example Usage

This example shows a client conjuring a Person instance, iterating a set of skills, deleting one skill and creating a new skill *and* adding it to the skills of the person.

```
object o =
tmom.Conjure("http://www.networkedplanet.com/people/gdm",
typeof(Person));

Person p = o as Person;
System.Console.WriteLine(p.HomePage);
System.Console.WriteLine(p.Age);
System.Console.WriteLine(p.Name);

foreach (Skill s in p.Skills)
{
    System.Console.WriteLine(s.Name);
}

// remove the relationship between a skill and a person.
// Note the skill is not deleted.
p.Skills.Remove(p.Skills[0]);

// add a new skill
Skill nskill = new Skill();
nskill.Name = "sql";
p.Skills.Add(nskill);

// look at txn
TopicMapTransaction txn =
TopicMapTransaction.GetTopicMapTransaction();

UpdateServiceClient usc = new
UpdateServiceClient("http://localhost/updateservice/tmupdateserv
ice.aspx");
```

```
XmlDocument xdoc = new XmlDocument();  
xdoc.LoadXml(txn.XML("tmobjects"));  
usc.PostUpdate(xdoc);
```

2.6 The Topic Map Web Service Update Capability

The Topic Web Service Update capability supports a number of operations to update, add or remove topic map data. The full capability of this service goes beyond the scope of this paper.

However, to complete the picture in the context of Topic Map Objects it should be noted that the transaction consists of delete and add operations for Topics, Names, Occurrences and Associations. All of these operations occur by value and occur within a single atomic transaction.

If any aspect of the transaction fails then the whole transaction is rolled back. This is to ensure that the topic map is always preserved in a consistent state. If an exception occurs in update then the client is required to re-fetch the objects it needs to ensure that it has the latest version which it can then begin modifying.

2.7 Packaging and Deploying the Different Components

The Topic Map Objects can be grouped into a number of different libraries. The reason for having a number of libraries is to ensure that client applications can be light weight modules that do not have a binary dependency on any of the server, Topic Map Objects service libraries or dependencies.

This approach supports a fully distributed model with no server dependencies. It also enables a heterogeneous environment with different language servers supplying different language clients.

3 Applications of Topic Map Objects

3.1 Data transformer

Topics Maps is a powerful technology for integrating data sources. However, one of the problems is how to easily map data into the topic map model. Using Topic Map Objects allows developers to use their existing sets of classes and with some schema bound annotation map their data into that of the topic map.

The vision is that multiple models that before were not mergeable become so when each is mapped into a topic map using this technology.

An example of this is to integrate different XML languages into Topic Maps Objects and then into a topic map as a way to merge different XML vocabularies

without needing to develop a bespoke merger for each combination of XML vocabularies.

3.2 Multiple views on data

The truth, or usefulness of a topic map can vary based on who is looking at it and in what context. The dynamic and late binding of the objects to the data means that developers can map a data and behavioural model to topic map data based on the application and the context. In addition, they are not forced to use one definitive model.

This freedom of expression enables developers to create patterns of code that can be bound to topic map structures in ways that best solve the problem to hand.

3.3 Hibernate on Steroids

One of the interesting aspects of Topic Map Objects, is that it is very similar to efforts such as Hibernate[8]. However, we see the dynamic and late binding, as well as the fundamental flexibility of the Topic Map Model as unique features that are not present in Hibernate.

4 Conclusion and Future Work

Future work will look at new patterns of attribute classes that can add more expressiveness and support other common patterns. Also, the use of TMCL to generate a starting set of classes would also be a valuable and worthwhile research activity. This would also lead to an extension of set of attribute classes.

However, without further development the Topic Map Objects Framework is a powerful and exciting addition to the toolset of all topic map application developers.

References

1. G.Moore, Kal Ahmed.: Networked Planet Topic Map Web Service,;
<http://www.networkedplanet.com/technology/webservices/intro.html>
2. Thomas Schwartzer, Graham Moore,.; SNAPI – Semantic Network API,
<http://sourceforge.snapi.org>
3. Graham Moore, Steve Pepper, .; TMRAP – Submission to SC34
4. Robert Barta; TMIP – Topic Map Interaction Protocol, Bond University
5. Graham Moore,.; Semantic Web Servers – Extreme 2004
6. Kal Ahmed, Lars Marius Garshol, Graham Moore,.; TMAPI – Topic Map API,
<http://sourceforge.tmap.org>
7. Lars Marius Garshol, Graham Moore (eds); Topic Map Data Model, ISO13250 part 2.

8. Hibernate, <http://www.hibernate.org/>.