Extreme Markup Languages 2005®

Montréal, Québec
August 1-5, 2005

Multiple Subject Map Patterns for Relationships and TMDM Information Items

Steven R. Newcomb
Coolheads Consulting

Patrick Durusau
Independent Consultant

Abstract

Open source implementations of Subject Map Disclosures (SMDs, aka TMAs) for relationship proxies are emerging. Each SMD represents ontological commitments regarding relationship reification. Proxies governed by different SMDs can be merged, resulting in proxies whose properties are governed by multiple SMDs. Thus, each relationship can be viewed from multiple perspectives. Examples are demonstrated using Versavant.

The same techniques can distinguish between the subjects which are TMDM information items vs. the subjects for which Topic information items are proxies. The two kinds of subjects have different properties, a fact perhaps most visible when a Topic information item "reifies" another information item.

A Conference of IDEAlliance
Extreme Markup 2005 Languages®

Multiple Subject Map Patterns for Relationships and TMDM Information Items

Table of Contents

A Conference of IDEAlliance
Extreme Markup 2005 Languages®

Multiple Subject Map Patterns for Relationships and TMDM Information Items

Steven R. Newcomb and Patrick Durusau

§ Abstract

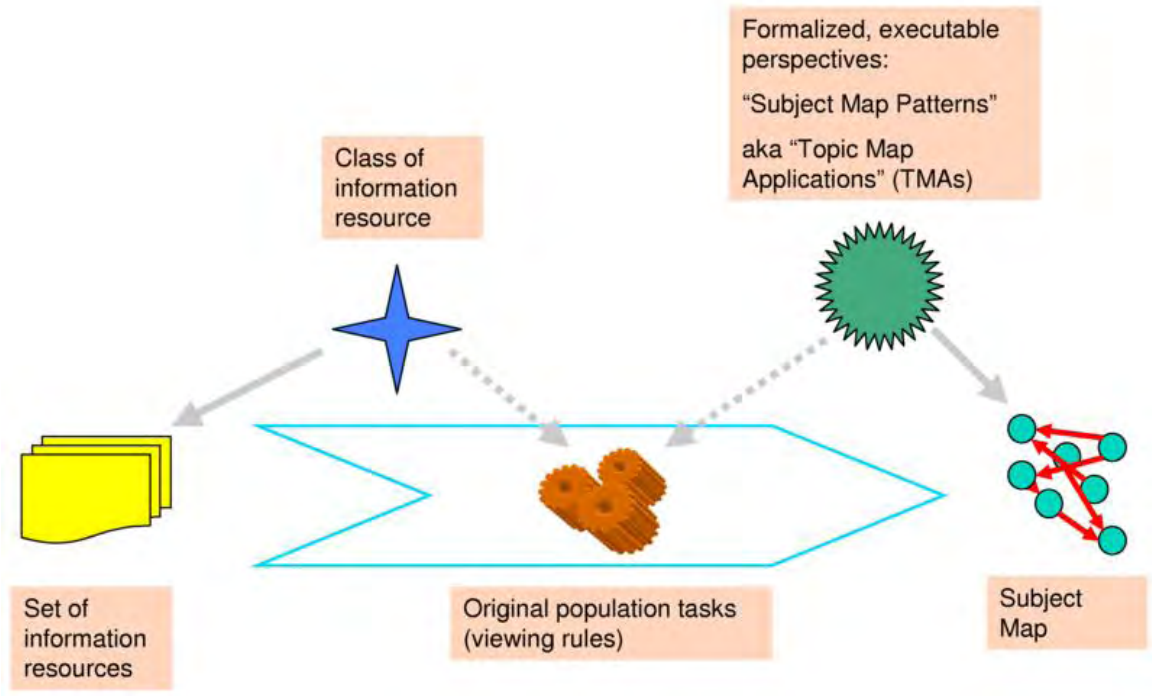
Open source implementations of Subject Map Patterns (SMPs, also sometimes called "Topic Map Applications (TMAs)") for relationship proxies are emerging. Each SMP represents ontological commitments regarding relationship proxification. Proxies governed by different SMPs can be merged, resulting in proxies whose properties are governed by multiple SMPs. Thus, each relationship can be viewed from multiple perspectives. Examples are demonstrated using Versavant.

The relationship model (associations) in the Topic Map Data Model (TMDM) is a subject map pattern. We reify associations and the resulting subject proxy has the combined properties of both TMDM "association" information items and TMDM "topic" information items -- in contrast to an "association" information item and its reifying "topic" information item remaining as separate information items. In addition to representing an "association" as a subject map pattern, we discuss a use case for a variation on that pattern.

§ Introduction

Subject Maps are Subject-centric Views of Data

Figure 1



A subject map is a subject-centric way of viewing information resources. The viewing rules (shown as gears) present structures seen in the data (collectively shown as a blue four-pointed star), and those structures are viewed through one or more formalized perspectives (collectively shown as a green many-pointed star) to produce a subject map.

In a subject map, here shown as a graph with green nodes, there is one virtual "place" for each subject, and everything known about that subject is available from the perspective of the one-and-only node ("subject proxy") that represents that subject.

Subject Maps Consist of Subject Proxies

Subject maps are expressed as sets of subject proxies whose properties have values that are data and/or other subject proxies. Subject proxies can be interchangeable syntactic objects, or executable in-memory objects, or they can be implemented in any other way that is disclosed in a way that meets the requirements of the paradigm. Every subject proxy:

- Represents one subject.
- Consists of "property instances" (key/value pairs).
- Has at least one "subject identity property (SIP)".
- Has unique identity as a proxy.

Thus, every proxy has two kinds of identity/addressability:

1. The identity and address(es) of its subject, and
2. its identity and address as a proxy.

Versavant: An Open-source, Modular Multi-pattern Subject Map Engine

We used the Versavant subject map engine (**http://www.versavant.org**) to demonstrate the research that we're reporting in this paper. Many of the slides reflect displays made by the Versavant engine's built-in subject map debugging tool. The following diagram and discussion is intended to allow the reader to understand Versavant's subject proxy displays that will be used later.

Figure 2

A Versavant debugger display of a subject proxy.

In figure 2, the subject proxy whose unique identifier is "001166" is shown. (In these examples, Versavant proxy identities are represented as decimal serial numbers that begin with two leading zeroes. They could, alternatively, be URIs or any other unique identifier.) The proxy is a member of the set of proxies of which the subject map called "Jacks Topic Map" consists. The proxy consists of ten property instances, only one of which is a "subject identity property (SIP)"; SIPs can be used for many kinds of information, but only SIPs can be used to detect whether two proxies are proxies for the same subject. The name (or "key") of each property instance is the concatenation of:

1. the name of the subject map pattern that defines its class, shown in italics,
2. a field separator (two commas), and
3. the name of the property class of which it is an instance.

For example, the first-listed property instance (which is a subject identity property [SIP]) is governed by the subject map pattern called "JacksTMA"; the name of the class of which it is an instance is "Name". The key of the property instance is "JacksTMA,,Name". The value of the property instance is the string "Joe".

For another example, the second-listed property instance (which is *not* a subject identity property, and is instead an "other property" [OP]) is governed by the subject map pattern called "A_assert", and the name of the class of which it is an instance is "x_roles". The value of the property instance is a list in which the first item is a two-member tuple.

A third example is the fifth-listed property instance. Its key is "U_assert,,x_LrnRrnRrp"; its value is a dictionary (a mapping object). The dictionary has one key ('nameOfPerson') whose value is a dictionary that has one key.

> *NOTE:* The upward-pointing gray triangle beside each property value is a link to an auditing display that shows how, and from what other properties, if any, the value was calculated. The downward-pointing gray triangle is a link to an auditing display that shows which properties were calculated on the basis of this property's value.
>
> Every property comes into existence in one of three ways: it is either:
>
> 1. "originally populated", probably on the basis of one or more "viewing rules" -- represented by the gears shown in Figure 1, or
> 2. "conferred into existence" as a result of the operation of a "conferral rule" that forms part of the subject map pattern that governs the property's class (subject map patterns are represented by the multi-pointed green star shown in Figure 1, or
> 3. the result of the merger of two or more instances of the same property class.

What is a Subject Map Pattern?

A subject map pattern is:

A set of subject address spaces.	In order to allow subject map engines to detect whether multiple proxies are proxies for the same subject, subject map patterns include ways of endowing subjects with addresses, and ways of detecting whether multiple subject addresses address the same subject. In fact, these ways of addressing subjects constitute "subject address spaces".
A vocabulary of property names (keys) and their semantics	A subject map pattern defines a vocabulary of property class names.
A perspective.	In other words, a subject map pattern is a way to formalize a universe of discourse, or a way of thinking and talking about some set of subjects. Most ontologists would say that a subject map pattern is an ontology, but it is an ontology with an interesting extra feature: it includes all the rules necessary to to detect when multiple proxies represent the same subject, and to allow multiple proxies for the same subject to be seen as a single proxy.

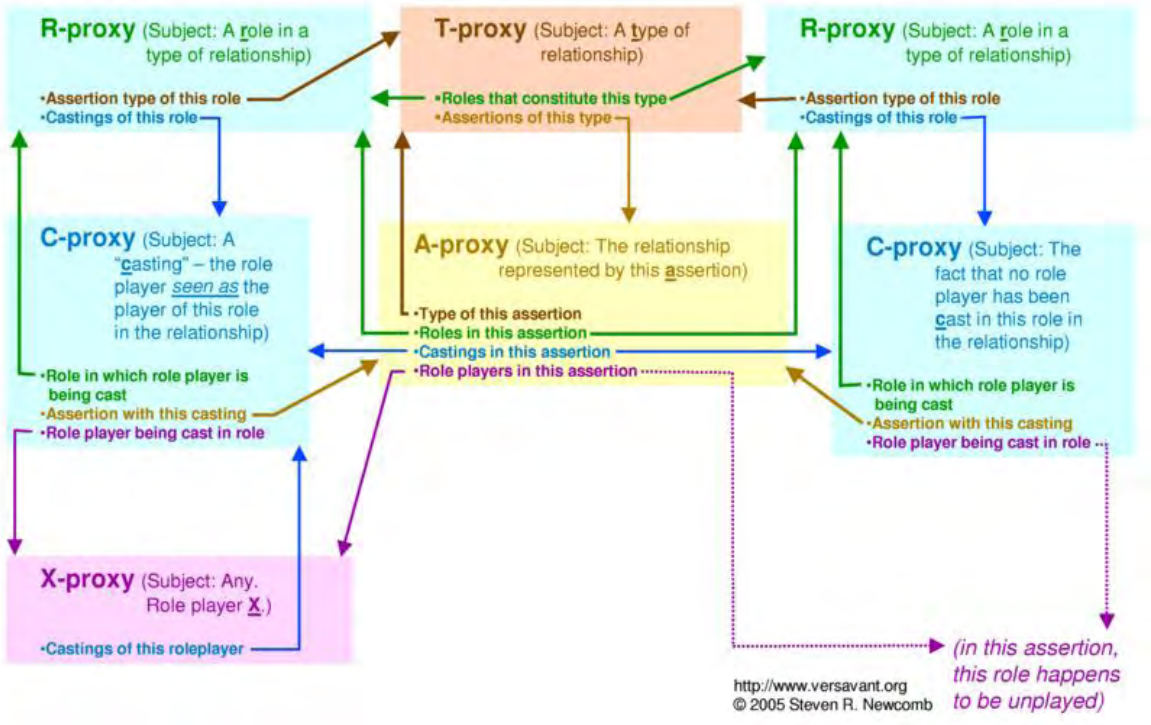
What is a Subject Map Pattern Disclosure?

A subject map pattern *disclosure* is:

1. Definitions of a set of *property classes*. The subject map pattern is said to "govern" all instances of these classes.
2. Optionally, a set of *conferral rules*. These rules are triggered by property instances, and the result of their operations, if any, are addtional property instances.

An Example of a Subject Map Pattern: "BigAssert"

Figure 3



A conceptual diagram of the "BigAssert" subject map pattern.

The "BigAssert" subject map pattern is designed to allow relationships between subjects to be seen as constellations of subjects.

> *NOTE:* What is now called the "BigAssert" subject map pattern was discussed in a paper given at Extreme Markup Languages in 2003 (**http://www.mulberrytech.com/Extreme/Proceedings/html/2003/Newcomb01/EML2003Newcomb01.html**).

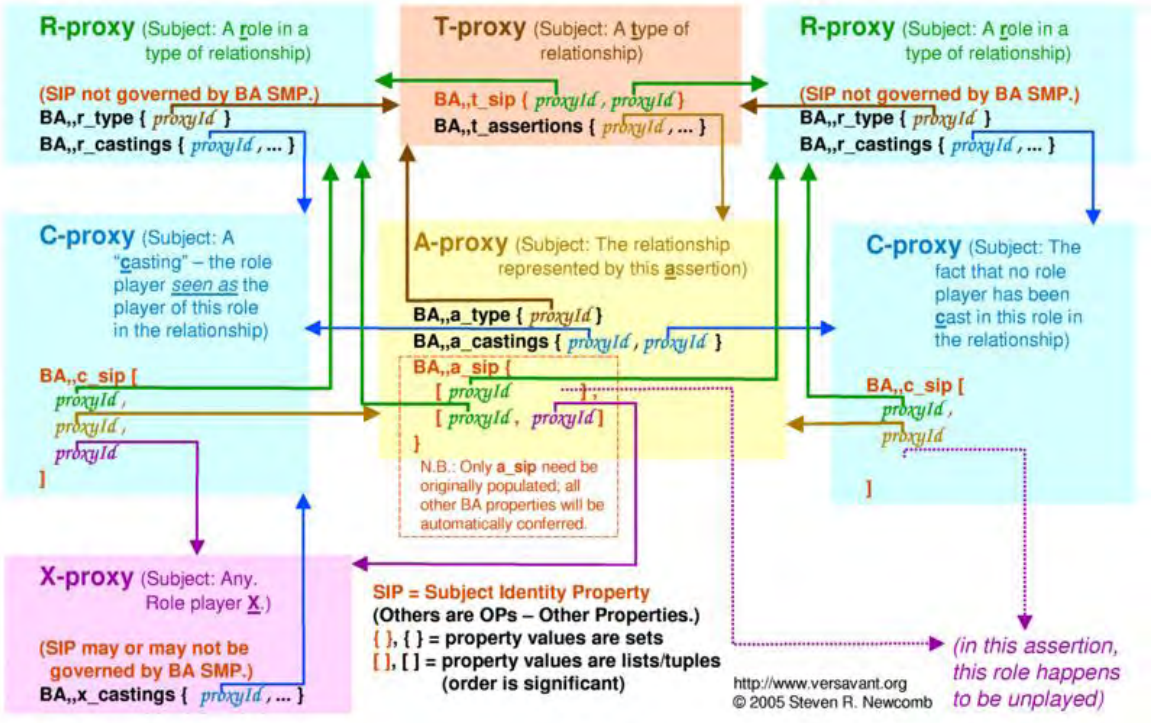
Figure 3 shows the subject proxies of a BigAssert relationship as colored boxes.

- The subjects that have a relationship between them are called "role players", because each of them plays a specific role in each of the relationships in which it participates; proxies for roleplayer subjects are called "X-proxies" (pink box in lower left).
- The subject that is the relationship itself is called a "relationship"; the proxy that represents it is called an "A-proxy", or "assertion" (yellow box at the center of the diagram).
- The subjects that are the roles that are played are called "roles"; the proxies that represent them are called "R-proxies" (green boxes at the upper left and upper right of the diagram).
- The subject that is the type of relationship of which the relationship is an instance is called a "relationship type". The proxy that represents the relationship type is called a "T-proxy" (brown box at top center of the diagram).

- The subjects that are the role players seen as the players of their roles in the relationship are called "castings". The proxies that represent castings are called "C-proxies" (blue boxes at center left and center right of the diagram).

Listed in each subject-proxy-box in Figure 3 are the notional properties of each kind of subject.

Figure 4



A detailed diagram of the "BigAssert" subject map pattern.

Figure 4 shows the BigAssert subject map pattern's actual property classes and the structure of the values of their instances. It's especially interesting to note the following things about BigAssert:

- BigAssert provides SIP classes for A-proxies, T-proxies, and C-proxies, but not for R-proxies, even though OPs for R-proxies are provided.
- BigAssert doesn't provide SIP classes for X-proxies, either.
- When instantiating a BigAssert in a subject map, it is only necessary to create an A-proxy and its SIP (an instance of the property class named "a_sip"). All the other properties and proxies of each instance of a BigAssert pattern will be "conferred into existence" automatically by the conferral rules defined by the pattern.

§ Jack Park's Challenge

Jack Park, who gave the keynote presentation at the 2002 Extreme Markup Languages Conference, "Douglas Engelbart, Open Hyperdocument Systems, XML, and everything" (**http://www.mulberrytech.com/Extreme/Proceedings/html/2002/Park01/EML2002Park01.html**), recently challenged us to disclose the assertion model that we've been talking about for years as a "Subject Map Pattern" (formerly known as a "Topic Map Application") in a way that would meet the disclosure requirements envisaged in the draft Topic Maps Reference Model (ISO/IEC 13250-5, "TMRM"), an emerging standard co-edited by the authors of this paper.

Jack was interested in actually seeing the BigAssert model work. He gave us a simple example to implement as a subject map. Here's what he said:

> Consider a subject, a specific individual whom I won't identify for the time being, but who has a name: Joe.
>
> I want to say that Joe's shoesize was a 6 when he was 7 years old. How do I do that?

We replied that we needed more information. Specifically, we needed to know what, exactly, Jack wanted to proxify in this subject map.

> *NOTE:* "Proxify" means "provide a subject proxy for" and, therefore, at least in the context of this "subject maps" paradigm, it also implicitly means "provide with an address in one or more subject-address spaces".

In other words, we needed specific yes-or-no answers to the following questions:

1. (We assumed he wanted to proxify Joe, so we didn't ask.)
2. Do you want to proxify shoe sizes?
3. Do you want to proxify ages?
4. Do you want to proxify the relationships between shoe sizes and the people who have them?
5. Do you want to proxify the relationships between shoe sizes the ages at which they were observed?
6. Do you want to proxify the names of people?
7. Do you want to proxify the relationships between people and their names?
8. Is there anything else about this example that you want to proxify? (If your answer is "yes", please give details.)

Jack replied:

> The categorical answer, given that I think that a topic map is there to point to answers to questions is YES. I want to be able to make statements about people, places, and things, the statements for which I want to be accurate, timely, and discussable. It's not enough to say "Oh, Joe wore size 6 shoes" and leave the user wondering when ... he wore that size shoe or [whether he is a little person]...

We gathered that his answer to all but the last of our questions was "yes".

We answered his question about how to build a subject map in terms of the API of the open-source implementation (**www.versavant.org**) of the Versavant Disclosure Discipline. In that API, and assuming you have already created a subject map, there are basically two steps to the proxification of any subject:

1. Create a proxy for the subject to be proxified.
2. Add an SIP (a Subject Identity Property) to the proxy you just created.

> *NOTE:* At least theoretically in terms of the TMRM, it's impossible for a proxy to exist without an SIP, so the act of creating a proxy necessarily *includes* the act of providing it with an SIP. As a practical matter, at least in the Versavant API, it's a two-step process.

§ Jack's example, step by step

We needed to declare some Subject Identity Property classes (SIP classes) in order to provide a way to proxify the subjects Jack was interested in. We decided to declare them as a group, in a subject map pattern we called "JacksTMA" ("TMA" is an acronym for "Topic Map Application", which is what we used to call subject map pattern). We'll describe them below as we need them.

We needed to create a subject map (a "subject map" is a set of subject proxies that is treated as a unit) in which to put all the subject proxies we were about to create, and we needed to declare that it might contain instances of the property classes declared by the JacksTMA subject map pattern.

```
vsv.newTopicMap( tmName,  ## name of this topic map
                 tmDesc)  ## description of this topic map
vsv.loadTMA( tmName, '    ## topic map name
             JacksTMA',   ## name of subject map pattern
             'JacksTMA/tmaconf.xml')  ## SMP config file
```

Proxifying Joe's name

We declared a subject identity property class (an SIP class) in JacksTMA. We named it "Name". We declared ("disclosed") that proxies that have instances of this property class will merge (i.e., that they will be viewable as if they were one and the same proxy) if the values of their respective instances of this property class are the same. We declared that when multiple instances of this property are merged into a single instance, any one of the original values becomes the value of the new single property instance.

> *NOTE:* The above are the highlights of the disclosure of the "Name" property class of JacksTMA. The full details of the discipline are beyond the scope of this presentation, but we plan to present them at a Nocturne tonight.

In our subject map, we created a proxy for the name "Joe", and then we added to that proxy an instance of the "Name" property class, whose value was the string "Joe".

```
proxyForTheNameJoe = vsv.newProxyObject()

vsv.newPropertyInstance(
    proxyForTheNameJoe,  ## proxy
    'JacksTMA',          ## subject map pattern ("TMA")
    'Name',              ## property class name
    'Joe')               ## property value
```

> *NOTE:* In Versavant, this kind of process is implemented as a user-specified "original population task" -- a function whose purpose is to populate a subject map with subject proxies *de novo*.

Proxifying Joe

We declared a subject identity property class (an SIP class) in JacksTMA. We named it "Person Name Proxy". We made the necessary disclosures about this new property class.

In our subject map, we created a proxy for Joe (the person himself) and then we added to that proxy an instance of the "Person Name Proxy" property class, whose value was the proxy for the name "Joe".

```
proxyForJoe = vsv.newProxyObject()

vsv.newPropertyInstance(
    proxyForJoe,                  ## proxy
    'JacksTMA',                   ## subject map pattern
    'Person Name Proxy',          ## property class name
    [ proxyForTheNameJoe.proxyId]) ## reference to Joe's name's proxy
```

Proxifying the relationship between Joe and his name

We added the BigAssert subject map pattern to our topic map, so that we could proxify relationships:

```
vsv.loadTMA( tmName,
             'BigAssert',
             '${VSVEXECDIR}/Lib/BIGASSERT/tmaconf.xml')
```

Since BigAssert does not provide SIP classes for the proxification of roles, we decided to use a general-purpose subject map pattern for role proxies, which we called "Roles". It discloses an SIP class called "role name". So, we loaded the pattern:

```
vsv.loadTMA( tmName,
             'Roles',
             '${VSVEXECDIR}/Lib/ROLENAME/tmaconf.xml')
```

Now we were in a position to create proxies for subjects that are roles played in relationships. We needed to create a kind of relationship with two roles -- roles we called "nameOfPerson" and "namedPerson", respectively, so that we could proxify relationships between persons and their names (and, more specifically, between Joe and his name). Then we created proxies for those roles:

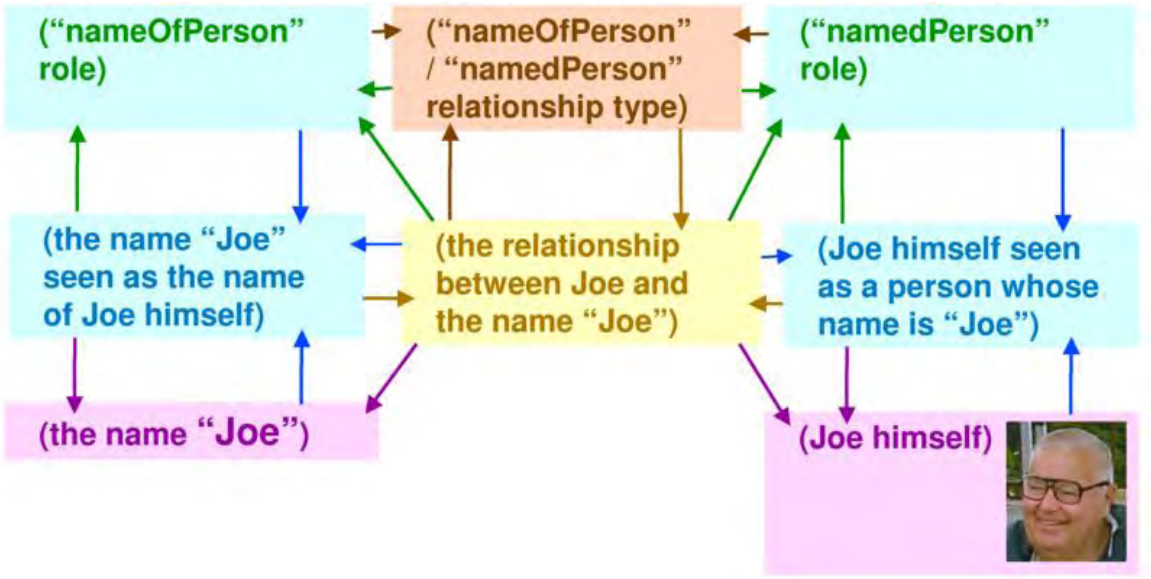
```
proxyForTheNamedPersonRole = vsv.newProxyObject()

vsv.newPropertyInstance(
    proxyForTheNamedPersonRole,
    'Roles',
    'role name',
    'namedPerson')
```

```
proxyForTheNameOfPersonRole = vsv.newProxyObject()

vsv.newPropertyInstance(
    proxyForTheNameOfPersonRole,
    'Roles',
    'role name',
    'nameOfPerson')
```

Now we were ready to proxify the relationship between Joe and his name, using the BigAssert pattern.

```
proxyForRelationshipBetweenJoeAndHisName = vsv.newProxyObject()

vsv.newPropertyInstance(
    proxyForRelationshipBetweenJoeAndHisName,  ## proxy
    'BigAssert',                               ## subject map pattern
    'a_sip',                                   ## property class
                                             ## Value:
    [ ( proxyForTheNamedPersonRole.proxyId,  ## ( role proxy,
        proxyForJoe.proxyId),                ## role-playing proxy)
      ( proxyForTheNameOfPersonRole.proxyId, ## ( role proxy,
        proxyForTheNameJoe.proxyId) ])       ## role-playing proxy)
```

Figure 5



A BigAssert representation of the relationship between Joe and his name.

NOTE: As previously noted, the rest of the proxies and properties of the BigAssert pattern will be automagically "conferred into existence" later, at merge time.

Figure 5 diagrams the work we've done so far. There are proxies for the role-playing subjects that are Joe and Joe's name, the "nameOfPerson" role, the "namedPerson" role, the "nameofPerson/namedPerson" relationship type, the relationship between Joe and his name, and the casting subjects.

Proxifying the observation of Joe's shoe size at age 7

We added a "Shoe Size" SIP class to JacksTMA, and we proxified size 6:

```
proxyForShoeSize6 = vsv.newProxyObject()

vsv.newPropertyInstance(
    proxyForShoeSize6,
    'JacksTMA',
    'Shoe Size',
     '6')
```

We added an "Age" SIP class to JacksTMA, and we proxified age 7:

```
proxyForAge7 = vsv.newProxyObject()

vsv.newPropertyInstance(
    proxyForAge7,
    'JacksTMA',
    'Age',
    '7')
```

We decided to represent observations of shoe sizes at particular ages as relationships between shoe sizes and ages.

NOTE: There are many ways to model things. We thought this would be fun. Why not?

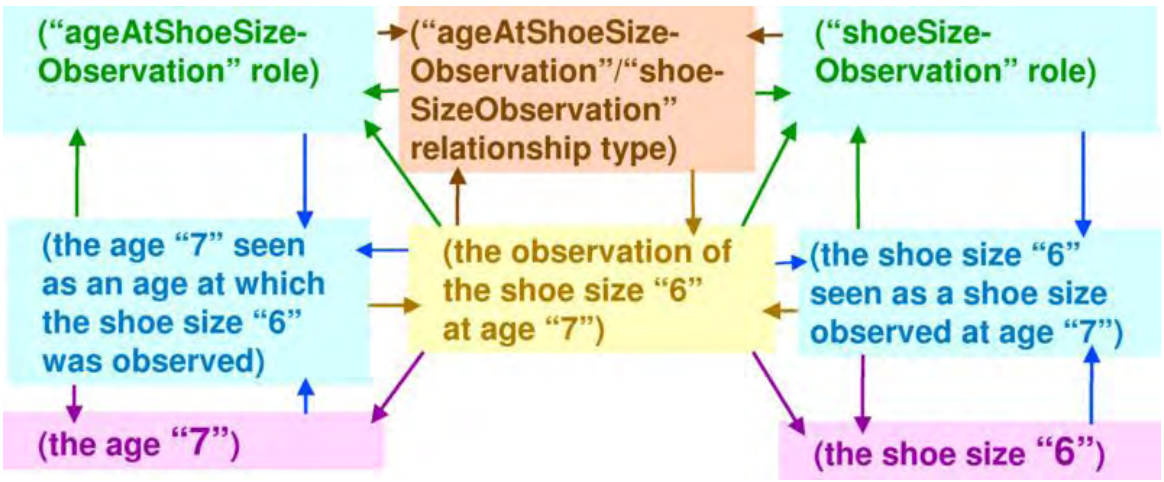
So, we needed a couple of roles. We decided to call them "shoeSizeObservation" and "ageAtShoeSizeObservation", and we proxified them:

```
proxyForTheShoeSizeObservationRole = vsv.newProxyObject()

vsv.newPropertyInstance(
    proxyForTheShoeSizeObservationRole,
    'Roles',
    'role name',
    'shoeSizeObservation')

proxyForTheAgeAtShoeSizeObservationRole = vsv.newProxyObject()

vsv.newPropertyInstance(
    proxyForTheAgeAtShoeSizeObservationRole,
    'Roles',
    'role name',
    'ageAtShoeSizeObservation')
```

Now we were ready to proxify the relationship between the shoe size 6 and age 7:

```
proxyForRelationshipBetweenJoesShoeSizeAndHisAgeAtObservation = vsv.newProxyObject()

vsv.newPropertyInstance(
    proxyForRelationshipBetweenJoesShoeSizeAndHisAgeAtObservation,
    'BigAssert',
    'a_sip',
    [ ( proxyForTheShoeSizeObservationRole.proxyId,       ## role
        proxyForShoeSize6.proxyId),                       ## role player
      ( proxyForTheAgeAtShoeSizeObservationRole.proxyId,  ## role
        proxyForAge7.proxyId) ])                          ## role player
```

Figure 6



A BigAssert representation of a shoe size observation.

Proxifying the relationship between Joe and the observation of his shoe size at age 7

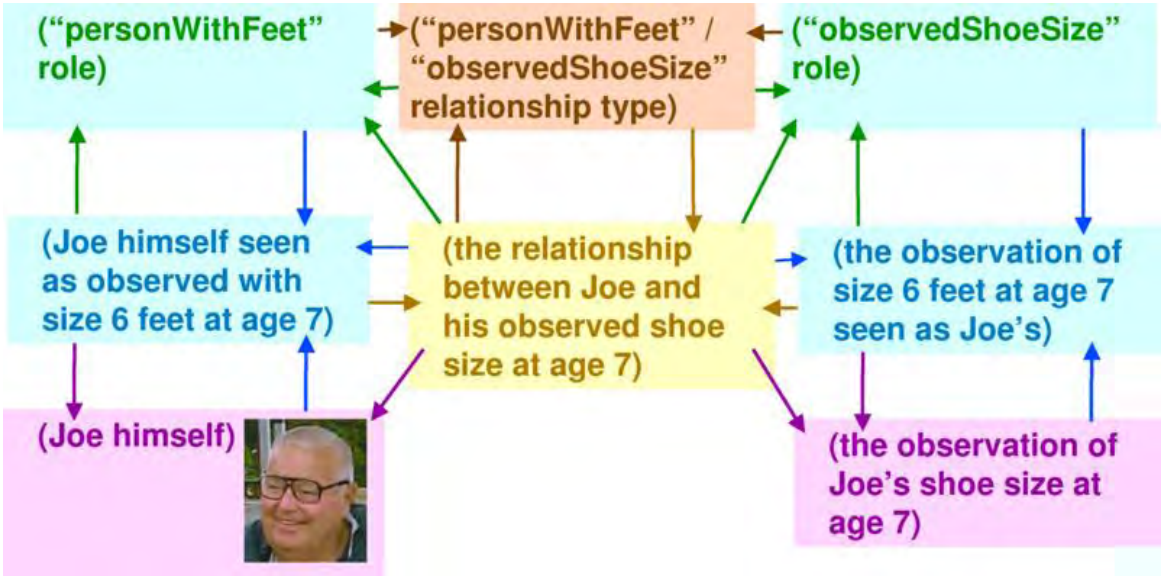
We needed a couple of roles. We decided to call them "personWithFeet" and "observedShoeSize", and we proxified them:

```
proxyForThePersonWithFeetRole = vsv.newProxyObject()

vsv.newPropertyInstance(
    proxyForThePersonWithFeetRole,
    'Roles',
    'role name',
    'PersonWithFeet')

proxyForTheObservedShoeSizeRole = vsv.newProxyObject()

vsv.newPropertyInstance(
    proxyForTheObservedShoeSizeRole,
    'Roles',
    'role name',
    'ObservedShoeSize')
```

Now we were ready to proxify the relationship between Joe and his observed shoe size at age 7:

```
proxyForRelationshipBetweenJoeAndHisObservedShoeSize = vsv.newProxyObject()

vsv.newPropertyInstance(
    proxyForRelationshipBetweenJoeAndHisObservedShoeSize,
    'BigAssert',
    'a_sip',
    [ ( proxyForThePersonWithFeetRole.proxyId,
        proxyForJoe.proxyId),
      ( proxyForTheObservedShoeSizeRole.proxyId,
        proxyForRelationshipBetweenJoesShoeSizeAndHisAgeAtObservation.proxyId) ])
```

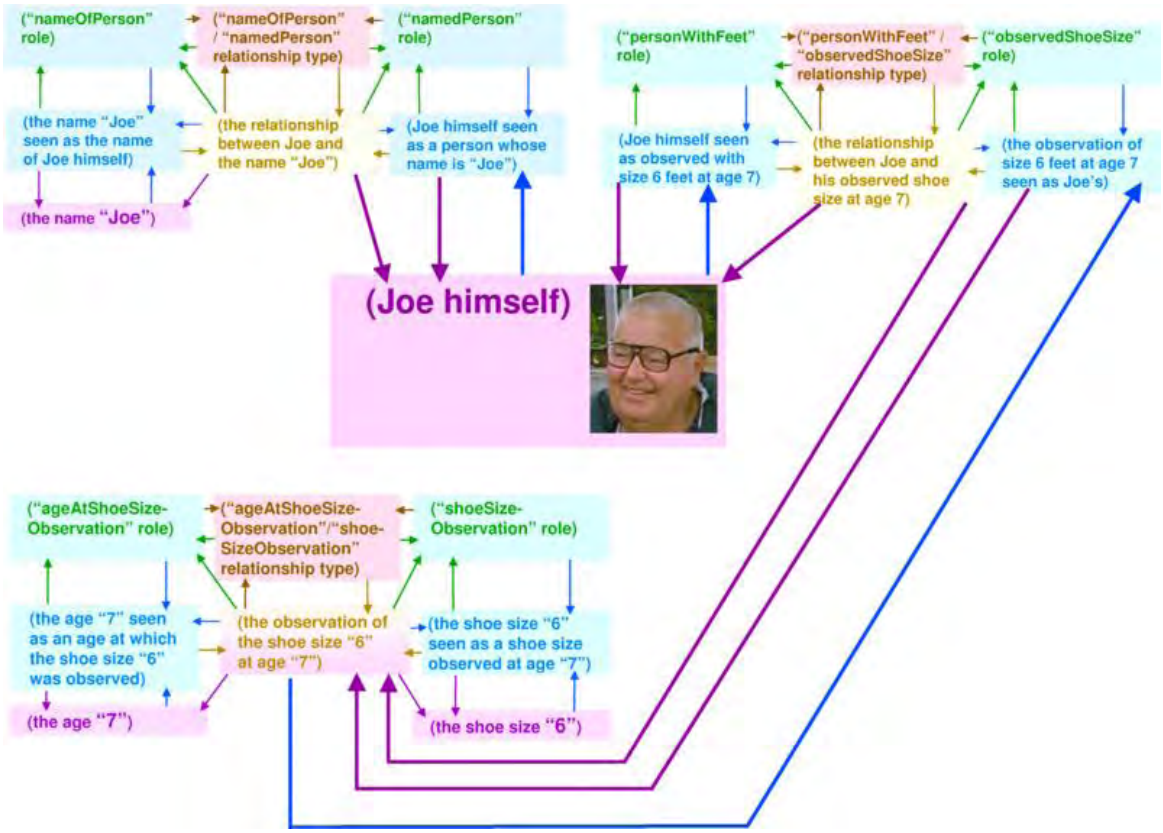
Figure 7



A BigAssert representation of a shoe size observation.

It's interesting that the proxy for the relationship between Joe's shoe size and his age at observation is both a relationship and a role player. After merging, the proxy for this relationship has all the properties of both a BigAssert assertion and a BigAssert roleplayer.

Figure 8



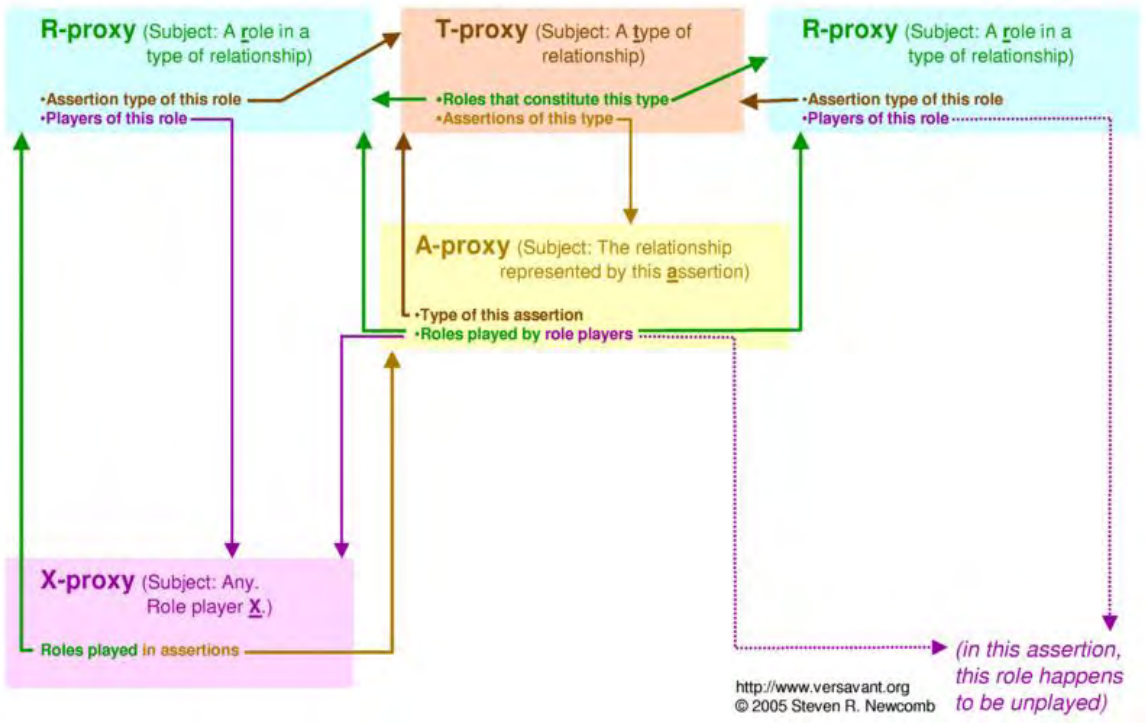
After merging, the proxy for the relationship between shoe size "6" and age "7" is itself a role player in another relationship, one in which Joe is the other role player.

§ Three More Subject Map Patterns for Relationships

TRA_assert

The TRA_assert subject map pattern is much like BigAssert, except that it does not proxify castings. ("TRA" stands for "Type, Role, Assertion", the subjects that it does proxify for each relationship that it represents.)

Figure 9



Conceptual diagram of the TRA_assert subject map pattern.

Figure 10

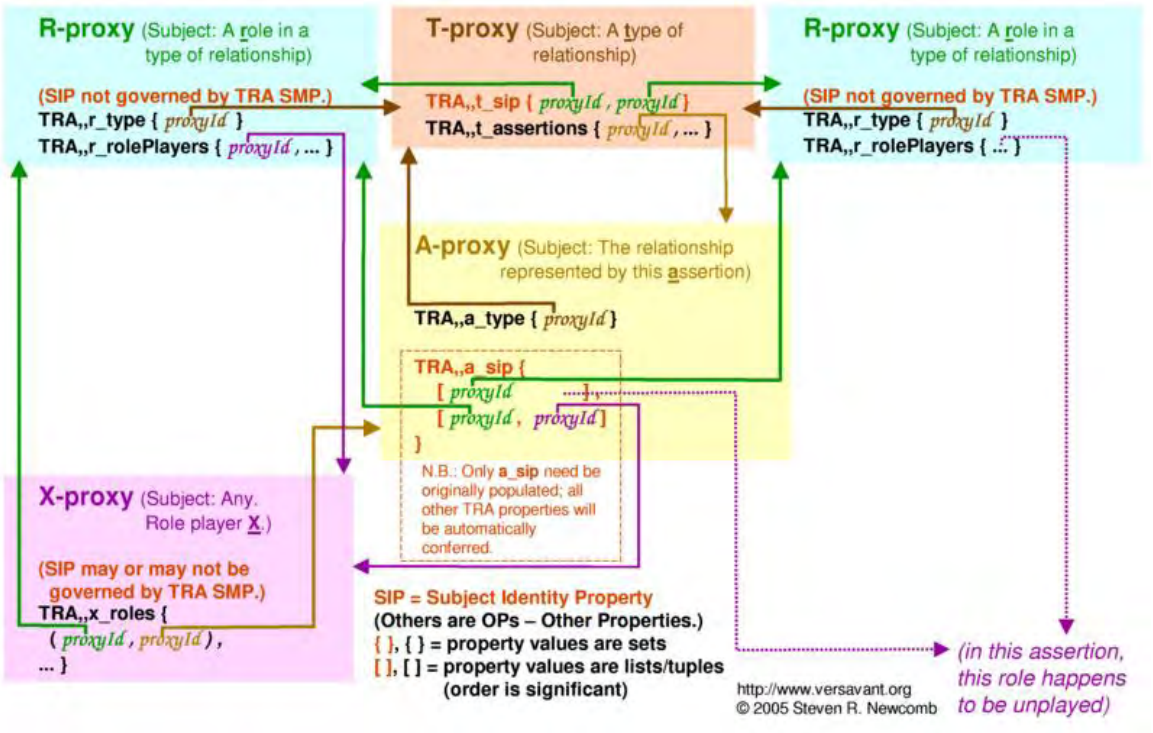
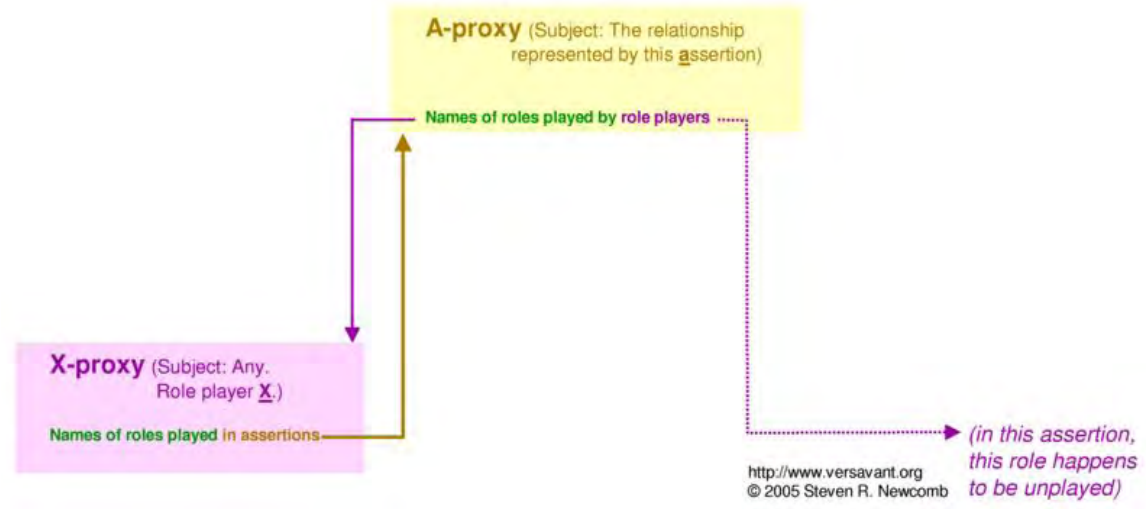


Diagram showing the property classes disclosed for the TRA_assert subject map pattern.

A_assert

The A_assert subject map pattern is significantly different from BigAssert and TRA_assert: it only proxifies assertions. ("A" stands for "Assertion", the only subject that it does proxify for each relationship that it represents.) Role identification is by role name strings; nothing can be said about the roles, at least from the A_assert perspective, because they are not proxified. The relationship type is not proxified, either.

Figure 11



Conceptual diagram of the A_assert subject map pattern.

Figure 12

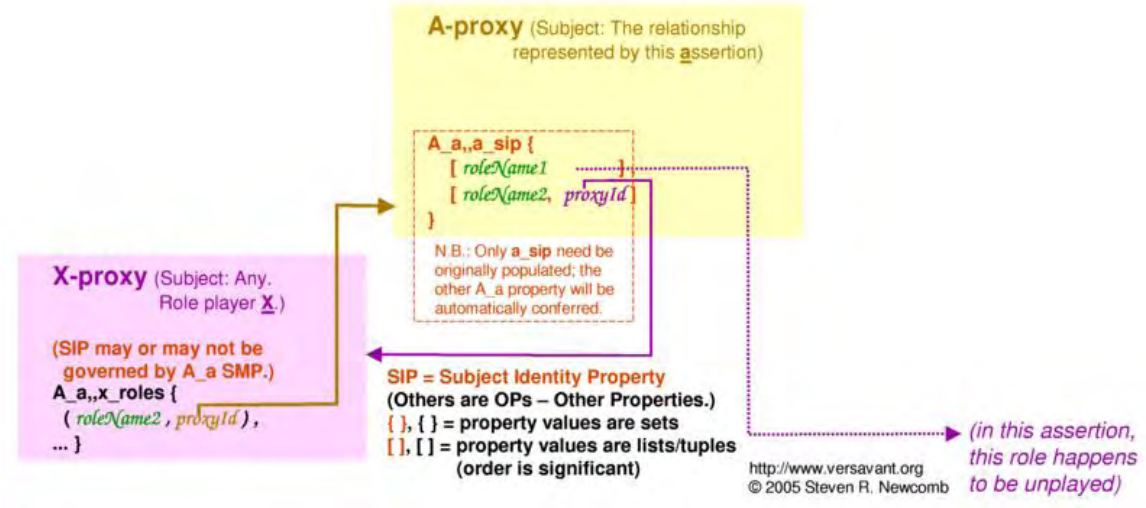
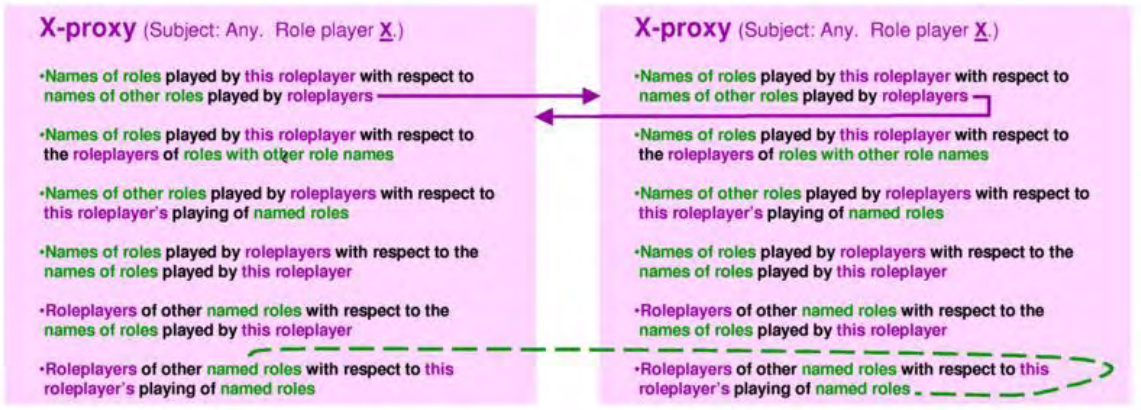


Diagram showing the property classes disclosed for the A_assert subject map pattern.

U_assert

The U_assert subject map pattern is very different from all the others. It doesn't proxify any subjects; none of the property classes that it defines are Subject Identity Properties (SIPs). ("U" stands for "Unproxified" or "Unreified".) The six "Other Properties" (OPs) that it defines are added to each role player. The six properties allow the role players of each U_assert-represented relationship to look each other up in terms of the roles they play, the roles the other role players play, and by the other role players themselves.

Figure 13



Conceptual diagram of the U_assert subject map pattern.

Figure 14

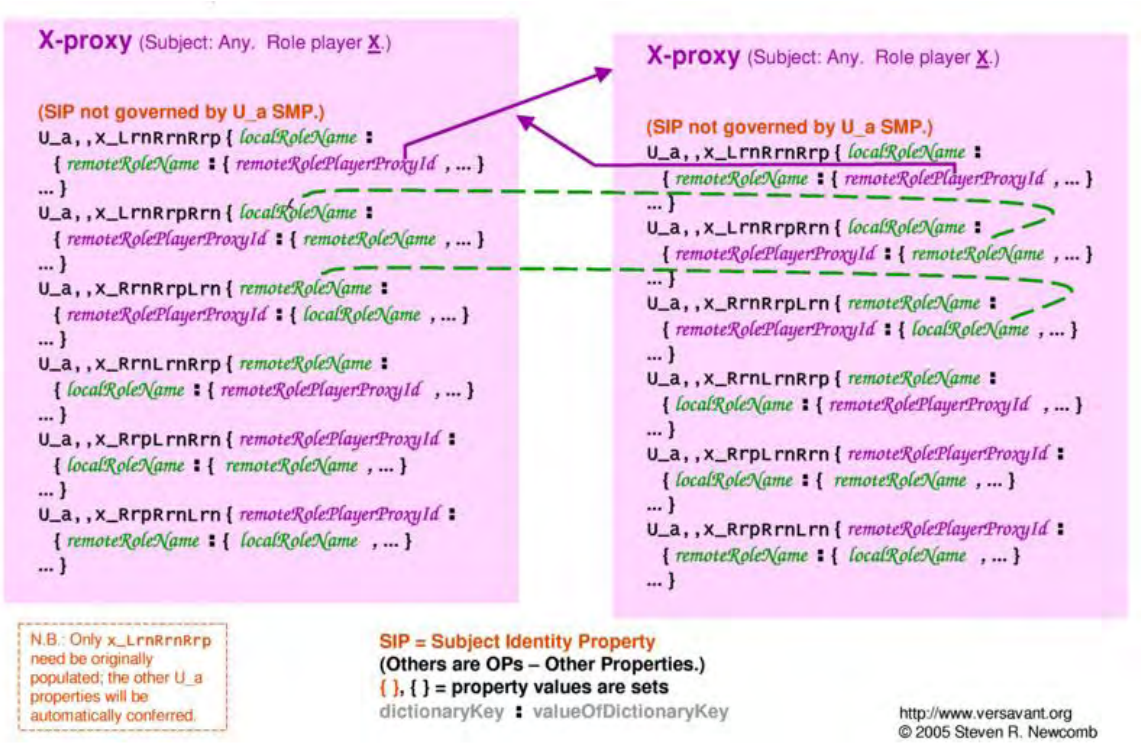


Diagram showing the property classes disclosed for the U_assert subject map pattern.

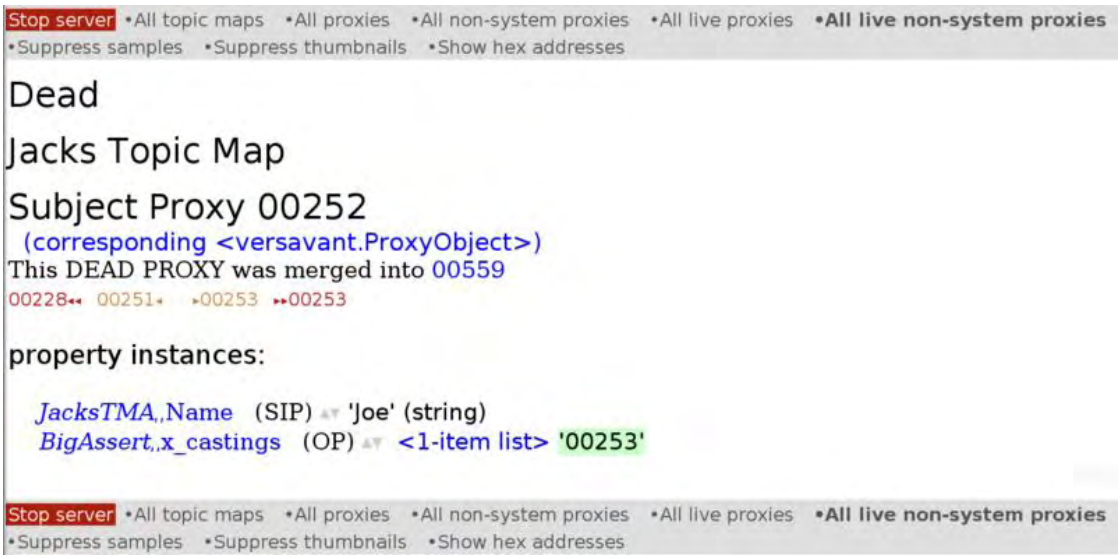
§ Multiple Simultaneous Perspectives on the Component Subjects of Heterogeneously-modeled Relationships

Heterogeneously-modeled Relationships

We created a subject map in which:

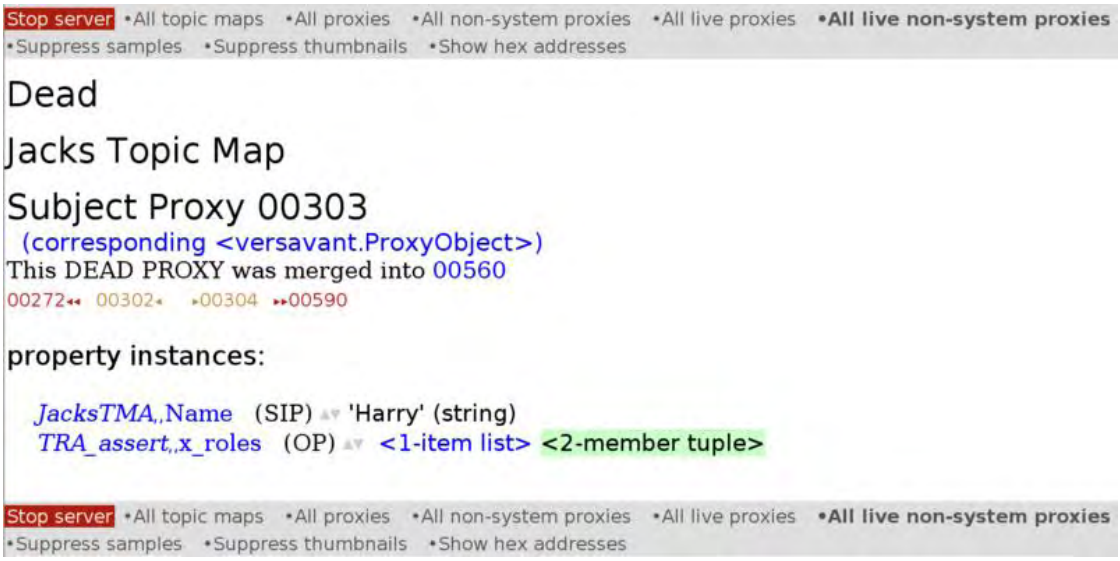
1. Joe participates in the relationships outlined above, represented in terms of the BigAssert perspective -- the BigAssert subject map pattern (see Figure 15.)
2. Harry participates in relationships like those outlined above, but instead of being represented in terms of BigAssert, they are represented in terms of the TRA_assert subject map pattern (see Figure 16.
3. Bartholomew participates in relationships like those outlined above, but they are represented in terms of the A_assert subject map pattern (see Figure 17.
4. Ursula also participates in relationships like those outlined above, but they are represented in terms of the U_assert subject map pattern (see Figure 18.

Figure 15



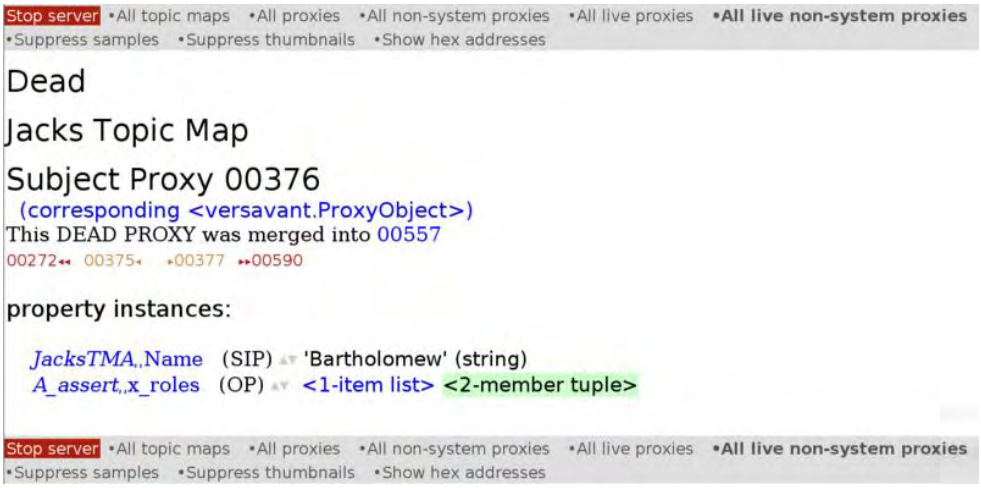
The proxy for the name "Joe", showing the BigAssert-defined property that was conferred upon it by the operation of BigAssert's conferral rules.

Figure 16



The proxy for the name "Harry", showing the TRA_assert-defined property that was conferred upon it by the operation of TRA_assert's conferral rules.

Figure 17



The proxy for the name "Bartholomew", showing the A_assert-defined property that was conferred upon it by the operation of A_assert's conferral rules.

Figure 18

The proxy for the name "Ursula", showing the U_assert-defined property that was conferred upon it by the operation of U_assert's conferral rules.

After Merging, Relationship Subjects with Heterogeneous Perspectives

After the closure of all merging and conferral rules, there is one living proxy for each of the subjects that is each person's name:

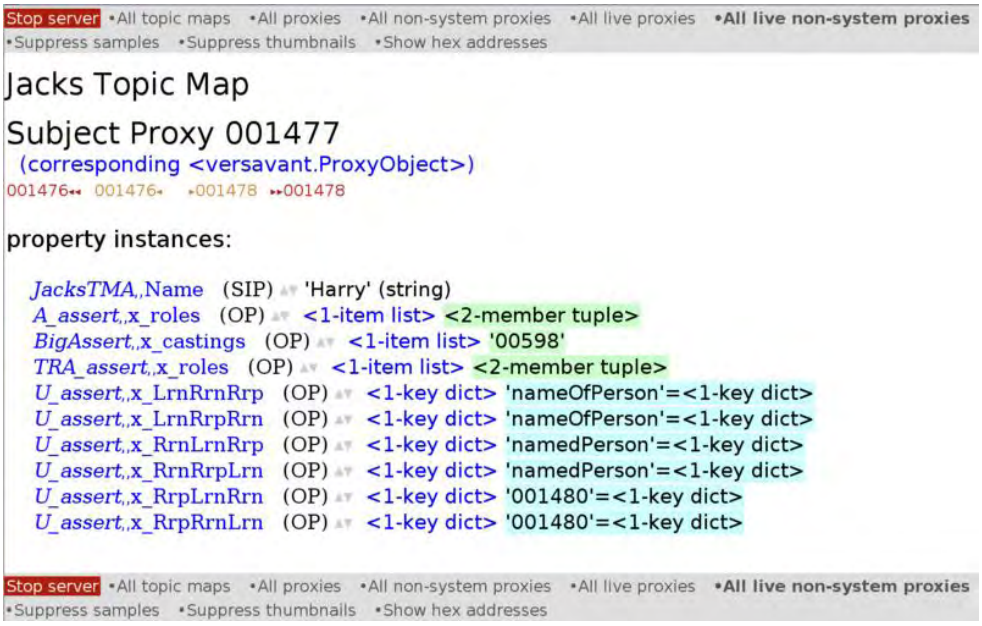
1. The one-and-only proxy for Joe's name (Figure 19) has all the properties necessary to allow it to be seen in terms of all four subject map patterns (BigAssert, TRA_assert, A_assert, and U_assert). Each of the four subject map patterns can read the properties of the other three patterns; each confers its own corresponding properties into existence on the basis of the others.

   What is being demonstrated in this subject map is an approach that makes subject-centric information sharing possible on a cross-cultural, cross-ontology, cross-implementational, and cross-user-interface basis. It doesn't matter what the user needs to see, and it doesn't matter how the information was originally reported. All the cross-mappings represent auditable ontological commitments.
2. Similarly, the one-and-only proxy for Harry's name (Figure 20) has all the properties necessary to allow it to be seen in terms of all four subject map patterns.
3. Similarly, the one-and-only proxy for Bartholomew's name (Figure 21) has all the properties necessary to allow it to be seen in terms of all four subject map patterns.
4. Similarly, the one-and-only proxy for Ursula's name (Figure 22) has all the properties necessary to allow it to be seen in terms of all four subject map patterns.

Figure 19

The proxy for the name "Joe" after merging, showing the properties that were conferred upon it by the operation of the conferral rules not only of BigAssert, but also of the other three subject map patterns.

Figure 20

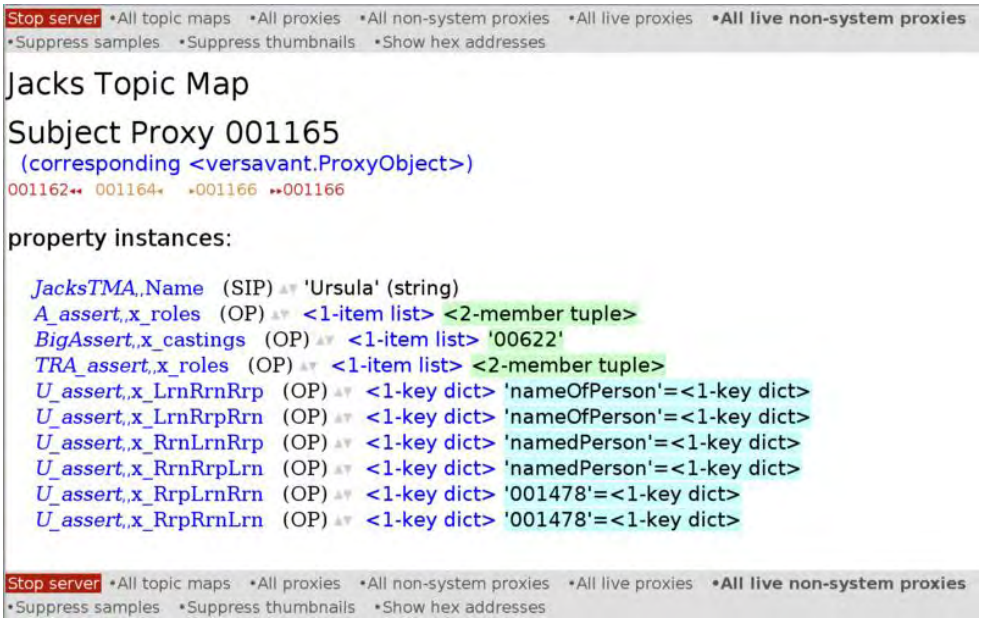


The proxy for the name "Harry" after merging, showing the properties that were conferred upon it by the operation of the conferral rules of all four subject map patterns.

Figure 21

The proxy for the name "Bartholomew" after merging, showing the properties that were conferred upon it by the operation of the conferral rules of all four subject map patterns.

Figure 22



The proxy for the name "Ursula" after merging, showing the properties that were conferred upon it by the operation of the conferral rules of all four subject map patterns.

§ Commercial Interruption

One of the authors of this paper -- the one on the right side of Figure 23 -- is available for modeling work of the kind exemplified in Figure 23. The other author doesn't care much for the way his recent statue (on the left) turned out; he has lost most of his enthusiasm for the idea of extending his modeling career.

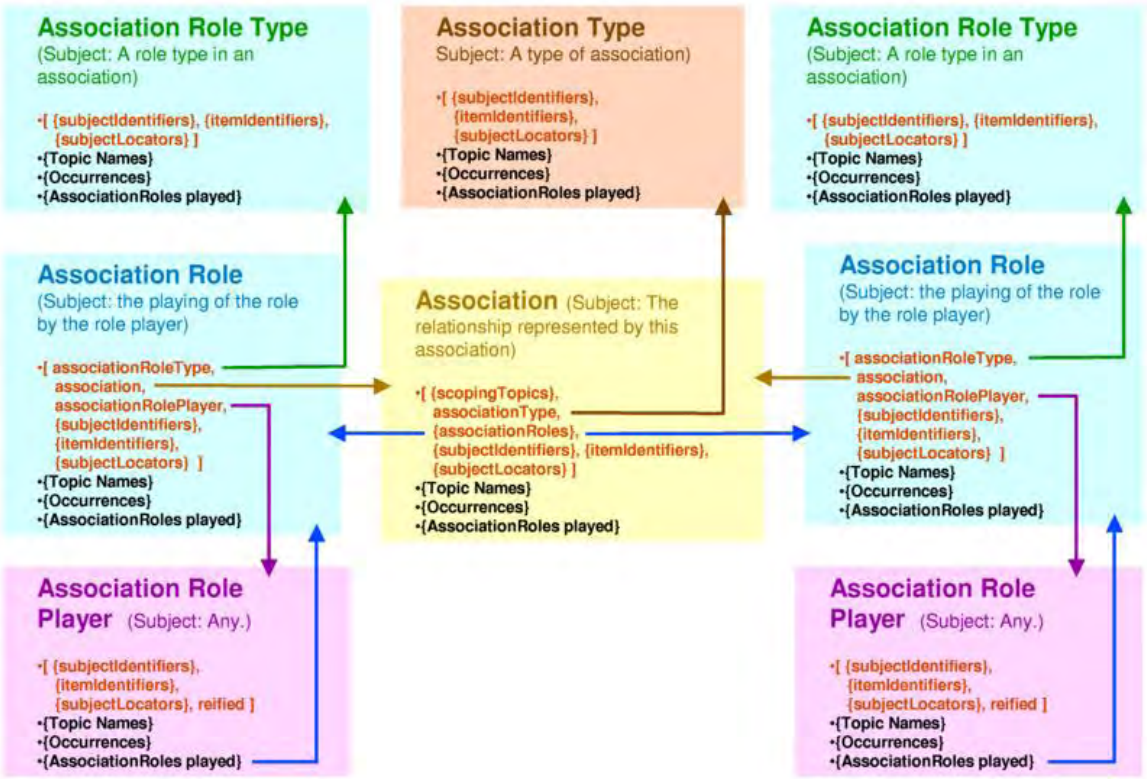
Figure 23

Likenesses of the authors.

§ A Subject Map Pattern as a Component of the Topic Maps Data Model

Relationships can also be represented in terms of the Topic Maps Data Model (TMDM). The TMDM itself implicitly reflects a subject map pattern (see Figure 24) which is similar to, but not quite the same as, BigAssert.

Figure 24



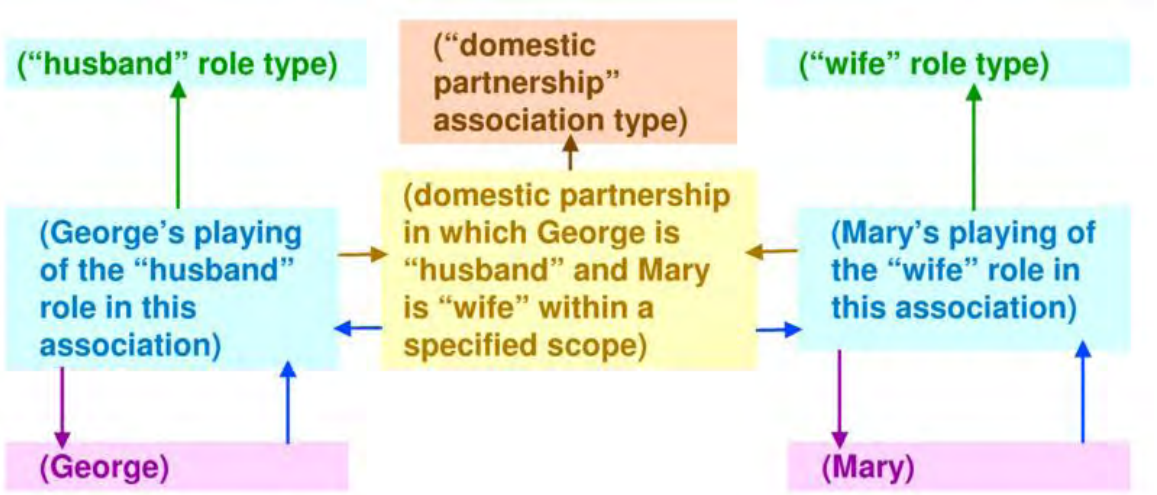
A detailed diagram of a possible way to make explicit the subject map pattern implicit in the "Association Information Item" portion of the Topic Maps Data Model.

Note that in TMDM, the relationship type is an aspect of the identity of relationship subjects, and that there is always exactly one relationship type per relationship. We'll come back to this in a moment.

§ A Use Case for Subject Map Pattern Flexibility

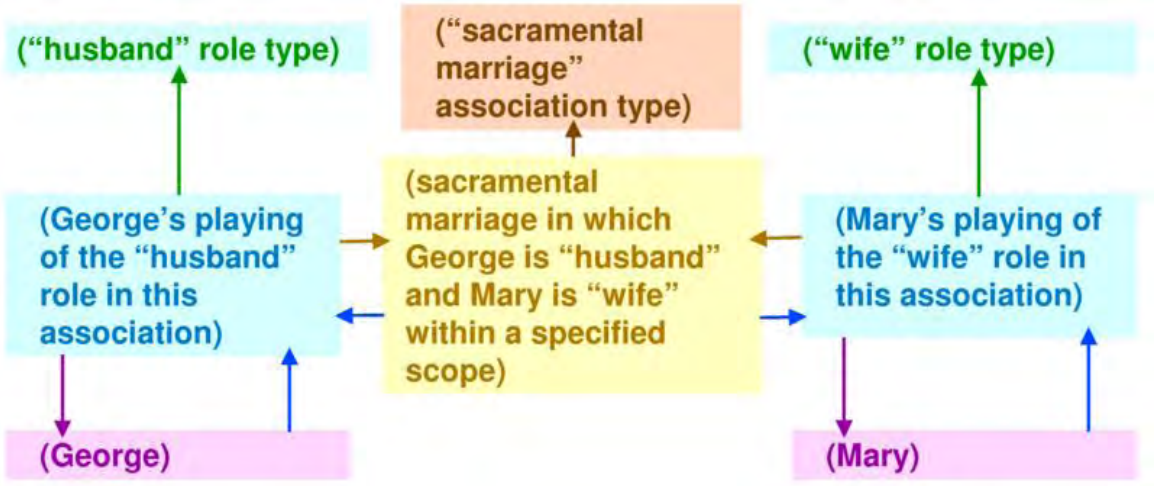
Let us imagine that we are in the healthcare business. We have two sources of public information about marriages, one of which is secular, and the other of which is religious. We are required to keep auditable records regarding all transactions and determinations, and we are permitted and/or required to regard both secular and religious records as potentially authoritative with respect to the existence of marriages and partnerships that are marriage-like. Let us also imagine that it is in our best interests, not only from the perspective of regulatory conformance, but also from a customer-friendliness perspective, to make no distinction between secular and sacred marriages, as long as they are documented. Let us further imagine that, today, we are particularly interested in the marriage-like arrangement between Mary and George, because we need to record our determination of benefits eligibility based on the existence of their marriage. Finally, let us imagine that both our secular and sacred sources report a marriage between George and Mary, as shown in Figures 25 and 26.

Figure 25



The domestic partnership of George and Mary, seen as a TMDM association, as reported by a civil or other secular source.

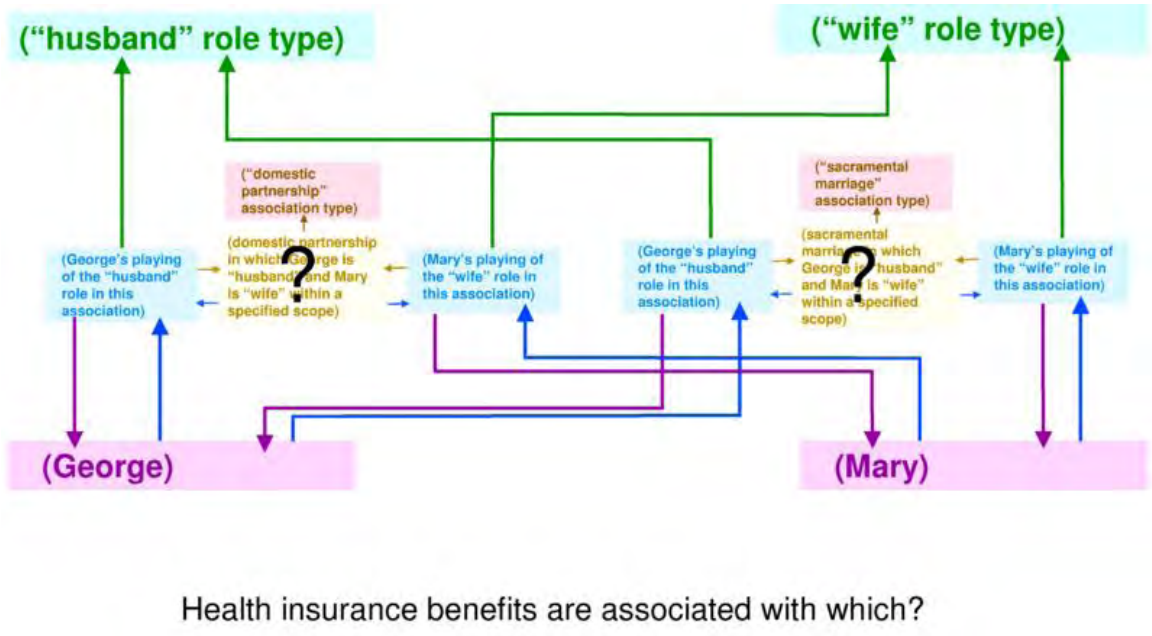
Figure 26



The sacramental marriage of George and Mary, seen as a TMDM association, as reported by an authoritative sectarian source.

Our need to see secular and sacred marriages as not distinct from each other conflicts with TMDM's doctrine on relationship identity, which assumes that different relationship types always mean different relationships. In TMDM, the two relationships, being of different kinds, do not merge. Which of the two relationships should we consider to be the one that justifies our determination of eligibility, and to which our determination should be permanently attached? (See Figure 27.)

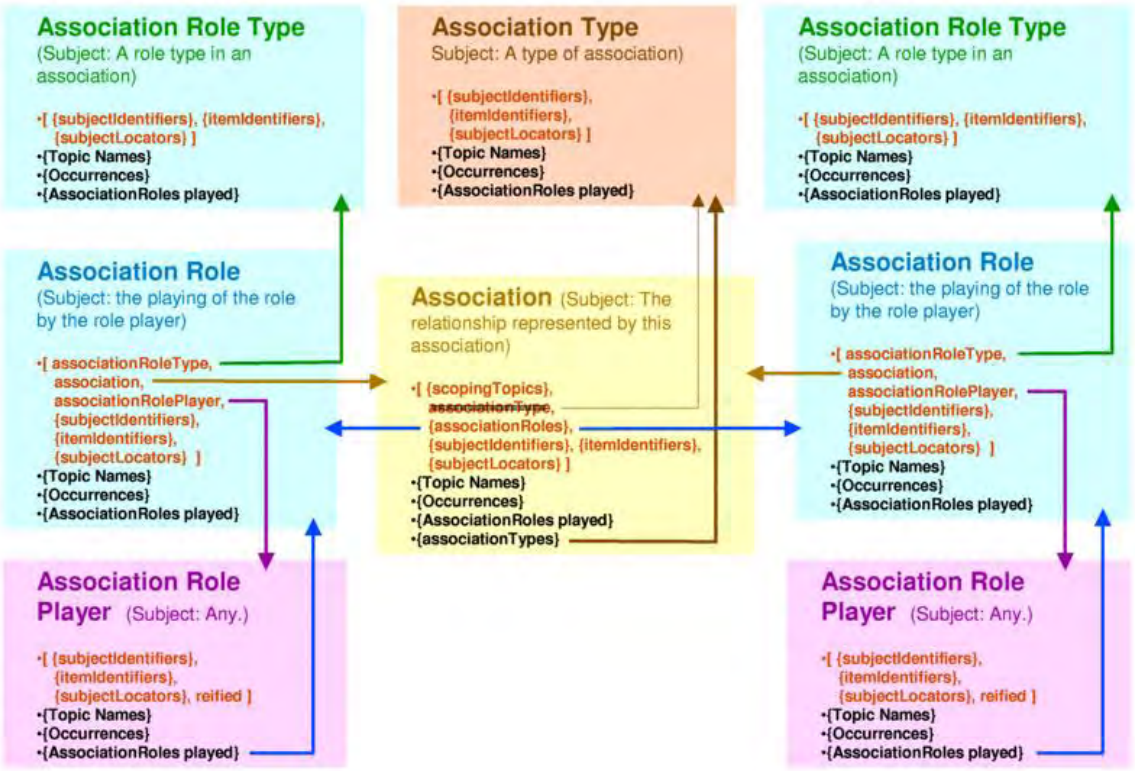
Figure 27



The TMDM forces us to distinguish between the two kinds of marriage, but that's not convenient for us. We'd prefer to see both kinds of marriage as exactly the same relationship, for all purposes of benefits determinations and record-keeping.

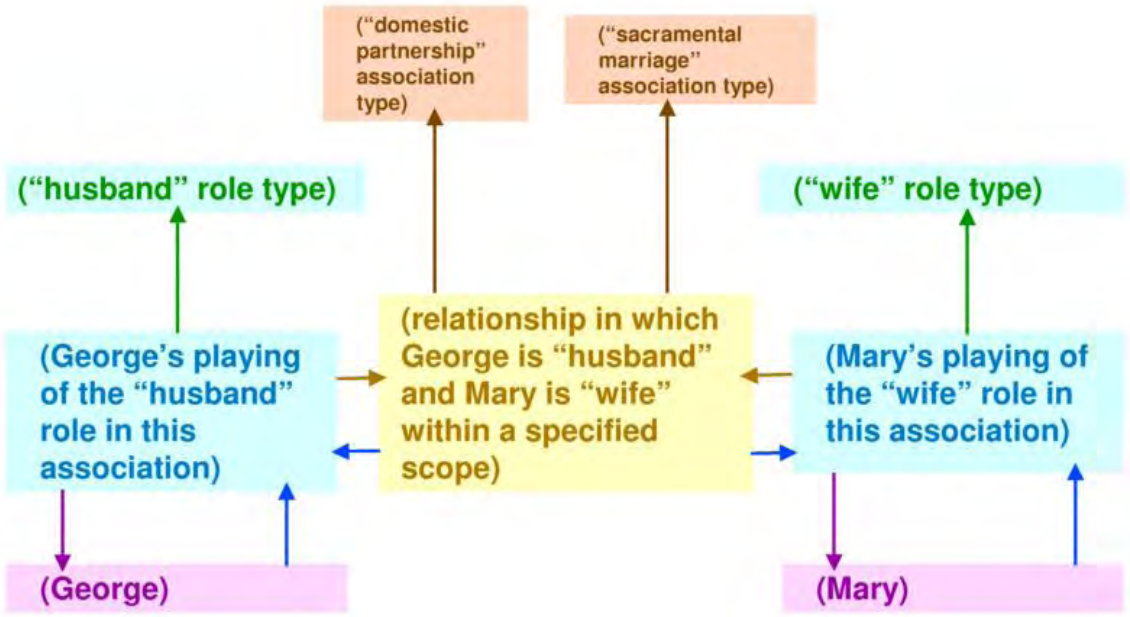
The most elegant answer to our dilemma is to clarify that, in our view of the world, we deliberately ignore all distinctions between secular and sacred marriage-like arrangements, regarding them as the same thing for all purposes of benefits eligibility determination and record-keeping. This view of the world, however, is not in keeping with the TMDM's doctrine of relationship identity. One solution is to create a slightly different Subject Map Pattern (see Figure 28) in which assertion types do not participate in the subject identity properties of proxies for relationships, but instead are the value(s) of an "other property". The result is a single proxy for the marriage of George and Mary -- a proxy that reveals that George and Mary enjoy two kinds of marriage, both of which are considered to be exactly the same relationship for purposes of our subject map (see Figure 29).

Figure 28



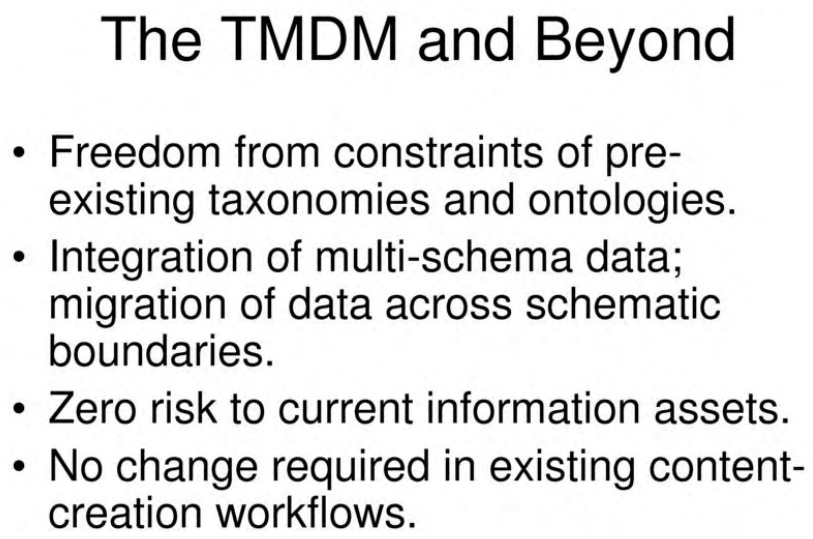
A variation on the subject map pattern for TMDM associations, in which relationships are permitted to have multiple types, and none of those types is involved in any determination of whether multiple proxies are proxies for the same relationship.

Figure 29



We can choose not to distinguish between the two types of marriage, and yet still record their distinct existence.

Figure 30



The modularity, flexibility, auditability and integrity afforded by the subject map pattern paradigm have significant value.

The use of subject map patterns enables the integration of information without requiring the modification of the underlying means by which that information is stored. While that may not seem like a large advantage to the average user or even most implementors, those charged with long term maintenance of such

information stores would loudly disagree. The goal of database and systems administrators is to preserve, without unintentional change, information in their custody. Any write access to or modification of that information, such as conversion, introduces the risk of unintended change to that information.

Subject map patterns enable the "viewing" of information across existing schemas and storage methods, which only requires read access to information. That greatly reduces the risk of unintended change to that information and permits existing information entry methods to continue to add information into the information store. While freedom from existing schemas is an important point to users and developers of new views, information preservation, which underlies any meaningful use of information at all, looms quite large in the view of those charged with with its maintenance.

§ Conclusion

Relationships may be viewed both by their participants as well as those who wish to represent them quite differently. Subject map patterns provide the means for those interested in representing relationships to make explicit choices about what aspects of relationships they wish to represent and ultimately, the means to merge different representations of relationships into a single representation of the subject of such a relationship.

As a results of that flexibility in the representation of relationships, subject map patterns are also capable of "re-representing" information or relationships that have previously been represented differently. That ability means that while the original representations are preserved, that new representations can be created and manipulated by different rules, preserving the integrity of both representations. An added bonus of that flexibility is that system and database administrators need only grant read only access to the information under their stewardship.

Regretably, subject map patterns cannot assist the participants in a relationship in reaching a common view of such relationships, although it can provide the means for them to disclose their respective views of it, if they dare.

The Authors

Steven R. Newcomb
Coolheads Consulting
208 Highview Drive
Blacksburg
VA
srn@coolheads.com

Steven R. Newcomb is an information architecture methodology pioneer, consultant, entrepreneur, and (former) university professor. He drafted and edited the ISO/IEC 13250:2000 and :2003 Topic Maps International Standard, also known as "XTM" ("XML Topic Maps"), and he drafted and co-edits (with the co-author of this paper) the Topic Maps -- Reference Model. He served as editor of the ISO Hypermedia/Time-based Structuring Language ("HyTime", ISO/IEC 10744:1992 and : 1997), and of the ISO Standard Music Description Language (ISO/IEC 10743:1996). He founded and co-chairs the "Extreme Markup Languages" summer technical conference series of IDEAlliance, now in its twelfth year.

Patrick Durusau
Independent Consultant
2149 Conyers Street, SE
Covington
GA
Patrick@Durusau.net

Patrick Durusau has spent the last 15 years involved in a variety of markup projects. Today, most of his energies are focused on topic maps and related technologies both in ISO and OASIS. He serves on the TEI Board of Directors and is the technical lead for the OSIS project (a standard for encoding bibles in XML). He is currently the chair of INCITS V1, the US National Body representative to ISO/IEC JTC1 SC34, which is the committee responsible for SGML, HyTime, DSDL and Topic Maps. He also serves as the chair of the Published Subjects TC at OASIS.

Formerly Patrick Durusau was Director of Research and Development for the Society of Biblical Literature and was the director of the Society of Biblical Literature Font Foundation. He remains interested in the use of markup to enable both display and analysis of Ancient Near Eastern texts and languages.

He was a solo law practioner in Louisiana for ten years, accepting cases that ran from separation and divorce to death penalty litigation.

Extreme Markup Languages 2005®
Montréal, Québec, August 1-5, 2005
This paper was formatted from XML source via XSL
by Mulberry Technologies, Inc.