MultiDomainService

MultiDomainService

MultiDomainService

MultiDomainService

Multi Domain Master Index

Revision History

Date	Version	Description	Author
8/23/08	1.0	Initial version	Swaranjit Dua
9/18/08	1.1	Revised the CRUD APIs	Swaranjit Dua
10/14/08	1.2	Added Merge, Web service	Swaranjit Dua

Table of Contents

1. Introduction

The current Master Index offering provides the capability for supporting Master Data Management for a single domain. While this solution provides a "single version of truth" for each master data entity, it does not provide information of how master data entities relate to each other.

Master Data Relationships and Hierarchies can provide useful enterprise-wide view of the world a person encounters, whether that is business, health-care, on-line shopping, government etc.

1.1

2. Overview

Multi Domain service is the back end service that manages relationships across multiple master indices or domain. The data itself resides in Master Index, however Multi Domain services manages the relationships across the entities in Master Index. Multi Domain service has its own database repository wherein it maintains relationships across entities from different domains or could be across entities within same domain.

This is run time component that is generated from Multi Domain Netbeans project and runs on an Application server as an EJB.

2.1 Definitions

Domain : A complex entity and business rules that solves a particular business problem.

Ex. Customer Domain, Product Domain.

Currently, a domain corresponds to a Master Index project.

Relationship: A link between two master data entities. Relationship can be within a single domain or across domains.

The two entities are “source” and “target”. By definition Relationship implies Relationship instance.

Multi Domain Service: Back end service manages Master Data relationships, hierarchies and groups across multiple master indices. A multi domain service may be configured with many domains, across whose relationship will be managed.

Relationship Type: Template for linking relationships within or across domains.

Relationship Type has fixed attributes like roleName, source, target domain, and list of extended attributes that differ from one Relationship Type to another type.

Each relationship is using one of Relationship Type and so may have different set of values for the set of extended attributes.

Hierarchy: A Special kind of relationship in which “target” is linked with only one “source”.

This is “Parent-child” relationship. In other words, a Parent can have many children but a child can have only one parent.

Group: A group can act as an entity that performs a particular function like a group of people playing the role of a customer.

It can be formal (business, organizations, legal entities). It has similar attributes to that of a entity that it represents.

There is one to many relationship between a group and members, as in a household group.

Dun#: A unique id that identifies a company, is used in CDI market space.

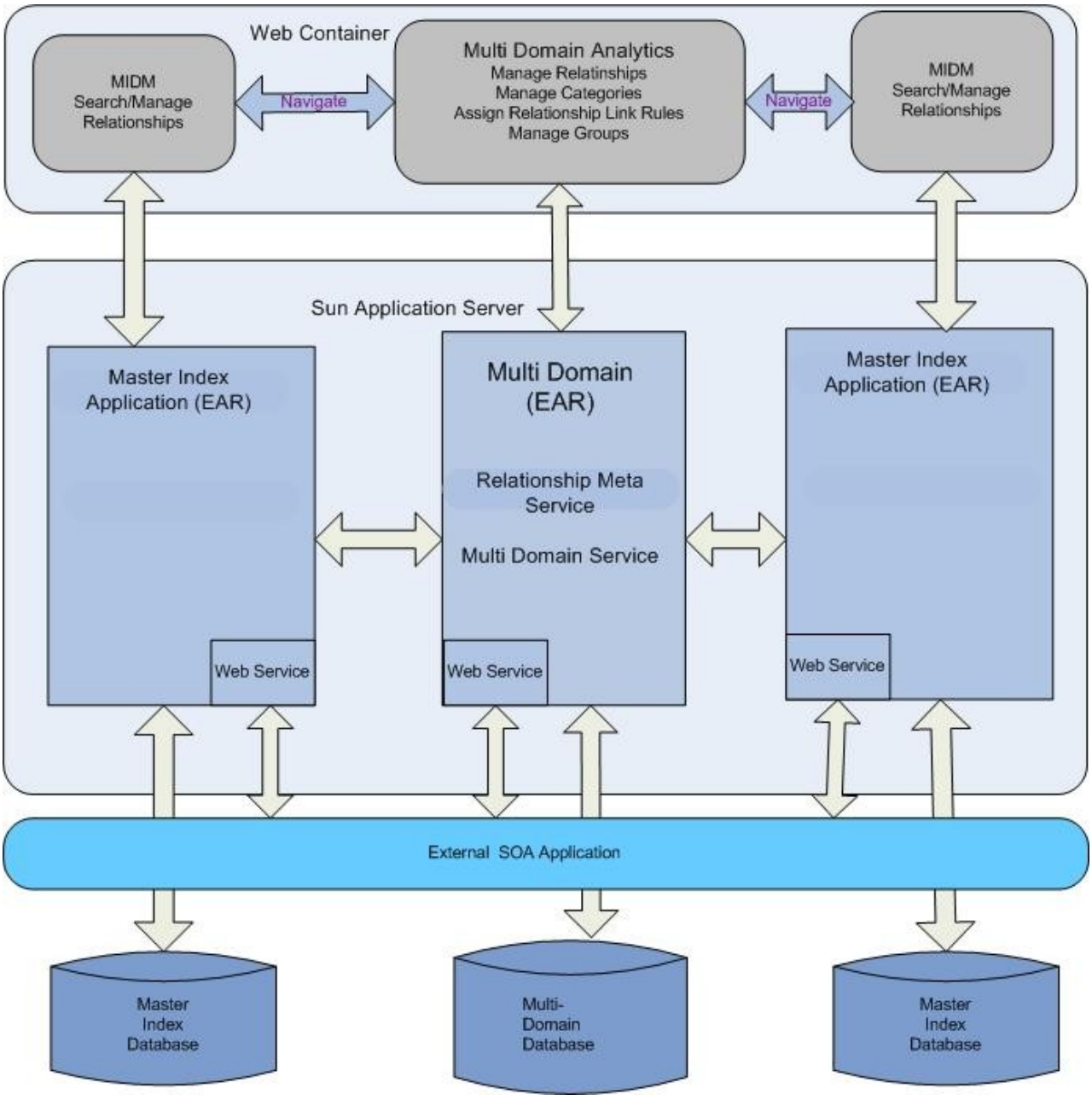
2.2 Key Requirements

Please refer to Functional Spec – at https://open-dm-mi.dev.java.net/specs/relman/FuncArchSpec.htm

2.3

3. Technical Description

3.1 Component Architecture



3.1.1 Database Design

Relationships aka Relationship instances would be stored in two tables:

The columns are:

relationshipid, sourceDomain, sourceuid, targetDomain, targeteuid, relationshipTypeid, valueId, startDate, endDate, purgeDate

Relationship (Relationship instances)

relationshipID **PK**	sourceuid	targeteuid	relationshipTypeId **FK** (to RelationshipType)	Type: Hierarchy/Relationship/Group	valueID **FK** (to exAttributesValues)	StartDate	EndDate	purgeDate
1	0001	0003	r1	1	ev1	1/1/08	5/1/08	

The attribute values for a relationship are stored in RelationshipAttributeValues.

RelationshipAttributeValues (created one time during multi-domain setup - number of columns in this table would be decided once only during installation)
contains values for extended attributes for all relationships and their types.
multiValue columns would store values for more than 1 attribute.

valueID **PK**	RelationshipTypeId **FK**	attrValue1	attrValuen	multiValue1	.. multiValuen	
ev1	m1					

attrValue columns created can be of any type such as int/varchar2/date/

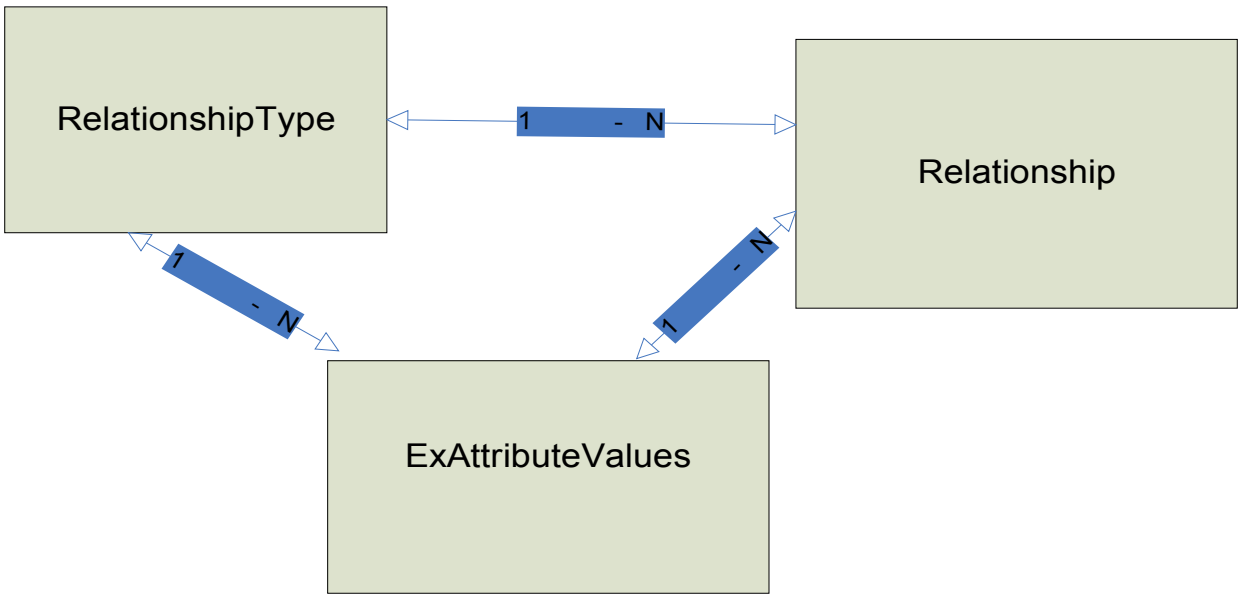
Note: Multi value column feature is not supported in the 1st release.

Similarly Group instance data would be stored in “Group” table and GrouopAttributeValues.

Hierarchy instance data would be stored in “Hierarchy” and “HierarchyAttributeValues” table.

RelationshipType, metadata, is stored in two tables: RelationshipType and AttributeMap table.

ER Diagram



3.2 Interfaces

Multi-Domain service is an EJB generated for each Multi-domain project.

We'll separate Multi domain service into two services or EJBs - MultiDomainMetaService for relationship Meta Data and MultiDomainService for relationship instance data.

MultiDomainMetaService contains operations related to RelationshipType that are performed using Administrator privileges

Note: In general all members are private and methods shown are all public. Some getters/setters are not shown for brevity.

All methods throw MultiDomainException.

public interface **MultiDomainMetaService** {

```
/**
  get all domain names
*/
    String[] getDomains();

     String[][] getDomainPairs();

/**
  get all RelationshipType from all domains
*/
    RelationshipType[] getRelationshipTypes();

/**
   get Relationship Types for given source and target domain
 */
    RelationshipType[] getRelationshipTypes(String sourceDomain, String targetDomain);

/**
  persist RelationshipType in the database repository.
  @param relationshipType input
*/
    String createRelationshipType(RelationshipType relationshipType);

 void updateRelationshipType(RelationshipType relationshipType);

  /**
   get Hierarchy Types for given domain
  */

   HierarchyType[] getHierarchyTypes(String domain);

  /**
    create a new HierarchyType based on input relationshipType.
    @return relationshipTypeID
  */
   String createHierarchyType(HierarchyType hierarchyType);
```

```
 /**
    updateHierarchyType
*/
   void updateHierarchyType(HierarchyType hierarchyType);

void deleteHierarchyType(String domain, String name);

void deleteHierarchyType(String hierarchyTypeid);

/**  creates a new group Type.
  returns groupTypeId
*/
  String createGroupType(GroupType groupType);

   void updateGroupType(GroupType groupType);
/**
  returns relationshipid
 */

  void deleteGroupType(String domain, String groupName)

  GroupType[] getGroupTypes(String domain)

}

public interface MultiDomainService {
 /**
   create relationship in the database that associates the two EUIDs.
@param relationship is the relationship attribute and their values that are associated with this relationship.
 @return relationshipid that identifies the new relationship established.
*/
String createRelationship(Relationship relationship)

/**
This API can be called from source systems to create relationship between two local system records, where
EUID for these records may be unknown.
*/
```

String **createRelationship**(String sourceSystemcode, String sourceLID, String targetSystemCode, String targetLID, AssociationValue association)

/**

create Relationship between objects which are identified using MultPairValue.

MultiPairValue represents EUID based on field and values such as Dun#, when EUID for source and target is unknown. So based on MultiPairValue, EUIDs can be queried from domain and then the relationship is established.

Relationship identifies the relationship attributes and values that are associated with this relationship.

@return relationshipid that identifies the new relationship established.

*/

String **createRelationship**(MultiPairValue sourceMultiValue, MultiPairValue targetMultiValue, AssociationValue relationship);

/**

update relationship

*/

void **updateRelationship**(Relationship relationship);

/** delete the relationship between sourceEUID and targetEUID and identifiied uniquely by sourceEUID, targetEUID, sourceDomain, targetDomain, relationshipName

*/

void **deleteRelationship**(String sourceEUID, String targetEUID, String sourceDomain, String targetDomain, String relationshipName);

/** delete the relationship that is identifiied uniquely by sourceSystemCode, sourceLID, targetSystemCode, targetLID, sourceDomain, targetDomain, relationshipName.

*/

void **deleteRelationship**(String sourceSystemCode, String sourceLID, String targetSystemCode, String targetLID, String sourceDomain, String targetDomain, String relationshipName);

/**

This method searches source domain and brings back the Master Index records and the records to which they have relationship, that qualifies the input search criteria. During search this will bring relationship records from all the Master Indices that are specified in searchOptions and that have relationships with primary domain records.

searchOptions contains epaths for fields across domains that needs to be retrieved.

SearchCriteria contains SystemObjects to which search criteria can be specified. The non-null fields in SystemObject are used as filter fields and combined using “AND” filter.

Currently search criteria for only source domain can be specified.

*/

PageIterator<MultiObject> **searchRelationships**(String sourceDomain, MultiDomainSearchOptions searchOptions, MultiDomainSearchCriteria searchCriteria)

/**

This method searches source domain and brings back the Master Index records and the records to which they have relationship, that qualifies the input search criteria. During search this method would go to the target domain as well to retrieve target records.

searchOptions contains epaths for fields across domains that needs to be retrieved.

PageMultiIterator<MultiObject> **searchRelationships**(String sourceDomain, String targetDomain, EPathArrayList[] sourceEPathList, EPathArrayList[] targetEpathLIst, MultiSearchOptions searchOptions)

*/

/**

This method searches different Master Indices and brings back the Master Index records that qualifies the input search criteria. It does not search/retrieve information from the relationship tables.

*/

PageIterator<ObjectNode> **searchEnterprises**(String sourceDomain, MultiDomainSearchOptions multiDomainSearchOptions, MultiDomainSearchCriteria multiDomainSearchCriteria)

String **createHierarchy**(Hierarchy hierarchy);

/**

create Hierarchy between parent and children that are identified using MultiPairValue.

MultiPairValue are used where EUIDs are unknown and

parentFieldValues and childFieldValues are used to query EUIDs based on fields such as dun#.

*/

String[] **createHierarchy**(MultiPairValue parentFieldValues, MultiPairValue[] childFieldValues, AssocationValue attributeValue)

void **deleteHierarchy**(String hierarchyTypeid)

void **updateHierarchy**(Hierarchy hierarchy);

/**

search Hierarchy for an input EUID. The returned object contains all the ancestors for the input EUID as well as immediate children. The fieldsToRetrieve contain all the fields that will be retrieved for all the ancestors and immediate children.

*/

HierarchyObjectTree **searchHierarchy**(String hierarchyTypeid, String euid, EPathArrayList fieldsToRetrieve);

/**

create a group whose attributes values are defined in relationship

@return groupid that is created

*/

String **createGroup**(Group group);

/**

create Group members

@param relationship are assoicated with a groupid and EUIDs.

@return list of relationshipid strings that are for relationship between EUIDs[] and groupid.

*/

String[] **createGroupMembers**(GroupMember[] groupMembers) ;

deleteGroup(String domain, String groupid)

updateGroup(Group group)

/**

search list of groups.

@param Relationship attribute values that at non-null are used as filter condition to find the groups that match this criteria.

@ return list of group that are represented by Relationship.

*/

```
public Group[] searchGroup(Group group);
```

/**

get Group Members that are associated with input groupid. Fields represent list of fields to retrieve for all the members EUIDs.

@return ObjectNode[] list of object nodes that have relationship with input groupid.

*/

ObjectNode[] **getGroupMembers**(String groupid, EpathArrayList fieldsToRetrieve);

/**

get list of groups that are assoicated with input euid. An EUID or SBR can be associated with multiple groups. So this API returns list of all such groups.

```
        */
        Group[] getGroups(String euid)


         /**
              This will invoke RelationshipMerge merge plugin method, if present.
         */
         merge(String domain, String mergedEUID, String survivorEUID);

        /**
              This will invoke RelationshipMerge unmerge plugin method, if present.
         */
         unmerge(String domain, String unmergedEUID, String survivorEUID);
    }


      public class MultiDomainSearchOptions {
            /**
                  List contains fully qualified field name FQFN  from different domains
              Ex.: Person.firstName
                   Product.partNumber
                   Every element of epathLists array should have Epaths for each different domain.
                   This list is used in case of relationship  of primary domain object with other fields
            */


            EPathArrayList[] epathLists;   /* EpathArrayList utility is defined in Master Index */
            int maxElements;  /* Max elements to be retrieved */
            boolean weightedOption;  /* Whether to use weighted option */
            String searchId; // QueryBuilder search Id.
             int pageSize;
      }

/**
  search criteria. It is resented by non-null field values in the SystemObject.
*/
            public class MultiDomainSearchCriteria {
```

```
    String relationshipTypeName;
    SystemObject primaryDomainObject;
/* SystemObject is the primary Object that is used for defining search criteria. It is similar to Master Index search Criteria.
   This criteria is in turn delegated back to Master Index search method. Any field that has non-value in SystemObject and its children objects, is used as filter value. All such filter values are used in composite AND clause to query the Master Index. */
    Relationship relationship; // the relationship attribute values will be used for filtering relationship
                               // that are “AND” with primaryDomainObject attribute values.

    SystemObject[] secondaryDomains;
}

 public class PageIterator<E > {
   private int count;
   public <E> t next();
}

/**
   Contains top level primary object for primary domain, and the top level objects from other domains to which primary object has a relationship.
 */
public class MultiObject {
  ObjectNode primaryDomainObject;
  /** All the objects that are related to primaryDomainObject.
      These can be from different domains
   */
  RelationshipObject[] relObjects;
}

/**
    Contains target and source Object that have relationship. When this object is part of MultiObject, sourceObject points to primaryDomainObject in MultiObject.
  */
public class RelationshipObject {
   private ObjectNode targetobject;
  private Relationship relationship;
}
```

```
public class HierarchyObject {
    private ObjectNode parent;
    private Object child;
    private Hierarchy hierarchy;
}

 /**
     Represents an hierarchy of an object. Contains an object, all its ancestors and all its immediate children.
  */
 public class HierarchyObjectTree {
    /** current object */
    private ObjectNode object;

   /**
        All the ancestors of an object up-to root. The ancestors of an object are in order of position in the array. So the parent is the 1st element in the array. The root is the last element of the array.
    */
    private ObjectNode[] ancestors;

    /** All the children of an object
    */
    private ObjectNode[] children;
}

/**
    RelationshipType represents a template of attribute names and their properties that are used during run time association of Relationships.
*/
 public class RelationshipType {
      String name;
      String relTypeId;  // primary key,  unique across domains,  used during run time Relationships
      int  type; // 1 for Relationship, 2 for hierarchy, 3 for group
      String displayName;
      String sourceDomain;
      String targetDomain;
      String sourceRoleName; //
      String targetRoleName;
      int direction; // 1 one direction, 2 bidirectional
```

```
        boolean includeEffectiveFrom;
        boolean includeEffectiveTo;
        boolean includePurgeDate;
        boolean effectiveFromRequired;
        boolean effectiveToRequired;
        boolean purgeDateRequired;
        /*  list of all attributes */
        List <Attribute> attributes;
    }

    /**
     An extensible Attribute associated with a RelationshipType.
     Conceptually it is like a field or a database column name.
     Contains set of properties.
    */
    public class Attribute {
     String name;
     String displayName;
     String columnName;
     boolean searchable;
     boolean isRequired;
     String defaultValue;
     String dataType;
    }

      /**
         The list of values that are assigned to list of attributes.
         These are persisted in exAttributeValues table
       */
      public class AttributeValues {
         String relationshipTypeName;
         String domain;
         Map <Attribute, String> values;

         List<Attribute> getAttributes();
         /**
             get a Map for attribute names and the values.
```

```
     */
    Map<Attribute, String> getAttributeValues();
    String getAttributeValue(String name);
}

/**
    Relationship or Relationship instance.
   Contains Attribute values for a RelationshipshipType.  Attribute Values can be shared across multiple relationships.
   The fixed attributes are stored in Relationship table and extensible attribute values are stored in exAttributeValues table.
  */
public class Relationship  {
        RelationshipType relType;
        String id; // unique value of relationshipid across all domains.
        Date startDate; /// start date for a relationship
        Date endDate;  // end date for a relationship
        Date purgeDate;
        AttributeValues attributeValues;
 }



/**
   List of field and values that are used for filter values during search.
*/
public class MultiPairValue {
 FieldValuePair[] values;
}

public class FieldValuePair {
        String fieldName;        /* Fully qualified field name. e.g. Company.Dun */
         String value;    /* value e.g. 11122333  */
}

/**
    This class is responsible for Relationship extensible attributes persistence and search.
     The attribute values are persisted in ExAttributeValues table.
 */
```

```
public class  RelationshipAttributesRepository  {
    void insert(Relationship relvalues);
    void update(Relationship relvalues);
    void delete(String relationshipid);
    Relationship[]  search(String relationshipTypeName, Map<String, String>);
}
```

Master Index additions- Generally a search method from MultiDomainService would use Master Index MasterController searchEnterprise method to do searches on Master Index. Currently the filtering supported in searchEnterprise method is based on filter conditions on SystemObject fiields that uses AND operator to create criteria.

However Master Controller search need to support searches that are based on “IN” clause. The IN clause would take a list of EUIDs passed to it, or can take a list of any other field values passed to it. e.g. IN clause would take list of Dun#. This list of Dun#s is used during hierarchy creation.

3.2.1 Merge/Unmerge

Assuming that there are 2 customer record instances.
Record 1 has relationship to products Computer and Monitor
Record 2 has relationship to product Computer

T0: Original

C1 John Smith ->> CustomerOf (3/23/2007, 4/15/2008) --> P1 Computer

->> CustomerOf (1/18/2008, 3/15/2008) --> P2 Monitor

C2 John Smith ->> CustomerOf (12/23/2008, 6/15/2008) --> P1 Computer

T1: Merge C1 into C2:

All relationships C1 has that C2 does not is moved to C2:

C2 John Smith ->> CustomerOf (1/18/2008, 3/15/2008) --> P2 Monitor

For relationships that both C1 and C2 have, we provide an optional plug in to allow for conflict resolution.
If no plug in is defined, than C2's relationship wins.

e.g. with plug in that combines the date:

C1 John Smith ->> CustomerOf (3/23/2007, 6/15/2008) --> P1 Computer

e.g. with no plug in present

C2 John Smith ->> CustomerOf (12/23/2008, 6/15/2008) --> P1 Computer

In case of unmerge, such as when EUID2 is unmerged from EUID, the plugin will be invoked passing the current relationship EUIDs to plugin and plugin can if needed re-compute new relationships for both EUID1 and EUID2.

Plugin For merge/unmerge

```
public interface MergeRelationship {
        /**
         RelationshipPair, list for the relationships that are in common with both mergedEUID and survivorEUID are passed as arugments to merge that returns Relatioship[] which resolve the conflicts between same Relationship pair.
         Conflict relationship means either source EUID for both mergedEUID and survivorEUID are same or
         target EUID is same or source/target of mergedEUID and target/source of survivorEUID are same.
        */
         Relationship[] merge(String domain, String mergedEUID, String survivorEUID, RelationshipPair[] conflictRelationships);

         Group[] merge(String domain, String mergedEUID, String survivorEUID, GroupPair[] conflictRelationships);

        /**
         This method can either re-compute new relationships between mergedEUID and survivorEUID.or use passed survivorRelationships before the unmerge operation to compute relationships for unmerged and survivorEUID..
         */
         RelationshipListPair  unmerge(String domain, String unmergedEUID, String survivorEUID,  Relationship[] survivorRelationships, <handle to Master Indices> );

}

public class RelationshipPair {
   Relationship rel1;
   Relationship rel2;
}

public class GroupPair {
   Group gp1;
   Group gp2;
}

public class RelationshipListPair {
  Relationship[] rel1;
  Relationship[] rel2;
```

}

3.2.2 Web Service template generated classes:

For each domain, the primary object class will be generated that will be different than current primary class.
Each domain class will have getters and setters for each of its domain pairs in addition to getter/setter for primary domain.
So say Customer domain has target domains with Product and Account. This will generate:

```
CustomerRBean {
    CustomerBean getPrimary();  //  primary object
    ProductRelationship[] getProduct();
    AccountRelationship[]  getAccount();
}
 ProductRelationshp is a generated class

 public class ProductRelationship {
   Relationship getRelationship();
   ProductRBean getProduct();
}

So the template classes are:
public class <Domain>RBean {
     <Domain>Bean   getPrimary();  // <Domain>Bean is a Master Index web service generated class.
  <TargetDomain1>Relationship[]  get<TargetDomain1>();
  <TargetDomain2>Relationship[]  get<TargetDomain2>();
}

 public class <Domain>Relationship {
   Relationship getRelationship();
   <Domain>RBean get<Domain>();
}


public class  <Domain>RBeanScore {
 float score;
```

```
  String euid;
  <Domain>RBean rbean;
}

/**
    Represents an hierarchy of beans. Contains an object, all its ancestors and all its immediate children.
*/
public class <Domain>HierarchyBean {
 /** current object */
   private <Domain>Bean bean;

  /**
    All the ancestors of an object up-to root. The ancestors of an object are in order of position in the array. So the parent is the 1st element in the array. The root is the last element of the array.
   */
   private <Domain>Bean[] ancestors;

    /** All the children of an object
     */
     private <Domain>Bean[] children;
 }
```

3.2.3 Web services API template

```
   public <Domain>RBeanScore[] searchExactRelationship(<Domain>RBean rbean)
   public <Domain>RBeanScore[] searchBlockRelationship(<Domain>RBean rbean)
   public <Domain>RBeanScore[] searchPhoneticRelationship(<Domain>RBean rbean)
   public <Domain>HierarchyBean search<Domain>Hierarchy(String euid);
   public <Domain>Bean[] get<Domain>GroupMembers(String groupid);

  public  Search<Domain>Result searchEnterprises(<Domain>Bean bean)
                /*Search<Domain>Result and <Domain>Bean template classes are  defined in Master Index web service generated classes.*/
```

The rest of methods will same as in MultiDomainService

For each domain these methods will be generated such as

public CustomerRBeanScore[] searchExactRelationship(CustomerRBean rbean)

public CustomerRBeanScore[] searchBlockRelationship(CustomerRBean rbean)

CustomerRBeanScore is generated class

```
public class CustomerRBeanScore {
  float score;
  String euid;
  CustomerRBean rbean;
}
```

4.