A MULTI–PARADIGM MAPPING METHOD SURVEY [1]

Marcel Verhoef [2], Thomas Liebich [3], Robert Amor [4]

ABSTRACT

Recent research into integrated engineering applications has shown that the definition of a shared product model only solves part of the data exchange problem. When product data models are taken from their modelling environment to be used in practice, integration problems occur due to the unavoidable semantic mismatch between the models, representing the different actors in the system. To solve this problem it is necessary to support multiple views on the application domain from within this integrated system. Futhermore, when an integrated system is developed, product models can not be treated as static and frozen but need to be adjusted frequently, requiring existing instances of the model to be migrated from one version to another.

Hence, the mapping problem domain is identified. In the emerging field of mapping languages, many solutions are proposed by research and industry, all having merits and weaknesses. We present an overview and a brief comparison of several approaches in this field from the system-developer's perspective.

INTRODUCTION

We present an overview of some of the mapping methods currently available in the product data technology arena. Methods based on different programming and specification paradigms will be discussed, including: declarative style (VML); functional style (VDM–SL); knowledge based (XP–rules, KIF); and imperative programming styles (EXPRESS–V, EXPRESS–C, EXPRESS–M, SDAI and C++).

This overview was established from a questionaire which was sent out, along with a mapping example, to a selection of developers and users of these methods. The example was comprised of five mapping exercises between two pairs of EXPRESS schemas, a `building system structural component` (BSSC) schema and a `simple geometry` (SG) schema as source schemas along with a

[1] The full specifications mentioned in this paper are available electronically on the World Wide Web `http://dutcu15.tudelft.nl/~marcel/mapping.html` or by anonymous ftp from `dutcu15.tudelft.nl` (130.161.136.157) in the directory `/pub/mapping`.

[2] `m.verhoef@ct.tudelft.nl`, `http://dutcu15.tudelft.nl/~marcel`, Delft University of Technology, Faculty of Civil Engineering, Building Engineering Group, P.O. Box 5048, 2600 GA Delft, The Netherlands

[3] `liebich@cab-muenchen.de`, Computer Anwendung in der Bauplanung, Osterwaldstraße 10, D–80805 München, Germany

[4] `trebor@cs.auckland.ac.nz`, `http://www.cs.auckland.ac.nz/~ramo01`, Department of Computer Science, University of Auckland, Private Bag 92019, Auckland, New Zealand

`preliminary structural system` (PSS) schema and a `structural system geometry` (SSG) schema as target schemas. Included with the schemas was an ISO–STEP part 21 physical file containing an instance of the EXPRESS source schemas and a STEP file containing the expected output. Using the results from the questionaire and the specifications of the different mapping examples, we have initiated a comparison of the languages (see the electronic documents referenced at the start of this paper) using the following criteria:

- coverage (what kind of problems can the mapping approach handle),
- adaptability (how easy is it to adjust the mapping to accommodate new issues),
- specification clarity (what kind of abstraction mechanisms are available),
- reading clarity (how understandable are mappings from a laymans point of view),
- tool support (support incremental development, graphical support),
- ease of integration (can it be plugged into a data exchange system),
- granularity of mappings (does the whole model get mapped, or can incremental updates be handled).

In the remainder of the paper we will discuss the problem domain in general. Then the mapping case will be presented and we will take one of the problems from the case study to illustrate all the mapping formalisms considered in this paper. Finally, a comparison will be made of the languages and some conclusions will be drawn.

The scope of the mapping problem domain

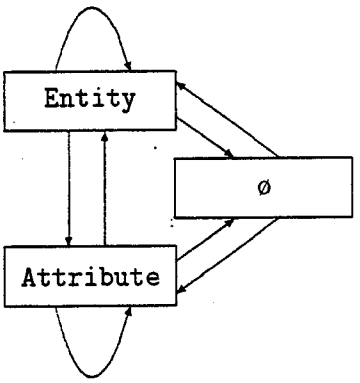
Although mapping problems are not limited to the domain of product data technology (e.g. relational databases) we have focussed our investigation on mapping methods that have emerged in this field, i.e. methods which are able to manipulate data models specified in the EXPRESS modelling language [Expr92], either directly (e.g. EXPRESS–C, SDAI) or in some abstract form (e.g. VDM, KIF). Some of the types of mapping problems that could be considered with these languages are as follows:

1. transformations performed within the context of a single schema (e.g. adding, deleting or updating instances: instance versioning) [5] or
2. transformations performed on two schemas which differ very little semantically (e.g., version migration) or

[5] We do not consider this to be a mapping problem at all. This example was merely included to sketch the problem domain as a whole and to position the mapping problem domain within it.

3. transformations performed on two schemas with a large semantic mismatch.

We consider the last category as the most important mapping problem. We believe that the complexity of these mappings are mainly influenced by the extent of the semantic mismatch between the source and target schemas. In figure 1 (reproduced from [Bijnen94]), an overview is given of the types of mappings that might occur within a typical mapping problem.



Entity → Entity
Entity → Attribute
Attribute → Entity
Attribute → Attribute
ø→ Entity (creation)
ø→ Attribute (initialization)
Entity → ø (deletion)
Attribute → ø (remove value)

Figure 1: Mapping problem domain break-down

The definition of a mapping between the types described in figure 1 has to deal with many issues that are independant of the language used to represent them. Fundamentally, a mapping language needs to define the entities which take part in a mapping. Where a different mapping needs to be performed between the same entities, depending upon certain conditions, it is necessary to be able to discriminate between these conditions. Where objects are created in a mapping it may be necessary to provide initial (or default) values for attributes of the object. These issues are handled in different ways by all mapping languages that we survey.

As well as considering the types of mappings in figure 1 it is important to consider the cardinality of the mappings that can occur. The 1:0, 0:1 and 1:1 cardinalities can be derived from the arrows in the figure, but it also covers the 1:$\mathcal{C}$, $\mathcal{C}$:1, 1:$\mathcal{N}$, $\mathcal{N}$:1, $\mathcal{C}$:$\mathcal{N}$, $\mathcal{N}$:$\mathcal{C}$ and $\mathcal{N}$:$\mathcal{M}$ cardinality cases [6]. The last five cases imply a conditional component to the mapping.

When performing the mapping of data between objects there are many types of mappings that need to be performed. The simplest can be expressed through equations relating various attributes. Some mappings will require the collapse, or construction, of links and structures, requiring methods for referencing values through chains of pointers, or for creating objects to provide a chain of pointers. Complicated mappings will require large procedures to be

[6] $\mathcal{C}$ = constant, $\mathcal{N}$,$\mathcal{M}$ = variable. Note that not all combinations of problem types and cardinalities are valid combinations.

solved and many mappings will need to iterate over the values found in lists, sets or bags.

At a meta-mapping level, mappings will require unit conversion, or type conversion. Where attributes reference other objects (through pointer structures) it will be necessary to map between the pointer types. Lists of object references may have to be split or combined with other lists of object references based on certain conditions. The complete list of the types of mapping that can occur are detailed in the electronic files referenced at the start of this paper.

```
TYPE connection_type =
  ENUMERATION OF
    (support_connection,
     element_connection);
END_TYPE;

ENTITY building_component
ABSTRACT SUPERTYPE OF (ONEOF
  (structural_component,
   component_relationship));
  ident : REAL;
END_ENTITY;


ENTITY  structural_component
ABSTRACT SUPERTYPE OF (ONEOF
  (column, beam))
SUBTYPE OF (building_component);
  specified_by : SET [1:?] OF
    product_characteristic;
  represented_by :
    geometric_representation_item;
END_ENTITY;


ENTITY component_relationship
SUBTYPE OF (building_component);
  related : structural_component;
  relating : structural_component;
  quality : connection_type;
END_ENTITY;
```

```
TYPE support_connection = ENUMERATION OF
  (free_support, restrainted_support, un_known);
END_TYPE;

TYPE element_connection = ENUMERATION OF
  (joint_connection, rigid_connection, un_known);
END_TYPE;

ENTITY structural_component
ABSTRACT SUPERTYPE OF (ONEOF
  (structural_assembly, structural_element,
   structural_connector));
  identified_by : INTEGER;
END_ENTITY;

ENTITY structural_connector
ABSTRACT SUPERTYPE OF (ONEOF
  (support_connector,element_connector))
SUBTYPE OF (structural_component);
  related, relating : structural_element;
END_ENTITY;

ENTITY support_connector
SUBTYPE OF (structural_connector);
  type_of : support_connection;
END_ENTITY;

ENTITY element_connector
SUBTYPE OF ( structural_connector );
  type_of : element_connection;
END_ENTITY;
```

Figure 2: The source and target schema excerpts for example 4

THE CASE STUDY

In the example used to survey the presented mapping languages we have attempted to include the majority of the types of mappings and cardinalities listed in figure 1. A relatively small mapping example was sent out with the following five problems to be solved by each method:

1. `bssc.material` → `pss.material`, depending on the cardinality of the attribute `bssc.material.classified_by` (attribute → entity mapping with a 1:$\mathcal{N}$ cardinality).

2. `sg.simplified_block` → `ssg.linear_geometry`, a complex geometric transformation is required, involving many different mapping problem issues.
3. `bssc.column` → `pss.column` and in analogy `bssc.beam` → `pss.beam` (mainly 1:1 attribute → attribute and conditional mapping, depending on the entity type name).
4. `bssc.component_relationship` → `pss.support_connector` **or** `bssc.component_relationship` → `pss.element_connector` depending on the value of the `bssc.component_relationship.quality` attribute. Reuse of the results from example 3.
5. Conditional creation of `pss.simplified_frame` entities, depending on the relationships between `bssc.component_relationship`, `bssc.column` and `bssc.beam` entity instances. This is *not* a typical type of mapping problem, but a complex precondition for a mapping.

Due to size limitations, we will only show excerpts from the mapping code of all the methods based on example 4, unless otherwise indicated. For the full mapping specifications we refer the reader to the electronically accessible mapping code and questionaires. The relevant parts from the `bssc` and `pss` schemas for example 4 are shown in figure 2.

SURVEY OF MAPPING LANGUAGES

In this section we will briefly describe all the mapping methods surveyed, along with a mapping specification example. Initially we will present the four imperative languages which are oriented towards EXPRESS, followed by two EXPRESS related methods that use a non-imperative specification style and the section will close with two general specification paradigms that are not EXPRESS specific.

The EXPRESS–M language

EXPRESS–M was developed, initially as part of a PhD project, to solve the problem of AP inter-operability in the STEP standard [Expr–M]. As the language was required for use in STEP it was developed to look very similar to EXPRESS, to use all the standard types and functions of EXPRESS and to be useable with SDAI. EXPRESS–M is supported by a compiler which generates C code that will work with any late binding SDAI that is properly ISO–10303–22 conformant.

EXPRESS–M mappings are uni-directional and map a whole model at a time (no partial updates of models). There is a single MAP defined for each unique combination of classes in the two schemas and discrimination between mappings based on values of an object must be made in the MAP specification (as is shown in the example below). A full range of expressions

can be represented using all of EXPRESS's functions and external functions. EXPRESS–M uses an imperative programming style, utilising the iterative constructs of EXPRESS and allowing local and global variables to be specified for mappings.

EXPRESS–M allows for the specification of mappings between EXPRESS defined types and utilises explicit casting to transform values between simple types as well as complex types (as shown in the diagram below). EXPRESS–M has evolved considerably in the last few years and has taken on-board many of the good ideas from other mapping languages.

```
MAP ONEOF(support_connector, element_connector) <- component_relationship;
 IF quality = support_connection THEN
  MAP support_connector <- component_relationship;
   identified_by := {INTEGER}id;
   related := {structural_element}related;
   relating := {structural_element}relating;
   type_of := un_known;
  END_MAP;
 ELSE
  MAP element_connector <- component_relationship;
   identified_by := {INTEGER}id;
   related := {structural_element}related;
   relating := {structural_element}relating;
   type_of := un_known;
  END_MAP;
 END_IF;
END_MAP;
```

The EXPRESS–V language

At the Design and Manufacturing Institute of the Rensselaer Polytechnic Institute work has been undertaken to define the EXPRESS–V language as an addition to EXPRESS to accommodate views (comparable to relational databases) [Hard94]. The implementation of EXPRESS–V is based on ST-Developer, a product from StepTools Inc. EXPRESS–V allows the extraction of views from integrated databases. The language offers methods for

1. selection of those entities from the integrated database that are actually needed by an application and
2. simplification of the entities in an integrated database (e.g. from Application Interpreted Model (AIM) to Application Requirements Model (ARM) [Hard&94]).

EXPRESS–V explicitly distinguishes between the support of read-only views and read-write views. It introduces the `view` declaration to EXPRESS which defines the selection and transformation process. A view declaration can include `create` and `delete` clauses and a special `update` clause for read/write views.

In the example case, the source schema is first mapped onto an intermediate schema (e.g. the selection takes place in this process) and the result is mapped onto the target schema [7].

```
VIEW support_connector1
FROM (component_relationship)
WHEN (component_relationship.quality = 'support_connection');
VIEW_ASSIGN
  identified_by := Real_to_Integer(component_relationship\
    building_component.id);
  type_of := 'un_known';
  FROM (structural_element1)
  WHEN (structural_element1.off = component_relationship.related);
  BEGIN related  := component_relationship.related; END;
  FROM (structural_element1)
  WHEN (structural_element1.off = component_relationship.relating);
  BEGIN relating  := structural_element1; END;
END_VIEW;

VIEW support_connector
FROM (support_connector1)
WHEN TRUE;
VIEW_ASSIGN
  identified_by := support_connector1\building_component.identified_by;
  type_of := support_connector1.type_of;
  related := support_connector1.related;
  relating := support_connector1.relating;
END_VIEW;
```

The EXPRESS–C language

Within the ESPRIT-III project PISA, the information modelling language EXPRESS–C has been defined [Expr–C] (where "C" stands for conceptual). EXPRESS–C extends and enhances the capabilities of EXPRESS by modelling both static and dynamic (behavioural) properties. It can be considered as a first step towards a fully object-oriented version of EXPRESS, as suggested within the EXPRESS V2.0 development targets.

At the University of Karlsruhe, a generator called ECCO has been developed that produces code out of an EXPRESS and EXPRESS–C specification and incorporates a level 2 compliant implementation of ISO-STEP part21 physical file read/write functionality. ECCO also provides a graphical support environment. EXPRESS–C was not primarily intended as a mapping notation, its usage as a mapping language has been a more recent initiative.

In EXPRESS–C, the mapping is described in a `transaction` clause, which is called in the context of an `event`. EXPRESS–C is an imperative modelling language. It allows bi-directional views by specifying transaction clauses for both directions.

[7] Only the case of the `support_connector` is shown here, similar code exists for the `element_connector`.

```
TRANSACTION t_map_component_relationship;
LOCAL
  socr : SET OF component_relationship;
  sosc : SET OF structural_connector := [];
END_LOCAL;
  socr := POPULATION('BSSC.COMPONENT_RELATIONSHIP');
  REPEAT i := 1 TO HIINDEX(socr);
    sosc := sosc + map_component_relationship(socr[i]);
  END_REPEAT;
END_TRANSACTION;

FUNCTION map_component_relationship
  (cr : component_relationship) : structural_connector;
LOCAL
  sc : structural_connector;
END_LOCAL;
  IF (cr.quality = support_connection) THEN
    sc := compare (support_connector(support_connection.un_known) ||
      structural_connector (map_structural_component(cr.related),
      map_structural_component(cr.relating)) ||
      structural_component(cr.id));
  ELSE
    sc := compare (element_connector(element_connection.un_known) ||
      structural_connector (map_structural_component(cr.related),
      map_structural_component(cr.relating)) ||
      structural_component(cr.id));
  END_IF;
  make_instances_persistent([sc]);
  RETURN(sc);
END_FUNCTION;
```

The SDAI API and C++

One could claim that the current solutions provided by the ISO-STEP standard itself are mature enough to solve these issues and no new formalism or specification language is necessary. Wasn't part 22, the Standard Data Access Interface [SDAI], intended to adress these kinds of problems? SDAI does not offer the conceptual clearness as the more specialized methods do, but still it promotes an application programming interface that facilitates model transformations.

Within the COMBINE project, a parser kit and a data exchange system [Lock&94] where developed, both implementing large parts of the SDAI binding for C++. Both tools allow the user to generate a set of C++ classes directly from an EXPRESS schema thus giving the programmer access to the richness of the product model in C++ [8]. The parser kit is used for building interfaces to off-line tools that manipulate a central database in an integrated design system and the data exchange system is used to facilitate the on-line tools. Both systems have a generic, schema independant part, which performs

[8] An early-bound C++ model is meant here, although late-bound access to the data model is possible in both cases at all times.

the transformation from part 21 physical file to the C++ model and vice versa. The programmer can add behaviour functionality to the C++ classes and use standard object-oriented programming techniques to structure, develop, maintain and reuse this functionality.

In this example, we will show an excerpt from the mapping code written in C++ for the data exchange system (DES). The generated C++ class structure was used as-is, no functionality was added to the classes. All the added mapping functionality is shown here, to give a good impression of the amount of code necessary to implement even a trivial example [9]. The DES was implemented using a commercial available object-oriented database ObjectStore and much of the functionality shown in the excerpt relies on the C++ api that ObjectStore supplies with its product.

```
void Example1(/* os_database *kdb, SdaiModel *mod */)
{
  SdaiString entname("bssc_material");
  SdaiEntityExtents *ext = mod->GetEntityExtents(entname);
  if (ext->Instances()->cardinality() != 0) {
    os_cursor c1(*ext->Instances());
    bssc_material_ptr bmp = (bssc_material_ptr) c1.first();
    while (bmp != NULL) {
      if (bmp->ClassifiedBy() && (bmp->ClassifiedBy()->cardinality() > 0)) {
        os_cursor c2(*bmp->ClassifiedBy());
        CEString *name = (CEString *) c2.first();
        while (name != NULL) {
          pss_material_ptr pmp = &pss_material::create(kdb,mod);
          pmp->Name(new (kdb, CEString::get_os_typespec()) CEString(*name));
          name = (CEString *) c2.next();
        }
      }
      bmp = (bssc_material_ptr) c1.next();
    }
  }
}
```

XP–rules

XPDI is a toolset, developed by CSTB, for the specification and prototype implementation of STEP product models. XPDI offers the modeller a graphical and textual user-interface to develop EXPRESS schemas [XP-EXPG]. Futhermore, a Lisp late-binding implementation of the SDAI interface definition has been implemented to enable dynamic interaction with the XP rule base language [XP-SDAI].

The experiences drawn from solving rule-based problems by means of this system lead to the development of the XP-rule language. XP-rule allows the definition of bidirectional views and provides a high level of adaptability due to the declarative aspects of the mapping method. Integration of Lisp or C

[9] Note that example 1 is used in this case in stead of example 4.

code is possible. The system can generate an executable from an XP-rule definition that can interface to the SDAI repository. The notation of XP-rule language has been kept as close to EXPRESS-M as possible. XP-rule has been succesfully used by CAB to perform a mapping in the COMBI project.

```
RULE BSSC_component_relationship->PSS_structural_connector :
  LET BSSC_component_relationship A component_relationship IN
    MODEL BSSC THEN EXECUTE
     IF ( quality OF BSSC_component_relationship
          IS EQUAL TO "support_connection"
     THEN CREATE PSS_connector A support_connector IN MODEL PSS
     ELSE EXECUTE
      IF ( quality OF BSSC_component_relationship
           IS EQUAL TO "element_connection"
      THEN CREATE PSS_connector A element_connector IN MODEL PSS))
    AFFECT RESULT OF F_real2int (id OF BSSC_component_relationship)
      TO identified_by OF OBJECT PSS_connector
    EXECUTE conversion_GET ( OBJECT BSSC_related , related OF
      BSSC_component_relationship )
    AFFECT OBJECT BSSC_related TO related OF OBJECT PSS_connector
    EXECUTE conversion_GET ( OBJECT BSSC_relating , relating OF
      BSSC_component_relationship )
    AFFECT OBJECT BSSC_relating TO relating OF OBJECT PSS_connector
    AFFECT "un_known" TO type_of OF OBJECT PSS_connector
```

The View Mapping Language (VML)

VML is a bi-directional, high level and declarative language for the specification of mappings between two arbitrary schemas (or versions of schemas). A declarative style was chosen to enable the definition of a mapping at a level closely aligned to the level a developer would conceptualise the correspondences between classes. VML tries to distance the mapper from details of implementation to concentrate on straight specification. To this extent VML uses a very simple notation with a large amount of semantics implicit in the operators used to describe a mapping [Amor94].

The mappings shown below highlight the qualities of VML. An `inter_class` specifies that a mapping can occur between a class (or classes) from each of the two schemas taking part in the mapping. The `invariants` section specifies conditions which must be satisfied for the mapping to take place. An `initialisers` section specifies initial values for attributes of the entities which get created in the mapping. The `equivalences` section specifies the relationship between attributes in the classes being mapped between.

Equivalences can be specified through equations in a declarative manner, or through functions, or if no specification is possible with the previous two methods then through a procedural mapping. The mapping system associated with VML performs automatic type conversion for basic types (e.g., real to integer) and performs pointer conversion for attributes whose types are entity references (based on the `inter_class` mapping for the class of the referenced entity, e.g., *related* = *related* in the mapping below).

```
inter_class([component_relationship], [support_connector],
  invariants( quality = 'support_connection' ),
  equivalences( id = identified_by, related = related,
    relating = relating ),
  initialisers( type_of = 'un_known')
).

inter_class([component_relationship], [element_connector],
  invariants( quality = 'element_connection' ),
  equivalences( id = identified_by, related = related,
    relating = relating ),
  initialisers( type_of = 'un_known')
).
```

The mapping system associated with VML takes a more general approach to data mapping than the other languages specified here in that it is configured to perform in an interactive environment as well as handling full model conversions. The mapping system allows the simultaneous connection of multiple design tools or databases and accepts incremental modifications to a model which can be propagated through to all connected tools. This system can therefore verify the global consistency of models in an integrated system by tracking which modifications have been propagated to which design tools and which changes are outstanding for a particular design tool.

The VDM specification language (VDM-SL)

One could question the need for dedicated mapping techniques and claim that generally available specification languages are sufficient to adress the mapping problem. In this section we will consider the suitability of the formal specification language VDM-SL for application in this problem domain. VDM-SL is probably one of the best-known and mature formal specification languages available at this time [VDM].

VDM stands for "The Vienna Development Method": a collection of techniques for the formal specification and development of computing systems. It consists of a specification language called VDM-SL; rules for data and operation refinement which allow one to establish links between abstract requirements specifications and detailed design specifications down to the level of code; and a proof theory in which rigorous arguments can be conducted about the properties of specified systems and the correctness of design decisions.

When using VDM-SL, the specifier is faced with two problems, a) VDM-SL lacks an inheritance type model, therefore EXPRESS entity definitions cannot be translated directly onto a VDM-SL equivalent, b) the resulting specification still has to be integrated into a data exchange system with a well-defined api (such as the ISO-STEP part 22 SDAI definition offers) before an executable system can be delivered. However, not withstanding these limitations, the mapping examples could be specified with relative ease in VDM-SL as the following excerpt shows:

```
1.0  MapCompRela2StructCon : bssc-component-relationship →
.1                           pss-structural-connector-type
.2  MapCompRela2StructCon (cr) ≜
.3    let sc = mk-pss-structural-connector
.4               (MapStructCompTp2StructElemTp (cr.related),
.5                MapStructCompTp2StructElemTp (cr.relating)) in
.6    if cr.quality = SUPPORT_CONNECTION
.7    then mk-pss-support-connector (sc, UN_KNOWN)
.8    else mk-pss-element-connector (sc, UN_KNOWN)
```

VDM-SL has a very rich type system which compensates for the lack of type inheritance[10]. Futhermore, the abstraction mechanisms offered by VDM-SL are very powerful and therefore the mapping can remain concise. For very complex mappings, one could start by making a high-level implicit mapping definition and incrementally lower the level of abstraction by e.g. using refinement calculus and hence create a fully explicit (and implementable) mapping algorithm. Many excellent tools exist to assist the specifier during the development process, such as the IFAD VDM-SL toolbox which was used to create this mapping specification. This toolkit offers the possibility to generate C++ code from the resulting specification which makes integration into data exchange systems possible. At the Delft University of Technology this method has been succesfully applied to a similar mapping experiment.

The Knowledge Interchange Format (KIF)

The agent communication language (ACL) is centred around the knowledge interchange format (KIF) which was developed by the ARPA knowledge sharing initiative and is currently being enhanced at Stanford University [Khedro&94]. ACL/KIF was originally developed to exchange distinct information between applications, following the messaging paradigm, but due to the diversity of the tools to be integrated, translation was also a major topic of research. ACL/KIF identifies two kinds of translation problems:

1. vocabulary translation as it arises from the differences of the abstractions inherent in the implementation of different agents (mapping in the sense of this paper) and
2. the logical translation dealing with the consequences of the problems arising from the limits imposed by agents on the logical structure of messages.

The use of this language and the supporting agent-based architecture attempts to take software integration beyond the exchange of data files or the sole use of database management systems. As the language has to handle

[10]Within the Aprodite ESPRIT project, an object-oriented dialect of VDM-SL, VDM++, has been defined together with tool support, solving this problem

bidirectional mappings between the messages sent by the different agents in their native conceptualization, it can partly be considered as a formal mapping notation [KIF].

The specifications are based on first-order logic with various extensions to enhance its expressiveness. KIF follows the declarative and knowledge based paradigms. The implementation of KIF axioms for mapping are based on Lisp. KIF was not developed within the product data technology arena and does neither support EXPRESS on schema level nor STEP physical file format on instance level. However, the advantage of KIF is the data exchange and translation on demand, since it is message based. Therefore very distinct pieces of data can be exchanged between applications, rather than bulk data.

```
(<= (pss!support_connector ?ent) (bssc!component_relationship ?ent)
  (= (bssc!component_relationship.quality ?ent) support_connection))

(<= (pss!element_connector ?ent) (bssc!component_relationship ?ent)
  (= (bssc!component_relationship.quality ?ent) element_connection))

(<= (= (pss!support_connector.identified_by ?ent) ?id)
  (= (bssc!component_relationship.id ?ent) ?id)
  (= (bssc!component_relationship.quality ?ent) support_connection))

(<= (= (pss!element_connector.identified_by ?ent) ?id)
  (= (bssc!component_relationship.id ?ent) ?id)
  (= (bssc!component_relationship.quality ?ent) element_connection))

(<= (= (pss!support_connector.type_of ?ent) ?type)
  (= (bssc!component_relationship.quality ?ent) ?type)
  (= ?type support_connection))

(<= (= (pss!element_connector.type_of ?ent) ?type)
   (= (bssc!component_relationship.quality ?ent) ?type)
   (= ?type element_connection))
```

COMPARISON AND CONCLUSION

In this paper we have presented a wide range of mapping languages which provide the ability to specify the mapping between two (sets of) schemas. In many ways this is the only point of similarity between the languages as they all target different areas of mapping with some very different language paradigms. The comparison of mapping languages based on the questionaires and example code is available electronically from the location specified at the beginning of this paper.

While the comparison provides a more detailed analysis of the mapping languages we hope that this paper gives the reader an idea of the types of languages available, their style of specification and the area that they were developed to perform in. Perhaps the only conclusion that can be drawn from this paper is that no one mapping language can be used in every situation and

depending upon the type of mapping environment an integration group finds themselfs in this will determine which of these languages would provide the best solution to their needs. A description of some mapping situations and suitable mapping languages is described below.

If a requirement is for an interactive, integrated environment then a language which provides for incremental updates and which maintains the consistency between objects in the models being mapped between (i.e., VML, KIF, EXPRESS–V, XP–rules) will be necessary. If your requirement is for the mapping of complete models every time then the other languages may prove more efficient.

If a high level modelling of mappings is required for a rapidly changing set of schemas then a higher level mapping specification (e.g., VDM–SL, VML) will prove the most efficient to describe mappings, and to modify them as schemas are modified. Higher-level languages will prove to be easier for a lay-person to understand if it is necessary to describe mappings to a group of non-experts. A more iterative language will prove to be easier for a programmer to translate into executable code, especially if the language or environment the integrated system is being developed in does not support SDAI or other STEP parts.

If compatibility with STEP and EXPRESS are required, then the languages which are being developed as part of STEP (e.g., EXPRESS–M, EXPRESS–V, EXPRESS–C) will have great appeal to the developers. If knowledge based systems are the main tools to be integrated into the system then one of the knowledge-based languages (i.e., XP–rules, KIF) will probably provide the best service.

The list of breakdowns above is by no means complete (you may require bi-directional mappings, schema version migration, schema modification due to mappings, etc.) the choice of mapping language is obviously very dependant upon the project being tackled. Conversely, it is clear that no one mapping language is capable of serving all the needs of every integrated design system developer. To this extent we believe that many of the languages surveyed here will evolve and grow over the next decade as STEP grows, with each of the languages filling a particular niche in the broad field of integrated systems.

ACKNOWLEDGEMENTS

This paper would not have been possible without the collaboration of many contacts in the mapping arena. We would specially like to thank Martin Hardwick (EXPRESS–V), Günther Staub (EXPRESS–C), Alain Zarli (XP-rules), Taha Khedro (KIF) and Ian Bailey (EXPRESS–M) for taking the time to complete the questionaire and implement the presented case study in the mapping formalism of their expertise.

Furthermore, the authors wish to thank their collegues within their respective institutions for valuable comments on early drafts of this paper and last but not least for giving us the opportunity to write this paper.

REFERENCES

[Amor94] Robert Amor, *A Mapping Languages for Views*, Departmental report, Computer Science Department, University of Auckland, New Zealand, 1994

[Bijnen94] Anthony Bijnen, *Operation Mapping* or *How to get the right data?* First ECPPM conference, A.A. Balkema publishers, October 1994

[Expr92] ISO TC184/SC4, Industrial automation systems – Product data representation and exchange – Description methods: *The Express language reference manual*, ISO, 1992

[Expr–C] G. Staub, A. Nieva, F. Schönefeld, PISA Information Modelling Language: *Express–C*, ISO TC184/SC4/WG5 working draft, 1994

[Expr–M] ISO TC184/SC4/WG5 N243, *EXPRESS–M Reference Manual*, CIMIO Ltd, July 1995

[Hard94] M. Hardwick, *Towards Integrated Product Databases Using Views*, Design & Manufacturing Institute technical report 94003, Rensselaer Polytechnic Institute, 1994

[Hard&94] M. Hardwick, D.L. Spooner, M. Kilty and Z. Jiang, *Mapping EXPRESS AIM's To ARM's Using Database Views: A Comparison of Three Approaches*, Design & Manufacturing Institute technical report 94041, Rensselaer Polytechnic Institute, 1994

[Khedro&94] T. Khedro, M.R. Genesereth, P.M. Teicholz, *Concurrent Engineering Through Interoperable Software Agents*, In: First conference on Concurrent Engineering: Research and Applications, Pittsburgh, 1994

[KIF] M.G. Genesereth, R.E. Fikes, *Knowledge Interchange Format Version 3*, Reference Manual, Computer Science Department, Stanford University, 1992

[Lock&94] S.R. Lockley, W. Rombouts, W. Plokker, *The COMBINE Data Exchange System*, First ECPPM conference, A.A. Balkema publishers, October 1994

[SDAI] ISO/TC184/SC4/WG7 N–350, Industrial automation systems – Product data representation and exchange – Part 22: *Standard Data Access Interface*, ISO Committee Draft, August 31, 1993

[VDM] ISO/IEC/JTC1/SC22/WG19 N–20, *Information Technology Programming Languages – VDM-SL*, First committee draft standard: CD 13817-1, November 1993

[XP-EXPG] XIG, *XP-Express-G Reference Manual Version 2.0*, CSTB, 1994

[XP-SDAI] XIG, *XP-SDAI Lisp Binding Ilog Talk Implementation Reference Manual Version 1.0*, CSTB. 1994