ESPRIT 7280 - ATLAS

Data Abstraction Generalization and View Conversion Mechanisms

Authors

Patrice Poyet CSTB
Eric Brisson CSTB
Eric Grivart CSTB
Jan Jaap van Horssen TNO
Bart Behage TNO
Marco Mooij TNO
Roberto Greening IEZ
Rakesh Singh SNI

<table>
<thead>
<tr>
<th>Distribution</th>
<th>D302a</th>
<th>D302c</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>IEZ</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>TNO</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>CSTB</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>SNI</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>TW</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>CEC</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>15/02/94</td>
<td>15/09/94</td>
</tr>
<tr>
<td>Deliverable Reference</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WP3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D302c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Draft/Issued/Revised</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Draft</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Draft/Rev.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 August 1994</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Release to</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Document Control Sheet

<table>
<thead>
<tr>
<th>Issue N°</th>
<th>Page N°</th>
<th>Amendment Record</th>
<th>Date</th>
<th>By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draft 1</td>
<td>Full Doc.</td>
<td>First release of the Data Abstraction, Generalization and View Conversion (all partners) - PTC Meeting in SNI offices</td>
<td>18/10/93</td>
<td>Poyet</td>
</tr>
<tr>
<td>Draft 2</td>
<td>Ful Doc.</td>
<td>Second release of the Data Abstraction, Generalization and View Conversion (all partners) - WP3 Workshop in CSTB offices</td>
<td>11/01/94</td>
<td>Poyet</td>
</tr>
<tr>
<td>D302a</td>
<td>Full Doc.</td>
<td>Official release of D302a (all partners and CEC)</td>
<td>15/02/94</td>
<td>Poyet</td>
</tr>
<tr>
<td>Draft3</td>
<td>Full Doc.</td>
<td>Third draft of the D302</td>
<td>15/09/94</td>
<td>Brisson</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

1. Introduction............................................................................................................................................. 1
   1.1. Objective of this deliverable.................................................................................................................... 1
   1.2. Previous experiences............................................................................................................................... 2
   1.3. ATLAS modelling resources.................................................................................................................... 6
   1.4. ATLAS Models......................................................................................................................................... 7
   1.5. The View Manager (XVoM) and the View Converter (XVcM).......................................................... 8
   1.6 Analysis of Methods and tools for Conversions..................................................................................... 9

2. The XVoM Manager.................................................................................................................................... 9
   2.1. Introduction............................................................................................................................................. 9
   2.2. The methodology................................................................................................................................... 11
   2.3. The XP-SDAI implementation including View Management............................................................. 21

3. The XVcM Converter.................................................................................................................................. 22
   3.1. Introduction............................................................................................................................................. 22
   3.2. A Rule based approach........................................................................................................................... 24
   3.3. Hints to future conversion strategies..................................................................................................... 31

4. Frameworks, Models and APs Interoperability.......................................................................................... 33
   4.1. Introduction............................................................................................................................................. 33
   4.2. Task 3200 as a support to the APs interoperability problem ?............................................................. 35
   4.3. The AEC Framework example............................................................................................................... 38

5. Analysis of differents methods and tools for conversions........................................................................... 40
   5.1 Introduction............................................................................................................................................. 40
      5.1.1 Context within the ATLAS project..................................................................................................... 40
      5.1.2 The conversion problem.................................................................................................................. 41
      5.1.3 Approach......................................................................................................................................... 41
      5.1.4 Structure of the chapter.................................................................................................................. 42
   5.2 Conversion examples............................................................................................................................... 44
      5.2.1 Introduction..................................................................................................................................... 44
      5.2.2 A conversion benchmark................................................................................................................. 44
         5.2.2.1 A model for technical drawings............................................................................................. 45
5.7.2.2 Possibilities and restrictions ...................................................... 92
5.7.3 PEER .................................................................................................. 92
5.7.2.1 Introduction ........................................................................... 92
5.7.3.2 Possibilities and restrictions ...................................................... 93
5.7.4 Operation Mapping ........................................................................ 93
5.7.4.1 Introduction ........................................................................... 93
5.7.4.2 Possibilities and restrictions ...................................................... 94
5.7.5 A Mapping language for views ...................................................... 95
5.7.5.1 Introduction ........................................................................... 95
5.7.5.2 Possibilities and restrictions ...................................................... 97
5.8 Classification of conversions ............................................................... 97
5.8.1 Introduction ........................................................................... 97
5.8.2 A framework for classification ...................................................... 98
5.8.3 Examples .................................................................................. 100
5.8.4 Conclusions ........................................................................... 103
5.9 Conclusions and recommendations ..................................................... 103
5.9.1 Introduction ........................................................................... 103
5.9.2 Evaluation of the approach ........................................................... 104
9.3 Evaluation of the project ................................................................. 105
5.9.4 Evaluation of the alternatives for conversion ................................ 106
5.9.5 Recommendations for conversion within ATLAS ....................... 106
6. An IT Vendor Viewpoint ....................................................................... 108
6.1. Objectives ................................................................................ 108
6.2. Scope of Work ........................................................................ 108
6.3. Initial SNI results ........................................................................ 109
6.3.1. Use of Object Oriented Paradigm ........................................... 110
6.3.2. Adopting a strategy for development of Information Models ....... 110
6.3.3. Other ways to support View Conversion ................................... 112
7. Conclusions .................................................................................. 114
8 References .................................................................................... 116
9 List of the Figures ........................................................................... 121
1. **INTRODUCTION**

1.1. **Objective of this deliverable**

This document (D302a) is the first step towards the development of data abstraction / generalization and view conversion mechanisms atop the ATLAS product models. The Technical Annex identifies the need for a mechanism supporting data-abstraction and data-generalization for the life-cycle integration of product/process information. It is also clearly stated that an essential part of the problem to be addressed by the project is the conversion of information created by different disciplines (or partner-roles) having different views on the product/process. Even when the same LSE application is used, different partners will create and/or use different sets of information. For example, a beam or column will be described differently by an architect, a structural engineer, a contractor or a supplier. Therefore a mechanism supporting the conversion of views is needed.

It is also acknowledged in the Technical Annex, that related to this topic is the need to describe products/processes on different levels of 'granularity'. A column or beam will be described differently in the context of the entire project (building, plant, etc.) to when a particular detail is being considered (for example a joint). This is true for virtually all aspects of a product; good examples are shape and costs. When a project is in a detailed specification-, construction- or operation-stage, it is required that information on different levels of granularity be made available and that this information be consistent (for example cost of a single component but also of the entire project).

The definition of different levels of abstractions - and therefore of various models located along the generalization / specialization axis - should be made possible in a straightforward manner by means of software tools handling directly the conceptual definition of the product models (LSE Project Model, Building Project Model and View type Models).

Moreover, the aggregation of a number of these models into a single Product Reference Model, for a specific usage, should be supported. A company may wish to select resources at different levels of abstraction / specialization (these can be STEP resources such as Integrated Generic Resources, Integrated Application Resources, Application Integrated Constructs, APs or their own - custom - resources) and to organise for a given purpose (e.g. for the duration of a particular project) these resources into a single Global Product Model (GPM).
Once this aggregation has been nicely done by the software system(s) by means of appropriate overloading mechanisms the need for view mechanisms still remains. The first requirement which can be stated is to manage views on the global project model(s) - generated as instances of the GPM - according to the individual resources used to build up the overall model. Another is to convert information from one of these views to the others. Therefore, the view conversion facilities required are twofold: on one hand extracting data from project models according to specific views which can be traced back at the conceptual level definitions and on the other hand converting information from one view to another ones.

As stated in the Technical Annex, the mechanisms for data abstraction, generalization, view conversion and conversion between levels of granularity are from an IT viewpoint. Thus, a service module will be developed with the intelligence to abstract information by view and at an appropriate level of detail (granularity). Therefore, this module is a n implementation of the SDAI (C Binding, Lelisp Binding) able to take in charge the notion of views not only at the conceptual level (NIAM and Express-G) but also at the implementation level.

1.2. Previous experiences

The experience of the Computer Integrated Construction Division at CSTB with respect to the integration of various models (eventually representing different levels of abstraction) comes from previous projects, one of them of particular interest being the EC COMBINE I project (1990-1992).

Moreover, this R&D effort, conducted under the auspices of DGXII, brought one of the first convincing large scale demonstration of the usefulness of the STEP techniques for the exchange of Product Data Models at an early stage of the development process of the standard. It also explored the possibility to exchange information in a neutral way according to different viewpoints, i.e. the Design Tool Prototypes (DTPs), and in some sense the data abstraction / generalization and view conversion system addressed in that deliverable finds its roots in the functionalities explored during the COMBINE I project.

Based on a conceptual model validated by means of the COMSET system, an Express version of the Integrated Data Model (IDM) could be produced and delivered to the project partners. Each one of the COMBINE partners could read in the conceptual schema in Express and map it to its local implementation in order to get an internal representation of that logical schema.

Then for a particular building project selected for the COMBINE demonstration held in Stuttgart in Nov. 1992 (i.e. referred to as the schoolbuilding example) ISO neutral files could be produced, reflecting
different stages of the design process and exchanged across the partners. The Views were defined as subsets of the Entities and of their Attributes according to the needs of each of the specific DTPs (different tools using the same model do not use the whole set of the concepts). The complete IDM has been implemented by means of the XPDI Station, and the software stored the different Express aspect models corresponding to each of the DTPs supposed to make use of the IDM. Each one of these DTPs could read a Step Physical file, and then instantiate accordingly local data structures corresponding to the information transferred in the neutral form.

These data could then be processed according to the particular functions to be performed by the application. When additional information is produced by the application (e.g. dimensioning of the heating system, of the building envelope, whatever, etc.), it can also be exported by means of the ISO exchange mechanism and sent back to the IDM manager, first to validate it and second in order to make it available to other tools. The scenario of the demonstration was the following:

Step 1

- Load an ISO NF corresponding to the geometry of the School Building (delivered by TUD) in the XPDI Environment (the XPDI platform provides the VAMSET Tool - 3D visualization - and the GAMSET Tool - 2D CAD system - to visualize the geometry of the entities of the IDM);
- Generate an ISO NF corresponding to the DTP1 - Design of external building elements - (use of the Sub-Schema, i.e. DTP1 View type Model);
Step 2

- DTP1 uses the STEP Interface to load this file into its own tool;
- Process of the model by the tools of DTP1;
- DTP1 delivers back the result to the XPDI platform, generating an ISO NF using the STEP Interface (DTP1 View type Model);

Step 3
Load this file into the XPDI platform in accordance with the Sub-Schema of the DTP1, therefore updating the existing objects and creating the new ones; Generate an ISO NF corresponding to the DTP2 - HVAC simulation of the building and of its equipments - (use of the Sub-Schema, i.e. DTP2 View type Model);

Step 4

- DTP2 uses the STEP Interface to load this file into its own tool;
- Process of the model by the tools of DTP2;
- DTP2 delivers back the result to the XPDI platform, generating an ISO NF using the STEP Interface (DTP2 View type Model);
- Load this file into the XPDI platform in accordance with the Sub-Schema of the DTP2 for updating the existing objects and creating the new ones;

The limitations of these initial results and of the approaches followed should be clearly stated so that the real merits of the work done should not be misunderstood. In fact, the views on the product model were defined in a rather awkward manner and without the support of any software function.

Given the entire product model, the subset of the entities being relevant for a specific DTP and their related attributes for the design system considered were determined by the modeller and then compiled in separate files. These files were then reused by low level specific software modules (with no user interface) to generate for each entity of the product model an appropriate mapping for a given DTP.

At the software level, the implementation associated to each XPDI Class (so-called abstract objects) its corresponding Express format, but also kept track of a filtered Express representations as expected by specific DTPs. Of course, the lack of a clear methodological approach to define views at the conceptual level on the product model with a set of associated tools supporting the methodology is quite obvious.

But, it should be added that the goal of the COMBINE I project was not to manipulate views on product models (this was not envisaged at all initially), but that the very large size of the IDM lead to implement these functions to make the management of the model more practical, especially as far as some specific DTPs could not accommodate themselves of such as large neutral format.

1.3. ATLAS modelling resources
The set of modelling resources defined by the ATLAS project is rather traditional in the STEP community. NIAM (Nijssen's Information Analysis Method) is used for data modelling and IDEF0 for the functional modelling effort and the business analysis (i.e. building construction and process plants design and installation).

The formal information modelling language Express supports the exchange of information flows. Provisions have been added to support a complete mapping of NIAM models into Express constructs and vice-versa.

Task 3200 will make use of these modelling resources, and will introduce a methodology and appropriate software tools to handle the concepts of generalization / specialization, to support the life cycle of models and to manage the different views actors may have on the same model.

The first step on the road of view management is to support the ATLAS modelling methodologies by means of a software platform compliant with the guidelines. This has been made possible throughout the efforts carried out in task 1400, where the XPDI Station has been up-graded so that the ATLAS modelling methodologies be supported. This has been done in a progressive manner, first by consolidating the kernel (internal meta schema, efficiency of the internal object management system, etc.), then by incorporating the guidelines defined in task 1300.

This permitted to exchange the LSE Project Model, the Building Project Model and the Various View type Models (VtMs) with TNO. This is quite important because developing a view manager supposes harmonisation with the concepts defined in the LSE Project Model, Building Project Model and VtMs. The view conversion system necessarily has to "understand" the concepts of actors or of life cycle stage defined into the LSEPM and to manage the views as suggested by the different VtMs which have been defined (e.g. Architectural View, Structural Engineer View, etc.).

1.4. ATLAS Models

The ATLAS project defines a number of conceptual models which can be understood as progressive specializations of a generic kernel. The generic kernel is referred to as the LSE Project type Model and contains the concepts which can be of interest to both Building and Process Plant activities. A first specialization of this core model is the Building Project type Model which contains the entities of interest to all the actors involved in the construction process. Furthermore, the View type Models define the concepts manipulated by specific actors.
One should notice that this suite of models represents a progressive move along the generalization / specialization axis. The controlled and dynamic overloading - in the sense of object oriented technologies - of generic models by more specialised sub-models is the approach followed by the service module developed in task 3200 to handle the generalization / specialization mechanisms.

View type Models are linked to particular life cycle stages of the project and this approach leads to the definition of a bi-dimensional matrix where the actors represent one axis and the life cycle stages the other. For each one of the cells of the matrix there exists a specific VtM at a particular project life cycle stage (this is just one of the possible examples of modelling frameworks). Of course, other axis representing different concepts could be envisaged and would extend the number of cells to be considered (e.g. systems leading to System type Models or StM).

The view conversion system will consider any specific extension to a neutral model as a view, whatever the number of axis being used to position the model along an axis, into the cells of a matrix, somewhere in the modelling space, or hyperspace. This will enable to address different modelling frameworks such as the one envisaged in Figure 19.

To operate on and to manage these views, the view manager will require additional data structures, some being typically internal to the software module whereas others should be harmonized with the concepts and entities defined in the generic LSE Project type Model (e.g. actors, etc.). This model will be referred to as the View Manager Model (VMM) and will also be delivered in Express as all the other ATLAS models.

1.5. The View Manager (XVoM) and the View Converter (XVcM)

This task will address two separate issues : on one hand the management of View type Models (VtM), System type Models (StM) or more generically any View oriented Model (VoM) referred to as the XVoM Manager and on the other the conversion of Project Models (PMs) or View Models (VMs) into other PMs, VMs or even into Application type Models (AtMs) - (e.g. Autocad AEC is an AtM) - by means of the View conversion Mechanisms, a software system referred to as the XVcM.

The management of View oriented Models supposes first to offer means to the modeller to define conceptual VoMs. Then it should be possible to load these conceptual VoMs into a software environment supporting at the operational level the management of the information according to these
VoMs or to any other view defined for example as a composition of these VoMs (§ 2.1). The conceptual methodology and the implementation software is entirely supported by STEP methodology and Implementation. It means there is no need to develop new conceptual method nor new implementation concepts.

The View conversion problem is quite different and should offer means to easily develop converters permitting to map a given source model into a - different - target model (§ 3.1). This conversion is certainly not a one to one mapping and the entities encountered into the source model are not just transformed into their counterparts into the target model. In fact, converting a model into a different model is also a view conversion problem, and as far as the semantics of the models differs most of the time, complex reasoning procedures have to be undertaken. The rule language defined in task 3100 (to exchange LSE knowledge by means of neutral technologies), will be used to demonstrate how different models can be converted into each others.

XVoM and XVcM will be introduced into this deliverable and are released as software prototypes presented at the review Meetings in the context of the Atlas demonstration.

**.m2.1.6 Analysis of Methods and tools for Conversions**

The TNO CAD Centre will focus on view conversion in Chapter 5. The conversion problem is a very comprehensive problem. One of the difficulties is that it is not clear how large the semantical gap between two models can be. Therefore it is not easy to solve this problem in a general way, so it is a priori not clear that the solution will be applicable to all conversion problems.

The problem will be tackled as follows. First, a number of conversion examples (among which two specific ATLAS conversions) will be studied and implemented:

1. A conversion benchmark, originating from a research project carried out by TNO CAD Centre.
2. Conversion from AtM Speedikon Architecture to VtM Architect in Global Design Stage.
3. Conversion from VtM Architect in Global Design Stage to BPrM/LSE PtM kernel.

The implementation will be done in the TNO PMshell environment, using C++ to do the actual programming. The purpose of this exercise is twofold: it can yield information for both the implementation of the first ATLAS demonstration and the analysis of the conversions.
Second, the examples will be used to do some kind of "reverse engineering". The conversions will be analyzed and the types of conversions will be classified. As much as possible the need for bidirectional conversions will be taken into account. Several ways to (formally) specify and implement conversions are investigated and recommendations will be given.

2. The XVoM Manager

2.1. Introduction

As it has been previously mentioned (§ 1.4), the ATLAS project leads to the definition of a variety of models, some being neutral in the sense that they are not specific to a certain viewpoint on the project, others being view-oriented as the VtMs (actor oriented) or StMs (system oriented). Other view-oriented models may appear in the future as normal extensions to the ATLAS resources (considering other modelling axis § 1.4.), or simply because other LSE product life cycle stages can be addressed (e.g. ATLAS focuses on creation, but operation, maintenance, refurbishment or demolition could be envisaged as well).

As far as software mechanisms have to be developed to implement and demonstrate the feasibility of a view-oriented management of the Project Models (PMs), it seems reasonable to address the notion of "View" at a very generic level, whatever the type of view being considered (VtMs, StMs, etc.). This should enable the software (XVoM) to be rather independent of the semantics of the views and to operate on current but also on future view-oriented models which may appear later. Therefore, a generic mechanism for managing Views is introduced and makes it possible to consider, for example:

- that a particular description which can be made of a Product / Project Model according to a given stage of the complete life cycle of the product / project (i.e. LCtMs),
- or that the viewpoint a specific actor may have on a Product Model or on a Project Model (i.e. VtMs),
- or any other View oriented Models (VoMs), such as System type Models (StMs);

are only Views in the same generic sense. So actor-oriented, life-cycle-oriented or system oriented management of the LSE knowledge will be made possible by a generic and unique "View" mechanism.

Therefore a View is an EXPRESS schema acting as a functional filter applied on a root object of a complex network $f$(object). This permits the composition of applications (i.e. Views) by means of
classical mathematical notations such as $f \circ g$(object) defining still more complex Views. Views will be defined as classes of functions: e.g. $\mathcal{F} \{f_1, f_2, \ldots, f_n\}$, $\mathcal{G} \{g_1, g_2, \ldots, g_n\}$, etc., any complex view being defined as an arbitrary composition of functions ($f_x \circ g_y$ etc. (object)) filtering the project model. This can be understood as the generalization of the functions developed during the Combine I project (refer to Figure 1), while creating a complete methodological framework to define, implement and manage VoMs and VMs.

2.2. The methodology

A complete methodology is proposed to support the definition, the implementation and the management of Views oriented models.

The first step is to follow at the conceptual level a well organised methodological approach. By means of the modelling tools of the XPDI environment, the modeller defines separately his/her Views oriented models. In fact, if we take for example the VtMs, for each of the cells corresponding to a distinct cross product of an actor at a given life cycle stage (any other modelling spaces could be considered such as the one presented in Figure 1), the modeller should create a separate occurrence of VtM.

In case of the VtMs, as XP-NIAM is a multi-models environment or as XP-XPRESS-G is a multi-schemas system, each of these separate VtMs can exist concurrently and live in a separate model / schema within the modelling environment. Therefore, these models can be manipulated simultaneously.

These VtMs, or PtM and VtMs can be linked at the conceptual level by means of the external NOLOT concept. External NOLOTs are recognized by the software and displayed as blackboxes. These external NOLOTs represent the places where the different models will establish their future inter-connection, in EXPRESS-G USE and REFERENCE interfaces are used.

Even though the methodology requires that the VtMs be kept separate as distinct files, the modeller can experiment interactively in various models / schemas the correct assembly of the set of conceptual VtMs.

The more generic model is loaded first (e.g. LSE Project type Model) with its external NOLOTs being clearly displayed as blackboxes. For instance, some will point to the entities of the Building type Model that will serve as connection means. Then the Building Project type Model can be loaded in due course into the test model / schema (containing already the conceptual model of the LSE PtM) to check the correct assembly of the two.
The process can extend to the other VtMs (i.e. from the Building PtM to the sub VtMs), therefore loading thousands of entities belonging to various PtMs or VtMs and properly inter-connecting the models (overloading the external NOLOTs by their complete definitions when made available by a sub-model). At the end of the process, test models / schemas used for the assembly purposes are removed and the PtMs and VtMs or VoMs are saved in their corresponding files.

We present hereafter, by means of an extremely simple example (things work in a similar way on complete models large of hundreds of entities) how the modeller can define a generic core (referred to here as the Global Product Model - Figure 2) supposed to be later supplemented with various View type Models (here VtM1 displayed in Figure 3 and VtM2 in Figure 4). We also describe the complete sequence leading to the management of the views in the operational environment (once implemented).
Figure 3. XP-NIAM - Simplistic VtM1.
As displayed in Figure 3, the VtM1 has a rather simplified understanding of what a building is (if it is still possible with such a limited example...) and the VtM2 defines differently the concept of Building_State.

At that stage of the process, various implementations can be created by the modelling tools of the XPDI Station to enable the process to move smoothly from the conceptual ground to the operational stage, where implemented models will be loaded and made available to the users through the API.

One Express schema per View cell (in case of the VtMs located in the simple matrix like modelling framework) is then generated and eventually the appropriate target implementation is also created. It can be C++ classes, SQL create-table orders, O2 create-schema orders. Express entities are important as they will be re-used to demonstrate how the access to the Project Models (PMs) populated by the instances of a particular LSE projects, can be accessed according to the Views defined by the VtMs or by any other VoMs in the XP-SDAI implementation (LeLisp or C Binding).
All the PtMs, VtMs and VoMs are then stored as separate Express files (i.e. entities). According to the user needs a certain number of these files can be loaded into a given SDAI repository for a particular objective, therefore defining the views available in the environment (Figure 5). The Object Oriented Language of the platform (extension of SDAI) and the Repository / Object Management System are then powerful enough to support a sophisticated structural overloading of the entities. The same entity can be defined in different PtMs or VtMs (i.e. derived from the connection points - blackboxes, or as normal classes) and when the system loads these various and successive definitions of the (same) entity it re-configurates dynamically these entities to extend the attributes definition so as to include the additional attributes to be taken into consideration. Figure 6 displays the structure of the Entity "Building" - in Express-G - according to the Global product Model (GPM).
The attributes bear track of the PtMs, VtMs or VoMs where they were defined, therefore leading to the possibility to filter the information later accordingly.

For example, the Figure 7 displays the same Entity "Building" - in Express-G - but this time according to the VtM1 definition. One can easily on the hardcopies how the user may switch from the GPM (generic core) to any of the View type Models.
Once a certain number of these Express files have been loaded (corresponding to the PtMs, VtMs and VoMs defined), a core model is available to the user who may generate instances of these classes according to his own needs.

These instances will inherit attributes throughout the inheritance relationships the entities (used as prototypes to create the instances) are involved in. Then the instance is created and each attribute (local or inherited) keeps track, if appropriate, of the PtMs, VtMs or VoMs it corresponds to. This leads to the filtering mechanisms made available on the instances by means of a specific Application Programming Interface (§ 2.4) extending SDAI functions for manipulating the information according to the PtMs, VtMs or VoMs defined.

Of course, the composition of views is supported by the environment. For example, if an instance has...
attribute1 and attribute2 defined in VtM1 (f1) and slot2 and slot3 defined in VtM2 (g1) then a composition of views can be used, i.e. \( f_1 \circ g_1 \) where the instance will display the only attribute3 remaining in this particular new view. Any arbitrarily complex composition can be defined and reused as a new view in the environment.

Finally, any instance can be exchanged according to any view(s) - provided that the instance is defined - by means of STEP based technology (STEP Physical File Format - SPFF).

This is illustrated by means of Figure 8 where the SPFF has been produced for the VtM1 first and then for the GPM. One can easily notice that for this extremely simple example, the relationships between the Building entity and the Building_State and Building_System entities exist in the GPM but are irrelevant to the VtM1.

**STEP;**
**HEADER;**
FILE_NAME('/users/brisson/VtM1.iso',
'1994-01-08 T15:53:52',('VtM1 ISO'),('CSTB'),'STEP VERSION 2.0 e',
,, 'APPROVED BY ');
FILE_SCHEMA(('VtM1'));
ENDSEC;
DATA;
#1 = Building_State('Building_State');
#2 = Building_System('Building_System',$);
#3 = Building('Building','ATLAS_Building');
ENDSEC;
ENDSTEP;

**STEP;**
**HEADER;**
FILE_NAME('/users/brisson/GPM.iso',
'1994-01-08 T15:52:43',('GPM ISO'),('CSTB'),'STEP VERSION 2.0 e',
,, 'APPROVED BY ');
FILE_SCHEMA(('GPM'));
ENDSEC;
DATA;
#1 = Building_State('Building_State');
#2 = Building_System('Building_System',$);
#3 = Building('Building',10000,'ATLAS_Building',(#2),(#1));
ENDSEC;
ENDSTEP;

Figure 8. XVoM - SPPF for VtM1 and GPM schemas.

Figure 9. XP-SDAI a multi-repositories implementation platform

Of course, as far as the XP-SDAI platform is a multi-repository environment, various repositories can exist concurrently. This is illustrated by Figure 9, where the repository entitled ATLAS contains in our example an implementation of the Building Project type Model (delivered by TNO). Other repositories
exist simultaneously, such as the repository and repository-1 in the same example, and the user may switch from one to the other, still using the view mechanisms defined within each one of these repositories.

Let us summarize some of the advantages of the methodology introduced. A very positive point is that the entire life cycle of the information is covered, starting from the CASE tools involved at the conceptual stages where the definition of the models is undertaken, passing by the production of the entities (STEP Technology) with the overloading mechanism aforementioned (developed a top of SDAI), ending with the generation of the corresponding instances.

Moreover, all the operational models produced (i.e. classes and instances) are object oriented and this leads to a very efficient Object Oriented management of the systems' content by means of the API proposed, with straightforward connections capabilities to Object Oriented databases whenever necessary (e.g. O2).

As this module is an SDAI implementation 100% compliant with the ISO part 22 documents, the communication with other systems is supported by means of neutral technologies (i.e. STEP based exchanges) and the SDAI Models content can be exchanged in a global or in a selective manner according to the PtMs, VtMs or VoMs defined. This last capability generalizes the functionalities prototyped during the COMBINE I project experiment (§ 1.2) and provides generic routes to support the exchange of project models content (PMs) or view models content (VMs) according to the original views defined or to any combination of them.

Finally, the API developed to handle the views in the operational environment can be understood as an extension to the normal operations of an SDAI interface, as it could be implemented by means of a Lisp binding or a C binding.

Therefore, XP-SDAI supports the management and access to its information structures throughout a Lisp binding or a C Binding of the "normal" SDAI specification but also defines the API extension of XVoM as a natural extension to the basic SDAI specification, supporting an access and manipulation of the data oriented by PtMs, VtMs or VoMs mechanisms.

2.3. The XP-SDAI implementation including View Management

XP-SDAI is a 100 % SDAI C Binding (or LeLisp Binding) whic integrates two main features:

Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ / 25
- dynamic reconfiguration of the Express entities (overloading, filtering of entities by means of loading Express file into the current SDAI session);

- SPF facilities keeping track of the entities information in order to generate an instance according to any Express definition.

The extended SDAI (C Binding) functions are:

```c
void saveSPFView (char* filename, SDAIModel* model, SchemaDefinition* viewschema) ;
```

This function takes its possible to generate a Step Physical file into the file `<filename>` of a set of SDAI Instances stored in the model `<model>` according to the Express Schema `<viewschema>` different from the one currently associated to the SDAI model `<model>`.

```c
void loadSPFView (char* filename, SDAIModel* model, SchemaDefinition* viewschema) ;
```

This function takes its possible to load a Step Physical file `<filename>` in the model `<model>` according to the Express Schema `<viewschema>` different from the one currently associated to the SDAI model `<model>`.

The D302a Annexes deliverable provides SDAI C Binding code examples.
3. The XVcM Converter

3.1. Introduction

It is well recognized that one of the main dimensions of the integration problem is the data axis (Schefström, 1993) and semantic data models appear as one of the routes leading to better capabilities in that respect. Hereafter, we shall consider Project type Models or View type Models as natural extensions to the concept of product models, themselves regarded as semantic models.

In order that semantic models play the role they are expected to have in the general integration framework (together with the control and man-machine interface dimensions), various approaches may be considered. One of them is to expect them to serve as a neutral and reference core supporting the data exchange mechanisms. Applications can make use of these models in various ways depending on how far they "integrate" these models at the level of their internal application models. If the applications make direct use of the reference model (e.g. which can be a STEP AP), and provided that they benefit from a STEP pre and post file processor, they will be completely integrated along the data dimension.

However, this situation is likely to be just an unrealistic dream resulting from a flawed assumption: the wide and rapid use of a limited number of standard reference models. Even though APs (in various industrial sectors) deliver reference core models, and even in the favorable case where these models would be largely accepted and re-used by the software houses, only a partial degree of commonality between the various computerized representations would have been achieved.

This would result from the fact that applications would still manipulate information outside of the common kernel definition and an efficient exchange of information would still require (even in this extremely favorable scenario), from place to place, translation and conversion of parts of the semantic models.

The migration path may even proceed at a slower pace. In fact, making the assumption that software vendors adhere to the STEP technology (integrating pre and post STEP file processors in their product lines - or better SDAI interfaces), without fully integrating reference product models delivered by APs should already be considered as quite a challenging and positive perspective. Complete APs implementations will certainly often be considered as leading to very expensive modifications of existing software by software houses, and moreover STEP offers orthodox ways to limit the implementation effort.
by means of the mechanism of conformance classes.

The main factor on which depends the success of the integration along the data axis is the capability to develop, rapidly and at a limited cost, intelligent and versatile converters able to map the different application semantic models. One may guess that the more the reference models will be integrated into the software applications, the less these converters will be used, but in any case and whatever the integration scenarios may be, they remain one of the masterpiece for practical integration.

In any case, even if the software applications became in the future completely STEP compliant and use fully interpreted APs (in the sense of the ultimate mapping of APs models on STEP integrated resources), conversion mechanisms will still be required to ensure the inter-operability of the APs. This would be one of the ultimate merits of the View Conversion Mechanisms to contribute to help solve the problem of the inter-operability of APs in the STEP world.

The main goal of the View Conversion Mechanisms, in the context of the task 3200 of the ATLAS project, is to demonstrate how the conversion of a given semantic model or Application type Model (AtM) - (e.g. Autocad AEC is an AtM) into a target semantic model (or target AtM) can be performed.

In fact, whatever the model being considered, it is quite clear that the problem of the conversion of semantic models in the full meaning of the word, cannot be handled by a simple one to one mapping of the entities of one of the models into the corresponding entities of the other model. Various approaches can be envisaged to address this conversion problem, the first one being reported here in this deliverable D302a is the rule based approach. Others will be experimented in the further developments of this task.

It should be added that the conversion of models, mainly seen in this task as a means to support the integration of different actors involved in the same project, can also be considered as a route to handle the integration of software manipulating different models needing conversion. This also open the road to possible perspectives in the domain of AP inter-operability. In fact, each AP or each "brick" of an AP (as they could be referred to in the MARITIME approach - (Kock, 1993)) can also be considered as views. Ideas on this issue will be developed further later.

3.2. A Rule based approach

We present now briefly the syntax of the rule based language which has been developed and incorporated into the XPDI environment to experiment this functionality. One should add, that this effort is also
relevant for the integration of KB systems generally speaking, i.e. one of the major goals of the WP3. Without disclosing the work carried out in task 3100, it is worth noticing that the exchange of knowledge is one of the major endeavours to be achieved. This supposes that the knowledge, generally encoded within the rules of rule-base systems be exchanged in some way.

The approach followed in task 3100 is to transform the rules into semantic trees at parsing. This is what most of rule compilers do, except that these trees in our case are made of complex objects. For each rule of the grammar of the rule language are associated objects (i.e. classes) corresponding to the "objectification" of the language itself. Then once a rule is parsed, instances of such objects are created to form the semantic trees corresponding to the particular rule being parsed.

Of course, the Express model of the classes and the ISO neutral file corresponding to the instances produced during the construction of a given semantic tree can be exchanged, this opening the road to the real exchange not only of information on the products but also of knowledge on the products or the processes. This is one of the efforts done in task 3100 to support the integration of knowledge based systems.

Therefore, the rule language developed will have useful applications in both tasks 3100 and 3200. In the context of the task 3200 where we focus on the conversion problem, the syntax of the rules should be presented first before we give some examples of rules used to make a conversion attempt (between COMBINE and ATLAS). Here follows an example:

RULE Example:
let X, Z some man and
    T a man and
    Y the woman of X and
    W among {Jo , Jill} or a student and
    Bertrand something and
    K one of the sons of X and
    Tim the cousin of the 2 nd half-sister of Jo or the last gran-son of W

if X is Jill and
    X is of type man and
if for each A a man and
    B a woman
verifying that the age of A is greater than 18
then A is B and
if the name of the son of Y is alphaless than "andreas" or
if X is one of the son of Jo and
    there exists C a man such that A is a son of X
then
add "young" to the caracteristics of X.

Figure 10. A syntactical example of the expressiveness of the rule language.

This example aims at a pure syntactical illustration of the rule language capabilities and does not pretend to correspond to any meaningful action. It shows that the expressiveness of the language is certainly important enough to enable the users to describe their knowledge without resorting to the introduction of too much "code" into the rules, which would be harmful to the communication of knowledge by means of the rules.

Given the properties and expressiveness of this rule language (incidentally which can be updated whenever necessary given the technology used to develop it, i.e. ceyx-yacc a Lisp version of yacc), a first attempt has been made to convert a source product model into a target product model. Given the models available at the moment where the effort has been undertaken (and their corresponding instances : PMs), the models which have been selected were the Integrated Data Model (IDM) of the COMBINE project and the ATLAS LSE PtM and whenever possible the Building PtM (the Building Project type Model was not yet at an appropriate stage of development to completely support such an effort).

The goal of the exercise was to map the schoolbuilding project (a PM instance of the IDM schema) on a new instance of PM conforming with the ATLAS LSE PtM or Building PtM. From the beginning, one should mention that the exercise was hampered by the fact that the IDM on one hand and the ATLAS LSE PtM on the other are positioned at very different levels of genericity (the same thing applies on the Building PtM at the stage of development where we undertook this effort).

The ATLAS LSE PtM and Building PtM describe very generic concepts with a limited number of properties (LOTs) whereas the IDM is rather specific (moreover oriented towards energy efficient design and HVAC modelling) and goes to a level of detail where most of the elementary properties of the concepts are depicted (especially whenever HVAC oriented).
Nevertheless, and even though the example cannot be considered as a really meaningful and representative of what such kind of rule-based mapping process could offer, it demonstrates its feasibility but also enlightens the difficulties met. Let us now give some examples of rules and briefly present what they aim at.

RULE C-building->A-building
LET C-building A building IN SYSTEM combine
THEN
  CREATE A-building A building IN SYSTEM atlas
  EXECUTE cv-record-link (C-building , A-building).

Figure 11. Rule mapping a COMBINE building on an ATLAS building.

This rule is very simple and decides to map any occurrence of COMBINE building encountered in a given PM (i.e. here the schoolbuilding example) on an ATLAS building. Therefore, an instance of ATLAS building is created whenever an occurrence of COMBINE building is met (the two PMs co-exist in different XPDI repositories). Moreover a link is established between the two by means of an a-list ((Combine-building-1 Atlas-building-1)(Combine-building-2 Atlas-building-2))..

RULE C-building-spatial-system->A-elementary-space-group-of-section-space
LET
  C-building-spatial-system A building-spatial-system IN SYSTEM combine AND
  C-building the building OF C-building-spatial-system AND
  A-building APPLY cv-get-link (C-building)
THEN
IF THE C-building-spatial-system HAS is-composed-of-space
THEN
  CREATE A-building-space-system A building_space_system IN SYSTEM atlas
  ADD A-building-space-system TO consist_of OF A-building
  CREATE A-section-space A section_space IN atlas SYSTEM
  ADD A-section-space TO consist_of_section_space OF A-building-space-system
  CREATE A-elementary-space-group A elementary_space_group IN SYSTEM atlas
  ADD A-elementary-space-group TO consist_of_elementary_space_group
  OF A-section-space
  EXECUTE cv-record-link(C-building-spatial-system, A-elementary_space_group).
The rule RULE C-building-spatial-system->A-elementary-space-group-of-section-space executes the following action: if within the COMBINE PM (i.e. schoolbuilding) the content of the slot "is-composed-of-space" of the entity building-spatial-system is not null, then the rule creates for the ATLAS target the following entities:

- "building_space_system"
- "section_space" to be added to the slot "consist_of-section_space" of the "building_space_system"
- "elementary_space_group" to be added to the slot "consist_of_elementary_space_group" of "section_space".

Moreover the link between the entity "building-spatial-system" (i.e. occurrence of entity) of COMBINE and the entity "elementary_space_group" of ATLAS (i.e. occurrence of entity) is established (a-list).

RULE C-space->A-elementary-space-of-section-space

LET

C-building-spatial-system A building-spatial-system IN SYSTEM combine AND
C-space THE is-composed-of-space OF C-building-spatial-system AND
A-elementary-space-group APPLY cv-get-link (C-building-spatial-system)

THEN

EXECUTE cv-remove-link(C-building-spatial-system)
IF THE A-elementary-space-group IS DIFFERENT OF NIL
THEN
CREATE A-elementary-space A elementary-space IN SYSTEM atlas
AFFECT has-space-function OF C-space TO has_elementary_space_function OF A-elementary-space
ADD A-elementary-space TO consists_of OF A-elementary-space-group
EXECUTE cv-record-link (C-space , A-elementary-space).

The RULE C-space->A-elementary-space-of-section-space executes the following action: given the content of the a-list, if the entity "building-spatial-system" of COMBINE has a link with the entity
"elementary_space_group" of ATLAS then the content of the slot "is-composed-of-space" of "building-spatial-system" is not null and in such a case the rule creates the following entity: "elementary-space".

Moreover "elementary-space" is added to the slot "consists_of" of the entity "elementary_space_group". The slot "has-space-function" of "elementary-space" will have the value of the slot "has_elementary_space_function" of the COMBINE entity A-elementary-space.

The link between "building-spatial-system" of COMBINE and "elementary_space_group" of ATLAS is systematically suppressed in that rule. Moreover, "building-spatial-system" can possess two sorts of links: either with "elementary_space_group" of a "section_space", or with a "elementary_space_group" of a "storey_space".

Once the first possibility is handled the link is destroyed (if it existed) to remove any ambiguity for the second case.

RULE C-building-spatial-system->A-elementary-space-group-of-storey-space

LET
   C-building-spatial-system A building-spatial-system IN SYSTEM combine AND
   C-building the building OF C-building-spatial-system AND
   A-building APPLY cv-get-link (C-building)
THEN
   IF THE C-building-spatial-system HAS is-composed-of-storey
   THEN
      CREATE A-storey-space A storey_space IN atlas SYSTEM
      ADD A-storey-space TO consist_of-storey_space OF A-building-space-system
      OF A-building
      CREATE A-elementary-space-group A elementary_space_group IN atlas SYSTEM
      ADD A-elementary-space-group TO consist_of OF A-storey-space
      EXECUTE cv-record-link (C-building-spatial-system,A-elementary_space_group).

Figure 14. Rule mapping C-building-spatial-system->A-elementary-space-group-of-storey-space.

This rule is similar to the previous rule (Figure 6) except that it handles the "elementary_space_group" of a "storey_space".

Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ / 33
RULE C-space->A-elementary-space-of-storey-space

LET
   C-building-spatial-system A building-spatial-system IN SYSTEM combine AND
   C-space THE is-composed-of-space OF C-building-spatial-system
      IN SYSTEM combine AND
   A-elementary-space-group APPLY cv-get-link (C-building-spatial-system)
THEN
   EXECUTE cv-remove-link (C-building-spatial-system)
   IF THE A-elementary-space-group IS DIFFERENT OF NIL
      THEN
         CREATE A-elementary-space A elementary-space IN SYSTEM atlas
         AFFECT has-space-function OF C-space TO has_elementary_space_function
            OF A-elementary-space
         ADD A-elementary-space-group TO consists_of_elementary_space_group
            OF A-section-space
         EXECUTE cv-record-link (C-space, A-elementary-space).

Figure 15. Rule mapping C-space->A-elementary-space-of-storey-space.

This rule is similar to the previous rule (Figure 6) except that it handles the "elementary_space_group" of a "storey_space".

The mapping of structural elements between a COMBINE PM and an ATLAS PM is not possible for the simple reason that the slot "structural-element" of the entity "Building-fabric-system" is not defined.

As far as the elements and the components are concerned, the situation is the following. ATLAS offers two sorts of systems: the systems dividing the space "space_dividing_system" corresponding to internal wall and floors and the systems enclosing the space "space_enclosing_system" corresponding to the roofs and external walls.

RULE C-building->A-space-enclosing-system-and-space-dividing-system

LET
   C-building A building IN SYSTEM combine AND
   A-building APPLY cv-get-link C-building
THEN
CREATE A-space-enclosing-system A space_enclosing_system IN SYSTEM atlas
ADD A-space-enclosing-system TO consists_of OF A-building
CREATE A-space-dividing-system A space_dividing_system IN SYSTEM atlas
ADD A-space-dividing-system TO consists_of OF A-building.

Figure 16. Rule mapping C-building->A-space-enclosing-system-and-space-dividing-system.

This rule handles the mapping between a Building of the COMBINE PM and the Space decomposition of an ATLAS PM. The "space_dividing_system" and the "space_enclosing_system" are created corresponding to a Combine Building "C-Building".

3.3. Hints to future conversion strategies

The classical computer science approach to conversion / translation problems is based on rewriting systems. In some sense, what has been proposed in the previous conversion effort is also based on rewriting rules. Each one of the first order logic rules that we have implemented during this experiment are elements of a rewriting system based on the rule language developed.

Other programming techniques could be envisaged (and will in a future version of the deliverable) such as constraint based programming or propagation and maintenance of relations between interdependent objects. We will also report on more generic approaches aiming at the development of systems bringing methodologies and tools supporting the creation of converters (compilers of convertors).

On the other hand, the TNO CAD center will also contribute to the view conversion effort and they present the problem in the following manner : given a set of data structured according to an Express schema (either in a STEP physical file or in a database accessible via SDAI), how can this set of data be converted into another set of data structured according to a different (but somehow related) Express schema ?

The goal of this part of task 3200 is to gain insight in the types of conversions occurring within the ATLAS project, i.e. between typical AtM, VtM and PtM models. Possibly a classification of these conversions can be made, and recommendations on how to specify and implement conversions are needed. The problem will be tackled as follows. First, two specific ATLAS conversions will be studied and implemented :
1. Conversion from AtM Speedikon Architecture to VtM Architect in Global Design Stage

2. Conversion from VtM Architect in Global Design Stage to BPtM/LSE PtM kernel.

The implementation will be done in the TNO PMshell environment, using C++ to do the actual programming. The purpose of this exercise is twofold: it can yield information for both the implementation of the first ATLAS demonstration and the analysis of the conversions.

Second, the examples will be used to do some kind of "reverse engineering". The conversions will be analysed and the types of conversions will be classified.

As much as possible the need for bi-directional conversions will be taken into account. Several ways to (formally) specify and implement conversions are investigated and recommendations will be given.

This will be done looking closely at work done at CSTB on view management and rule-based view conversion, but also other developments will be evaluated (e.g. EXPRESS-M, which has been submitted to ISO STEP).

The expressiveness of the rule based language looks very extensive and powerful. It is possible to use complex constructions to define the conditions for a sub-structure of model and carry out conversion or creating relations. Most of the other tools/techniques for conversion have problems with the conversion of relations or creating relations because they convert entities/objects separately instead of sub-structures. Another advantage of the XVcM convertor will be the possibility to use the link between a source entity and its target entity. With this it is possible to create a relation during a conversion with an entity which was created before by a conversion in another rule.
4. FRAMEWORKS, MODELS AND APs INTEROPERABILITY

4.1. Introduction

As it has been suggested in the previous sections of that deliverable (e.g. § 3.1.) one of the possible outcomes of the data abstraction, generalization and view conversion facilities is to help solve the problem of inter-operability of different APs. Before delving further in that subject, it is worthwhile to recall the general objectives of STEP and the overall architecture proposed by the STEP community for the development of the resources, models, APs, etc.

"STEP - ISO 10303 - is an International Standard for the computer-sensible representation and exchange of product data. The objective is to provide a mechanism capable of describing product data throughout the life cycle of a product, independent of any particular system. The nature of this description makes it suitable not only for file exchange, but also as a basis for implementing and sharing product databases and archiving."

ISO 10303 is organized as various parts which fall in the corresponding series: Part 1 - Overview and Fundamental Principles, Series 10 documents - Language Definition (Express), Series 20 documents - Implementation Documents, Series 30 documents - Conformance Testing Documents, Series 40 documents - Generic Integrated Resources (GIRs), Series 100 documents - Application Integrated Resources (AIRs), Series 200 documents - Application Protocols (APs), plus the AIC's Application Interpreted Constructs.

As far as the STEP models are concerned, one can easily observe that these resources are organised along a generalization / specialization axis. The parts of the serie 40 offer the more generic STEP resources (GIRs) available, then come the resources useful for various type of applications (AIRs) and finally the Application Protocols satisfy the objectives of industry specific needs. A closer inspection into each of these series reveals that their content is also organised in a similar way, moving from the general to the specific. For example, the GIRs show that Part 41 presents the fundamentals of product description whereas the Parts 45, 46 or 47 address more specific topics such as materials, visual presentation or shape tolerances.

As long as the content of the Parts remains separate with no (or say as little as possible) overlap between
the different levels along the generalization / specialization axis, the use of the resources remains rather simple. Things get more complex as soon as the APs come in place. With the risk of oversimplifying, let us say that people define the models required for specific industry needs and that the ARMs section of the APs reflect these needs. Then the interpretation process aims at performing the mapping between the entities (and attributes) of the ARMs and the resources offered by STEP. This process is technical and its description is not relevant here, but one should keep in mind that the need to extend the integrated resources made available by STEP for the sake of the mapping exercise may arise and that similar extensions may also be suggested by different APs leading to the AICs.

Of course, various incoherencies may arise from the extensions envisaged by the APs teams. Therefore the STEP guidelines suggest a very rigorous organisational framework to proceed with the AP development process in order to master this problem. A stated by (Palmer, 1993) << The STEP integration framework establishes an explicit architecture for the conceptual models that are part of ISO 10303. This architecture provides the structure for the integrated resources and application protocols. The integrated resources provide constructs that are independent of a specific data application context. These constructs are used for developing the application interpreted models of application protocols. >>.

Moreover the emphasis is placed on the consistent usage of the integrated resource constructs in the same document: << All APs shall use the integrated resource constructs consistently. If two APs have the same information requirements, these APs shall use the same resource constructs for the common requirements.>> and further << The reuse of interpreted constructs provides the mechanism for ensuring that the relevant product data can be exchanged between APs and allows for interoperable APs. This library of shared application interpreted constructs (AICs) shall be used as a resource for defining the AIMS of APs. The WG4 AP integration Project is responsible for maintaining this library. >>.

The guidelines for the development and approval of STEP Application Protocols clearly illustrates this emphasis on an organisational response to the AP interoperability problem. The correctness of the reuse of the interpreted constructs and their proper incorporation into the AICs is of the responsibility of the WG4 AIM Development Project (ADP) and of the WG4 AP Integration Project (APIP). Each one of these teams has to be formally solicited by the AP development leader at some stage of the AP development process.

Even though this organisational schema is certainly very relevant, one may wonder whether it may be sufficient to maintain the consistency of the overall set of models / resources, and moreover if it is completely appropriate to satisfy all the needs arising from the interoperability requirements of various
software applications implementing parts of several distinct APs. In fact, situations in which APs may wish to extend the same resource but with slightly different properties (attributes) according to the views considered by each one of the AP is probable. In such a case, the approach followed by the task 3200 "Data abstraction, generalization and view conversion" is perhaps of interest.

4.2. Task 3200 as a support to the APs interoperability problem?

We shall recall the methodology described in the section 2.2. where each one of the conceptual models developed is first kept separate and where a number of these models can be assembled later for a given purpose. In that modelling framework, the same entities may appear at different levels of abstraction either for the sake of the modelling effort (and they are later connected by means of the external NOLOT feature) or for true generalization / specialization purposes (entities appearing in the models being more specific representing specializations of those contained in the models being more generic).

Moreover, models at the same level of abstraction and therefore located at the same abscissa along the generalization / specialization axis may contain slightly different definitions of the same entities. This means that the specializations with respect to one model may differ with those required for another model at the same level of abstraction. We shall name this property the Heterogeneous Structural Differentiation at Specialization (HSDS).

Nothing surprising there, really, except that this is absolutely conflicting with the classical and orthodox STEP vision as described in the previous section. One should remember that whenever similar constructs have to be used by at least two different APs the solution proposed is to "upgrade their status" at the level of AICs while harmonizing their definition. One might think that the goal is to prevent any kind of HSDS as it could lead to incoherences given the current status of the series 20 guidelines.

HSDS is not a usual feature of the object oriented environments as such, but is quite a natural outcome of the OO paradigms such as encapsulation, inheritance, polymorphism and overloading (i.e. this time not structural but functional). Encapsulation states that the structure of the data is hidden to the programmer and therefore the access to the data is granted by a late functional binding (referred to in that context as - functional - overloading) determined the functional polymorphism which may be encountered along an inheritance path. Presented in that light, it is just a way of accommodating by means of implementation strategies some kind of Heterogeneous Structural Differentiation at Specialization (HSDS), except that the underlying data structures remain hidden.
What is proposed by our paradigm, i.e. HSDS, is to provide some kind of late structural binding offering a late structural overloading. The functionalities are still supported at the implementation level (refer to section 2.2) and this brings some similarities with the OO approaches. An entity may be defined initially at a rather generic level (e.g. GIRs or AIRs) and then further specialised at more specific levels (e.g. APs) in different ways according to the models concerned (i.e. the different APs). What the HSDS capability offers is to perform a structural overloading of the generic construct with the more specific ones, still keeping track of where the extended properties have been conceptually defined (i.e. to which AP belongs which property for which construct), each one of the APs acting as a view on the global model.

This strategy is precisely the one supported by the View Manager system developed, and the same entities can be defined in different PtMs or VtMs and when the system loads these various and successive definitions of the (same) classes it re-configurates dynamically these constructs to include the additional attributes to be taken into consideration. The slots bear track of the PtMs, VtMs or VoMs where they were defined, therefore leading to the possibility to filter the information later accordingly. This supports the heterogeneity at the same level of abstraction.

The second mechanism offered by the View Manager system is to support heterogeneity along the generalization / specialization axis while instanciating the constructs. When an instance of an entity is created, attributes collected along the inheritance path supplement those existing at the abstraction level where the construct considered for instanciation is taken. This supports the heterogeneity at different levels of abstraction. Each one of the abstraction levels along the generalization / specialization axis is considered as a view and each one of the separate specializations at the same abstraction level is also considered as a view supported by the filtering functions described in section 2.4.2.

It should be quite clear that these mechanisms could represent hints to potential solutions to the AP interoperability problem. Even though the approach share some ground with the Object Oriented philosophies it also strongly depart from some of their basic assumptions. The encapsulation is not any more considered as a required nor completely desirable property. As the data models should be made explicit for the sake of exchanging the information accross software systems, the structural description of the entities is of the uttermost importance. This put a strong emphasis on the conceptual modelling phases but also on the structural description of the entities. We call "structural" the explicit description of the attributes and therefore relationships a construct has at a given abstraction level. This explicit structural description is in no way hidden and seems to depart from the encapsulation principle of OO techniques.

Moreover heterogeneous specializations should be made possible along the generalization / specialization
axis and the polymorphism envisaged here is more structural than functional. This structural polymorphism should occur along the inheritance path but also at the same abstraction levels when various models (APs) have complementary (and concurrent) definitions of the same construct which moreover should be used to perform the structural overloading of the same entity also defined at a more generic level.

Our implementation guidelines (refer to section 2.2) have clearly separated the structural overloading occurring at the same abstraction level and hence supported by the dynamical overloading of the classes definitions and the structural overloading occurring along the inheritance path (generalization / specialization axis) triggered when the instances of the constructs are created.

The view mechanisms supported by the View Manager on the global project model enable the user to extract specific set of data according to given perspectives (e.g. disciplines, actors, but also APs are examples of such perspectives). In some sense these filtering mechanisms on the global project model offer some kind of encapsulation as the user of the project model sees the data by means of the view facilities without really knowing the detail of the final organisation of the data which remains hidden as the encapsulation principle would recommend.

The overall framework developed in the course of this task comprising the modelling guidelines and the implementation strategies will be from now on referred to as the Data Abstraction and View Oriented System (DAVOS).

4.3. The AEC Framework example

How does this DAVOS modelling and programming paradigms help solve the inter-operability problems encountered in a modelling framework ?. The proposed AEC framework example released in Phoenix in January 1994 (Figure 10) is certainly of interest for various reasons, on one hand because of the complexity and the number of models involved (whatever they can be GIRs, AIRs, AICs, APs) is large enough to raise inter-operability concerns and on the other because most of the models visible in this frame fall in the scope of the ATLAS project and some of them could be directly adapted from ATLAS results. This is true for some of the proposed AICs (e.g. LSE PtM - D106-Ib, Building PtM - D106-IIa - TNO lead), but also of some the potential APs of the sector (e.g. HVAC VtM - D206a - CSTB lead).

A closer inspection at this figure shows that the first axis (abscissa) is the generalization / specialization axis where models at different levels of abstraction have been mapped and the other axis is more or less
the actors / disciplines dimension directly related to the generic concept of views developed in this task (refer to section 2.1). This means that the DAVOS modelling methodology and the implementation mechanisms proposed (already supported by the software tools of the XPDI Station) could suit well the practical needs of the modelling effort to be undertaken in ISO / STEP WG3/TC12 Committee. A remark is that the framework depicted in Figure 10 is just one of the possible frameworks of interest to the AEC community certainly not the last !.
Figure 17. The AEC IARs / AICs and APs Framework (Gielingh, 1994).
5.1 Introduction

5.1.1 Context within the ATLAS project

Within ATLAS, there is a general (and generic) part which can be used for all LSE projects, i.e. in building as well as process industry projects. This is the ATLAS LSE Project type Model (LSE PtM). For each application area there is a specialisation of this general model, for instance for building applications the ATLAS Building Project type Model (Building PtM or BPtM). Within these models (both the LSE PtM and its specialisations), several View type Models (VtMs) exist which describe how certain "actors" 1 look at a product or project in a certain stage of the life cycle. Finally, for each existing software application a model is constructed of the information serving as input or output of the application. These are called Application type Models (AtMs). They are not part of the neutral model.

Because of the multitude of models in the ATLAS project, several types of transformations between these models occur:

- **translation**

  Translation means that there is a transformation of one (data) format into another, with only a difference in syntax and no difference in semantics. Example: the translation of a DXF file to a STEP file (with the same semantics).

- **conversion**

  With this type of transformation there is no difference in the syntax used, but there is a difference in semantics. Example: the conversion of a STEP file of a specific application (AtM) to a STEP file according to some view (VtM).

- **generation**

  By generation we mean the automatic generation of software using a specification in some formal language. Example: the generation of C++ class definitions from an EXPRESS schema.

The TNO CAD Centre will focus on and contribute to view conversion.

---

1 An actor is an abstract person or group of persons responsible for a certain task.
5.1.2 The conversion problem

This problem can be summarised as follows. Given a set of data (either in STEP physical file or in a database accessible via SDAI), structured according to an EXPRESS schema. How can this set of data be converted into another set of data structured according to a different (but somehow semantically related) EXPRESS schema? Within the ATLAS project, this applies both to

- AtMs and VtMs, and
- VtMs and the LSE PtM (and BPtM).

The problem within the ATLAS project is that for constructing conversion software taking care of the necessary conversions, currently no satisfactory method exists. It is, however, possible to implement conversions by writing for instance C++ programs by hand. This has a number of disadvantages:

- programming (by hand) is a lot of work;
- programming takes place on the "physical" level instead of on the "logical" level;
- one is forced to deal with implementation details;
- maintenance of conversions is very laborious;
- no use is made of common properties of conversions;
- conversions are not recorded using a standard notation (such as EXPRESS).

So the problem is twofold. First we need a method for realising conversions which does not suffer from the disadvantages as mentioned before. Second the necessary conversions in ATLAS must be realised.

5.1.3 Approach

The conversion problem is a very comprehensive problem. One of the difficulties is that it is not clear how large the semantical gap between two models can be. Therefore it is not easy to solve this problem in a general way, so it is a priori not clear that the solution will be applicable to all conversion problems. That is why we have chosen a more practical approach; a more fundamental research on transformations in general is carried out (Atlas D105).

The problem will be tackled as follows. First, a number of conversion examples (among which two specific ATLAS conversions) will be studied and implemented:
1. A conversion benchmark, originating from a research project carried out by TNO CAD Centre.
2. Conversion from AtM Speedikon Architecture to VtM Architect in Global Design Stage.
3. Conversion from VtM Architect in Global Design Stage to BPTM/LSE PtM kernel.

The implementation will be done in the TNO PMshell environment, using C++ to do the actual programming. The purpose of this exercise is twofold: it can yield information for both the implementation of the first ATLAS demonstration and the analysis of the conversions.

Second, the examples will be used to do some kind of "reverse engineering". The conversions will be analyzed and the types of conversions will be classified. As much as possible the need for bidirectional conversions will be taken into account. Several ways to (formally) specify and implement conversions are investigated and recommendations will be given.

5.1.4 Structure of the chapter

In the next paragraph, the three conversion examples will be introduced. In paragraph three, formal and informal specification of conversion will be treated. Ways for formal and informal specifications will be explored, and the examples are used to try out some of the options. Only some extracts of the three examples are given to present the ideas; complete listings of the formal and informal specifications are included in the D302a Annexes document. A similar structure is used for the following paragraphs on implementation in C++ and Prolog, EXPRESS-M and Transformr. C++ and Prolog are well known programming languages. EXPRESS-M and Transformr are conversion tools originating from the STEP community. Paragraph 7 deals with miscellaneous methods for conversion which have been analyzed in less detail because less information one these methods was available, or the methods are not yet fully elaborated. In paragraph 8 an attempt is made to give a classification of conversions, again using the three examples from paragraph 2. Paragraph 9, finally, gives conclusions of this deliverable and recommendations for handling the conversion problem in ATLAS.
5.2 Conversion examples

5.2.1 Introduction

This paragraph introduces the three conversion examples. Two of them are ATLAS conversions; one of them originates from a research project carried out by TNO CAD Centre. From the latter model an explanation is given; the ATLIAM and EXPRESS schemas are listed in Annexe A. For the ATLAS conversions, a reference to ATLAS documentation is given.

5.2.2 A conversion benchmark

Next to the ATLAS conversion examples introduced in this paragraph, an additional conversion example is used. The motivation for using this example is:

- The ATLAS conversion example concerns large and complex models. It takes a lot of effort just to understand the models and their relation, before one can study the conversion between them.

- A problem with the first ATLAS conversion example (from VtM to AtM), from a conversion point of view, is that the structure of both models is very similar. A conversion between those models will not show any interesting problems.

- A problem with the second ATLAS conversion example (from VtM to BptM / LSE PtM) is that at this time the BptM and LSE PtM are not sufficiently detailed (i.e. no (AIM) EXPRESS schema is available where all low level attributes are specified) to consider this a meaningful conversion.

- The additional conversion example, which originates from a research project carried out by TNO CAD Centre, has proved to be an interesting conversion example because
  • it is a realistic example because it concerns a real problem that had to be solved in another project
  • the models involved are relatively small and easy to understand, but have sufficient diversity or complexity
the conversion shows a number of interesting problems which we believe to be typical conversion problems.

We will now explain the conversion example. The context of the conversion problem is the exchange of data from a CAD system and an NC programming system. The CAD system is used to make 2D technical drawings for the design of so called prismatic parts (a class of products with no free formed surfaces, which can be fabricated by removing material from a raw block of material by drilling and milling). In order to facilitate exchange of geometrical data, the CAD system has been extended with a shell which stores geometrical data on a relatively high conceptual level (i.e. in terms of views and contours, instead of only lines, arcs, etc.). The information model of the shell is called the technical drawing model. The input (file) of the NC programming system (called GEO format) has also been modelled in EXPRESS.

The way in which contours are described in the technical drawing model and in the NC programming model is, however, entirely different. This is what makes the conversion between them worth investigating. Both models are explained in the following sections. Refer to Annexe A for the EXPRESS schemas and ATLIAM diagrams. The conversion will be elaborated in the succeeding paragraphs.

5.2.2.1 A model for technical drawings

The model for technical drawings describes a technical drawing in terms of how a designer thinks about a drawing, i.e. in terms of views, contours of an object, etc. This means that a drawing is modelled on a higher conceptual level than for instance in a DXF file, where we only have lines, arrows, texts etc. In DXF the overall structure of the drawing is lost. In addition, the model has been simplified significantly by leaving out most information which is not used in the conversion. This concerns information about annotation (reference symbols, dimensions, tolerances etc.).

In the modelling process, simplicity of the model was our main goal. The model should be easy to understand. That is why certain properties of technical drawings are not modelled exactly the way they are. For instance, a technical drawing can consist of several views, but only one view can be the main view. This constraint has not been modelled because it would unnecessarily complicate the model. We assume that, instantiating the model, these things are taken into account. This strategy has been followed for the entire model.
Diagram 'main'

A technical drawing has a border, an information block (which contains information like author, date, version) and several drawing parts, which can be views, cross sections or details. A view can be main, left, right, etc. Cross sections and details are identified by a text, and details may be enclosed by a border.

Diagram 'drawing part'

Every drawing part has exactly one outer contour and zero or more inner contours (e.g. holes or pockets). Edge lines are visible edges of the product which are not closed inner contours. Hidden lines are edges of the product which are not visible in the particular view but are drawn to clarify the shape. Hidden lines are optional and are usually drawn using dashed lines. The line type is not modelled because we suppose that for each kind of line, standard line type are used.

Diagram 'Basic entities'

This diagram contains all low level "visible" entities which in the end are the constituent parts of a technical drawing. All other diagrams are only used to denote the structure of a drawing.

A point has X- and Y-coordinates in two-dimensional space. A curve has a starting point and an end point, and is either a line or an arc. In case of an arc there are two more attributes: the centre point of the arc and the orientation (1 for counter-clockwise and 2 for clockwise). A polyline is an ordered set of curves, with the restriction that all curves form one continuous polyline. A contour is a polyline which is closed, i.e. it starts and ends in the same point. (These restrictions are not modelled.) A text, finally, has a location (a point in 2D space), a height, a rotation angle (relative to the X-axis), the actual text (a string) and a font.

5.2.2.2 A model for the GEO format

The GEO model describes the input (file format) for an NC programming system (Dlog). Because we are only interested in the conversion from technical drawings to the GEO format, only the relevant part of the GEO format is modelled (i.e. only geometry; not machine data etc.).

The information which will be converted from technical drawing to GEO concerns geometry and contours. Because Dlog is a 2D NC programming system, every view instance in the technical drawing model must be translated to a separate instance of the GEO model. This instance only contains
information concerning geometry and contours; all other information (borders, administrative information, hidden lines, cross sections, details, etc.) is discarded.

Diagram 'file'

On the top level, a GEO file consists of four kinds of information. First, a so called window to indicate the area enclosing all geometric elements. Second, a list of geometry elements which can later be used (referred to). Third, a list of contours, referring to geometry elements previously defined. And fourth, a list of so called point groups which are used for inner contours which are a perfect circle, thus representing holes which can be drilled.

Diagram 'geometry'

Dlog recognizes three geometrical elements: point, line and circle. Every element is defined separately and is identified by a (per element type) unique number. Every element has an X-coordinate and Y-coordinate, which are interpreted relative to a reference point (referred to by its number). In our example we always use reference point zero (the origin (0,0)).

A point is simply defined by the X- and Y-coordinates. A line is defined by a point and an angle in degrees, relative to the X-axis. A circle is defined by the centre point and the radius.

Diagram 'contour'

Contours and point groups are the essential information in a GEO file. Contours are composed referring to geometry elements defined earlier. A contour always has a starting point and an end point. Furthermore a contour consists of a list of one or more elements. Each element can be either a circle element or a line element. Each element refers to a circle or line defined in the geometry section. Just like in the model for technical drawings, a circle has an orientation.

When there is a transition from a line element to a circle element or vice versa within a contour, the kind of transition (relative to the previous element) is denoted by the variable 'direction'. This variable can have the following values: 1 (for tangential), 2 (for right), 3 (for left), 4 (for top) or 5 (for bottom).

Examples:
If a line and a circle have two intersection points, the right point of intersection has to be stated. With perfectly vertical lines, 'top' and 'bottom' are used; for all other lines, 'left' and 'right' are used.
Diagram 'point group'

Full circles are described by both a contour and a so called point group. The contour description is used when a hole is milled; the point group description is used when the hole is drilled (or a similar process). The holes may be grouped by diameter (so the same drill can be used for all these holes). For all these holes the coordinates of the centre point should be specified. These are absolute coordinates, i.e. not relative to some reference point.

Diagram 'window'

As stated before, a bounding box or window for all geometry should be specified in the GEO file. The window is defined by the bottom left point (X- and Y-coordinate) and top right point (I- and J-coordinate).

5.2.3 From VtM to AtM

An important aspect of the ATLAS models is the distinction between a model that is view neutral (i.e. accessible for every discipline) and a number of separate view models, one for each discipline. The latter are called View type Models (VtM). An example of a VtM is a model providing a set of neutral entities and relationships which allow the Architect to describe and exchange Spatial Design data during the
Global Design Stage. From now on, this model is referred to as the VtM. (Atlas D106III)

Another type of model is the Application type Model (AtM). An example of an AtM is the model describing the input/output format of Speedikon Architecture, a CAD application designed by IEZ. From now on this model is referred to as the AtM. (Neuteboom, 1994).

The second conversion example comprises the conversion from VtM to AtM. In the remainder of this section we will give some more information on this example.

For both models, EXPRESS schemas are available, which are of course used as the basis for a conversions between them. The main objective to study this conversion is twofold. First, a clear insight into detailed conversion problems actually occurring in ATLAS is wanted. Second, the possibilities of implementing the conversion are researched, since the conversion is part of the first demonstration of ATLAS in June 1994.

The example is used to explore:

- informal specification of conversions (paragraph 3)
- implementation of conversions in C++ (paragraph 4)
- using EXPRESS-M for conversions (paragraph 5)
- using Transformr for conversions (paragraph 6)
- classification of conversions (paragraph 8)

An important issue for the conversion is the semantics of both models. The modelling method ATLIAM only describes the names of entities and the relations between them (attributes). An additional explanation should be available to explain the semantics of the models. Without this explanation it is not clear how to interpret the models, so realising a conversion between them becomes a dangerous task. The AtM is documented by the IEZ Speedikon Specifications manual. This manual, however, is not always clear and correct. For the VtM, no documentation was available. This problem was solved by discussing the semantics of the VtM with the people responsible for the model.

To give some rough idea about the conversion from VtM to AtM, the following table gives an overview of the translation of the source entities of VtM to the target entities of the AtM.

<table>
<thead>
<tr>
<th>from VtM</th>
<th>to AtM</th>
</tr>
</thead>
</table>

Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ / 53
5.2.4 From VtM to BPtM / LSE PtM

This section introduces the third conversion: from VtM to BPtM / LSE PtM. The VtM was explained in the previous section. The other models will be briefly discussed here.

The objective of the ATLAS Building Project type Model (BPTM) (Atlas D106IIib) is to provide a conceptual model of the entities and relations that describe building projects in such a way that different participants and different application programs can use the model as a common basis for interchange of building project data and knowledge. Building project knowledge, in the form of knowledge rules, is not part of the model itself, only the possibility for the (neutral) definition and exchange of knowledge rules (like constraints) will become available.

In order not to disturb the information interchange between different sectors of the Large Scale Engineering (LSE) industry, the Building PtM is derived as a specialisation of a more general model, the ATLAS LSE Project type Model (LSE PtM) (Atlas D106-Ic). If other sectors of the LSE industry also derive their PtM's as specialisation of the LSE PtM, communication between these sectors can be supported on the parent (LSE) level.
The objectives of the Building PtM are partly the same as the objectives of the ATLAS LSE PtM. The solutions proposed to meet these objectives however are now much more detailed. The LSE PtM for instance only contains general flow systems, but the Building PtM contains: HVAC-systems, electrical systems, water systems, etc.

Besides the general objectives, also a number of Building and Construction specific objectives have been taken into account. The main objectives of the BPtM are; to provide communication between partners belonging to different disciplines, and between project partners active in different project life cycle stages.

The idea to use conversion from VtM to BPtM / LSE PtM was that it might show conversion problems differing from the problems occurring in the conversion from VtM to AtM. During the process of making the informal specification for this conversion, however, it appeared that both the BPtM and the LSE PtM were not sufficiently detailed to make a conversion feasible. In ATLIAM terms, the models were not elaborated up to the level were LOTs are connected to the low level entities. To utilize the knowledge about the models (which developed during the specification process), TNO CAD Centre (after consulting with TNO Building research) devoted some time to propose expansion of the models with the appropriate LOTs. Unfortunately, at this time the BPtM and LSE PtM are still not yet sufficiently detailed so only an (incomplete) informal specification is available, but no implementation in C++. We did, however, take this conversion into account when making the classification of conversions in paragraph 8.

5.3 Formal and informal specifications

5.3.1 Introduction

This paragraph discusses the possibilities to give informal and formal specifications of conversions. In section 3.2, the informal specification of all three conversion examples is described. Different ways of informal specifications are explored; they are explained by using some extracts from the used examples; for a complete listing of the informal specifications, refer to Annexe B. In section 3.3 a formal specification of the conversion benchmark is given, after several methods for formal specification have been reviewed. Again only some extracts of the specification are used for explanation purposes; refer to Annexe C for a complete listing of the formal specification.
5.3.2 Informal specifications

All the conversions are specified from source model on the left hand side to target model on the right hand side. The notation of the informal specifications of conversion from VtM to AtM and from technical drawing to GEO are the same. For the third example (from VtM to BPtM / LSE PtM) a more EXPRESS-like notation is used to show the extensive (subtype) structure of the models involved.

5.3.2.1 From VtM to AtM

In spite of the similarity of both models there are two types of differences. First, the model of the VtM has more structure in contrast to the AtM. Second, the AtM uses references by name in contrast to the direct relations ("pointers") in the VtM.

The complete informal specification is listed in Annexe B. Here we give some examples for clarification purposes.

The method of the informal specification describes a direct one to one attribute conversion. The entities of the source model (VtM) are always standing on the left hand side and the target entities on the right hand side.

<table>
<thead>
<tr>
<th>VtM</th>
<th>AtM</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>grid_id</code></td>
<td><code>grid_number</code></td>
</tr>
<tr>
<td><code>x_axis_angle</code></td>
<td><code>x_axis_angle</code></td>
</tr>
<tr>
<td><code>y_axis_angle</code></td>
<td><code>y_axis_angle</code></td>
</tr>
</tbody>
</table>

There are some compulsory and optional attributes in the Application type Model (IEZ Speedikon). If an attribute is compulsory in the AtM then the attribute is not placed with the optional attributes, whether the attribute is compulsory or not in the VtM.

<table>
<thead>
<tr>
<th>VtM</th>
<th>AtM</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>building_id</code></td>
<td><code>project_number</code></td>
</tr>
<tr>
<td><code>building_name</code></td>
<td><code>project_name</code></td>
</tr>
<tr>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td><code>building_description</code></td>
<td><code>text</code></td>
</tr>
<tr>
<td><code>supervisor</code></td>
<td><code>supervisor</code></td>
</tr>
<tr>
<td><code>project_title</code></td>
<td><code>project_reference</code></td>
</tr>
</tbody>
</table>

It is possible that some optional attributes in the VtM are not optional in the AtM. The AtM attributes must get a default value if the VtM attributes have no value.
If there are no source attributes, default target attributes must be established.

- \( x\_axis\_text \) := 'x-axis'
- \( y\_axis\_text \) := 'y-axis'

Some VtM attributes can not be converted, because the are no AtM attributes to make the conversion possible.

- version := no AtM attribute

One important difference between both models is the direct entity relations (roles) in the VtM and the use of references by name in the AtM. An explanation describes the relations.

- (for storey) \( floor\_number \) := "storey_id of the Storey which is involved with this Grid"
- (for section) \( section\_number \) := "section_id of the Section which is involved with this Grid"

A Wall in the VtM consists of WallSegments; both are combined into one Wall_Definition in the AtM. This is an N to 1 relation. The symbol _ means a mapping from a source to a target entity.

### Wall _ Wall_Definition

- wall_id
  \( (wall\_id*100) + \) number of WallSegment \( wall\_number \)

Every Wall can have many WallSegments. Each WallSegment must have a WALL_DEFINITION with a copy of the attributes.

This example also gives us a calculation (sin, cos, +, -, *, /, square root, square, ) example.

An AtM attribute can be constructed from many VtM or AtM attribute values.

- \( wall\_name \) := 'D[wall_thickness]T[material]'

An example of a (conditional) type conversion is a mapping from STRING to INTEGER:
• wall_type = 'Enclosing'  wall_type := 10
= 'Dividing'   wall_type := 20

If a compact note is not possible then a comprehensive description is given.

• number_of_elements := number_of_elements + 1
(The only relation between a WALL and a B_E_PARAMETER (ELEMENT) is related by wall_number. Every time when there is an other B_E_PARAMETER (ELEMENT), which belongs to a specific WALL, the number_of_elements must be increased.)

A Ceiling in the VtM is built up of CeilingEdges, and a Roof in the AtM is built up of RoofEdges. The relation between a Ceiling and CeilingEdges is given by a role and the relation between a Roof and RoofEdges is defined by some reference roof_edge_names.

A list of entities which refer by name are described as follows:

• CeilingEdge refer to ROOFEDEGE
(roof_edge_name, roof_edge_name, etc.)

5.3.2.2 From technical drawing to GEO

The complete informal specification is listed in Annexe B. Here we give some examples.

Some entities may only be converted if a certain condition is true.

**TD_Arc _ GEO_Hole**

CONDITION:

\[
\text{start_point.x} = \text{end_point.x} \text{ AND start_point.y} = \text{start_point.y}
\]

(Convert only if the TD_Arc represents a full circle, i.e. if the previous condition is satisfied.)

The example above is a local condition, but it is also possible to describe global conditions (with possibly some parameters) which holds for the entire model.

**CONDITION :**

PARAMETER : (main, left, right, top, bottom, back)

The conversion has one global parameter specifying the desired view (main, left, right, top, bottom, back), since a technical drawing can have many views, but a GEO file can contain only one. This means that only contours, polylines, curves, points, etc. that belong to the specified view should be converted.
If many geometry elements (points, arcs) are mapped to one (GEO_Window) entity, an N to 1 conversion must be specified.

TD_Point[1:?], TD_Arc[1:?] _ GEO_Window

(All instances of TD_Point and TD_Arc should be scanned to produce one instance of GEO_Window.)

5.3.2.3 From VtM to BPtM / LSE PtM

The BPtM / LSE PtM is a model of a higher semantical level than the VtM and AtM. In contrast with the VtM and AtM, both BPtM and LSE PtM have a clear explanation of the models.

On account of the many SUB- and SUPERTYPEs in the models involved, the notation used for the specification is different compared to the other conversions. This is the reason why a part of the informal specification consists of a variation of the EXPRESS language, where some abbreviations are used, such as SUB OF (for SUBTYPE OF) and SUPER OF (1(...) (for SUPERTYPE OF (ONEOF(...))). The following table gives a overview of the used notation:

<table>
<thead>
<tr>
<th>ENTITY Source attribute number (.1 or .2)</th>
<th>ENTITY Target attribute number (e.g. A1 or B2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTITY Auxiliary attribute number (.1 or .2)</td>
<td></td>
</tr>
</tbody>
</table>

Informal specification of conversion examples:

Source attribute _ Target attribute

Explanation: entities from the source model are placed on the left hand side; entities from the target model are placed on the right hand side. In the source model we can also have auxiliary entities, i.e. entities which are not really mapped but are merely used to obtain some extra information. There can be no auxiliary entities in the target model. At the bottom of the table, the relation between the mapped entities and attributes is given by using arrows. Attributes are numbered (e.g. Source.1, Target.B2) for ease (shortness) of reference.

The complete informal specification is again listed in Annexe B. Here some examples are given.

It is possible that a list of entities in the source model is inversely defined in the target model. In the specification we express this by giving a condition fixing the inverse references. Some notation from predicate logic and set theory is borrowed.
### Entity Storey
- **storey_ID:** INTEGER

### Entity Section
- **storey_ID:** INTEGER

### Entity BuildingStoreySpace
- **storey_level:** REAL

### Entity BuildingSectionSpace
- **storeys:** SET OF BuildingStoreySpace

### Entity RectangularWallOpening
- **in_wall_segments:** SET OF WallSegment

### Entity RectangularOpening
- **opening_length:** REAL
- **opening_width:** REAL

### Entity Wall
- **wall_ID:** INTEGER
- **wall_type:** ENUM OF (enclosing, dividing)

### Entity RectangularOpening
- **length:** REAL
- **width:** REAL

### Entity OuterWindow
- **length:** REAL

### Entity InnerWindow
- **length:** REAL

### Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ / 60

Depending on the attributes of the source model, entities are created and values of attributes are mapped to the target model BPTM / LSE.

For the conversion only one start_point of a WallSegment of the VtM is essential, because this attribute must be converted to the attribute location of the Outer- or InnerWall in the BPTM / LSE PtM. In the VtM a Wall consists of many WallSegments with their own start- and end point. The start_point of the WallSegment for which the following condition holds: Wall.1[A.1 _ B.2], must be mapped to the attribute location (Wall.Segment.1 _ E1). This means: A Wall has only one WallSegment with the condition: the start- and end point of this WallSegment are not equal to the other WallSegments.
### 5.3.3 Formal specifications

#### 5.3.3.1 Introduction

Next to the informal specification discussed in the previous section, the formal specification of conversions has been studied. In this section several formal specification techniques will be surveyed. One of these techniques will be adopted to give a formal specification for the conversion benchmark...
problem as defined in paragraph 2 and informally specified in the previous section.

5.3.3.2 Overview of formal specification techniques

VDM and Z

On a conceptual level, we can distinguish two classes of specification languages [SPI88]:

1. Model-oriented languages, where the aim of a specification is to construct an abstract model of the information system being specified. Examples of such languages are VDM and Z.
2. Property-oriented (or algebraic) languages, where the aim of a specification is to describe a system in terms of its desired properties, without constructing an explicit model. Examples of such languages are Clear, OBJ and ACT ONE.

Because of the nature of the conversion problem, in our opinion the first class of languages are more suitable for the formal specification of conversions. VDM (Vienna Development Methodology) and Z are the two most well known specification languages belonging to this class [BHL90, SPI88, SPI92]. VDM and Z have a lot in common, such as that they both define a state space together with the operations on this state space. A feature supported by VDM but not by Z are: pre- and post-conditions are separated explicitly, so these can be viewed as giving proof obligations for the author of a specification. On the other hand VDM does not offer the possibility to combine schema's using a schema calculus, but Z does. As both languages are very similar, we will only discuss one of them, Z. We only give a very brief overview by showing a small example, for a more detailed description refer to [SPI92].

Example:

```
  __BirthdayBook___________________________________________________
  _ known : P NAME
  _ birthday : NAME --> DATE
  ______________________
  _ known = dom birthday
  ______________________________________________________________________

  __AddBirthday____________________________________________________
  __BirthdayBook
  _ name? : NAME
  _ date? : DATE
```

Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ / 62
NAME and DATE are declared to be basic types for this specification.

The schema BirthdayBook defines the state space of the system. The state space consists of two variables: known, which is a set of names (P stands for the powerset), and birthday, which is a partial function from names to dates. In addition to this, an invariant is specified, stating that the value of known is equal to the domain of the function birthday.

The schema AddBirthday specifies the effect of an operation to add a birthday. It operates on the state space BirthdayBook. The symbol _ denotes that BirthdayBook may be modified. There are two parameters, name and date. The symbol ? denotes that name and date are input parameters. The next line gives a pre-condition for the operation, stating that the name to be added is not already known. The post-condition says that the birthday function is extended to map the new name to the given date. The value of birthday after the operation is denoted by birthday'.

The schema FindBirthday, finally, shows how to find the birthday of a given person. The symbol _ denotes that in this case BirthdayBook may be used but not modified. Again the name is an input parameter, and now we also have an output parameter (denoted by the !) date. The rest is self-explaining.

In addition to this, it is also possible to combine schemas using the Z schema calculus, and to write generic schemas (using some type as a parameter). This is not dealt with in this short introduction to Z.

Recent developments show object-oriented extensions of Z, such as MooZ, Object-Z, OOZE, Z++, ZEST and a combination of Z and HOOD [SBC92]. These are mainly designed to support very large specifications, but is still worth investigating because EXPRESS models have an object-oriented nature.
because of the use of sub- and supertype relations.

**ExSpect**

ExSpect stands for executable specification. It is developed at the Eindhoven University of Technology [VB90]. It is designed to write executable specifications for distributed information systems. ExSpect consists of a model, a language and a toolbox. These are used to describe processes and their interaction (by using a variant of coloured Petri nets, together with a functional languages which is a sugared typed lambda calculus), and the structure of objects (by using a type system).

**RAISE**

The RAISE specification language (or RSL, (RAISE, 1992) has been developed in two ESPRIT projects: ESPRIT 315 RAISE (Rigorous Approach to Industrial Software Engineering) and its successor ESPRIT 5383 LaCoS (Large scale Correct Systems using formal methods). The development os RAISE has been influenced by languages such as VDM, ML (a functional language), COLD-K (which features dynamic object creation), Larch, and object-oriented developments. RAISE consists of a formal specification language with a semantics and a proof system. It features:

- parameterizable abstract data types
- modularity
- concurrency
- non-determinism
- subtypes.

Furthermore, the following specification styles are supported:

- applicative vs. imperative
- sequential vs. concurrent
- direct (explicit) vs. axiomatic (implicit)
- abstract data types (algebraic) vs. concrete data types (model oriented).

Especially the presence of an imperative specification style is remarkable, when comparing RAISE with other specification languages. The reasons for introducing this style are threefold: first, many formal specifications are eventually implemented in an imperative language; second, it reduces the number of parameters of functions; and third, some problems appear to have an imperative nature. The use of global
variables (databases), sequencing and imperative specifications might be of great importance for writing formal specifications for conversion problems.

Predicate Logic

First order predicate logic is not itself a language for formal specifications for software development. Its notation, however, is well known and its semantics are also well defined. Predicate logic is a standard tool for all mathematicians and (computer) scientists. Indeed, it forms the basis for most formal software specification languages. For an extensive treatment of predicate logic refer to the standard literature (e.g. [HAM88]).

5.3.3.3 Formal specification of the conversion benchmark

In this subsection we discuss the formal specification of a conversion example: the benchmark as defined in section 2.2. The complete formal specification is listed in Annex C. In this section some explanation is given.

First we have to choose a notation from the alternatives discussed in the previous subsection. The choice has been made to use first order predicate logic. The reasons for this choice are:

- **Z** has not been chosen because, compared to predicate logic, it is less well known and has little advantages for specifying conversion problems. For instance, standard Z lacks object-oriented notation to properly deal with EXPRESS schemas and instances (such as a STEP physical file structure). Furthermore, a conversion can be specified by one post-condition, so there is no need to introduce several Z operation schemas where the result of the operation is specified in terms of input and output variables, etc.

- **ExSpect** has not been chosen for a number of reasons. First, a conversion program is not a distributed system. Furthermore, the fact that a specification is executable is currently not the most important issue. So only the functional language may be of any use. But then we might as well use any other language with the same expressive power.

- **RAISE** has not been chosen because it is relatively unknown, also to the authors of this deliverable. It looks, however, very promising and further investigation is recommended.

- **Predicate logic** is a very well known language, with a more or less standard notation and semantics. It is easy to read and to extend with domain specific notation. With predicate logic,
we can concentrate on the conceptual specification of the result of a conversion in terms of post-
conditions, without having to worry about implementation details.

Next we will discuss the formal specification of the chosen example. The formal specification of a
conversion can be seen as a post-condition on an instance of a source schema and an instance of a
corresponding target schema. The condition states when a conversion can be called successful. This
means that when a conversion has been applied to the source model, thus creating a target model, the post
condition should evaluate to true.

The notation used is a mixture of several notations. First, common first order predicate logic (like the
predicate logic used by Z) is used for expressing the logical conditions. Second, a notation similar to
EXPRESS is used in order to be able to directly refer to the EXPRESS schemas involved. This saves a
translation from EXPRESS notation to common set theory notation for handling data types.

Next to using source and target EXPRESS schemas we also need a way to denote source and target
schema instances. This is done as follows. First, both schemas are extended with an implicit entity which
is a supertype of all other entities, i.e. for the technical drawing model:

    ENTITY TD_Entity
    SUPERTYPE OF (        
        TD_TechnicalDrawing, TD_View, TD_CrossSection, TD_Detail,
        TD_MainView, TD_LeftView, TD_RightView, TD_TopView,
        TD_BottomView, TD_BackView, TD_DrawingPart, TD_Text, TD_Point,
        TD_Arc, TD_Line, TD_Curve, TD_PolyLine, TD_Contour        );
    END_ENTITY;

And analogously for GEO_Entity. Second, two global variables are introduced to represent the source
and target schema instances:

    source : LIST OF TD_Entity;
    target : LIST OF GEO_Entity;

The index in the list can be compared to the instance numbers in a STEP physical file (part 21). Third, an
auxiliary variable map is introduced to be able to express conditions on which instances should be
mapped to which instances, and how instances of both models relate to each other.

    ENTITY MAP_Entity;
    source_instance : TD_Entity;
    target_instance : GEO_Entity;
    END_ENTITY;

    map : SET OF MAP_Entity;
Both the notation \((s, t) \in \text{map}\) and the notation \(\text{map}(s, t)\) (where \(s : \text{TD}\_\text{Entity}\) and \(t : \text{GEO}\_\text{Entity}\)) are used although they are logically equivalent. The former is used in the case that there is a condition stating that some source instance \(s\) (somewhere in the list source) should have been mapped to target instance \(t\) (somewhere in the list target). The latter is used to express the fact that some instance is mapped to another instance. \text{map} is only used when necessary, i.e. when it is essential to know that some instance in the source model is mapped onto some other instance in the target model.

Fourth, the entity names in both models are overloaded (i.e. used for more than one purpose) to denote the set of instances of this entity in the source (or target) schema instance, i.e. occurring in the list source (or target). For example,

\[
\text{TD}\_\text{Point} : \text{BAG OF TD}\_\text{Entity};
\]

is implicitly defined to contain the bag of all instances of \text{TD}\_Points in the list source. So the list of all instances in the source model is partitioned into a number of bags containing all instances of a certain entity. Now we can write \(p : \text{TD}\_\text{Point}\) to state that \(p\) is an instance of a point in the source model.

The formal specification can now be given as a set of predicates which all have to hold in order for the conversion to be valid. Before we give the specification, some notation is explained.

- \(\_ \ e : S \cdot P(e)\) means that for all (or there exists, \(\_\)) elements \(e\) of some set \(S\), predicate \(P\) (on \(e\)) holds. Other used symbols are connectives like \(\_\) (implies) and \(\_\) (logical and).
- For instances \(i_1\) and \(i_2\), \(i_1 = i_2\) means that \(i_1\) and \(i_2\) denote the same (reference to an) instance, not that \(i_1\) and \(i_2\) have the same value. (See also the recent discussion on instance equality for the EXPRESS definition.)
- \(\text{SET OF}\) (and \(\text{BAG OF}\), \(\text{LIST OF}\)) is also used as a set constructor, i.e.

\[
\text{SET OF } n : \text{INTEGER} \cdot \ldots
\]

denotes the set of all integers satisfying some condition.
- \(\text{MAX}\) (and \(\text{MIN}\)) is used as a quantifier, i.e.

\[
\text{MAX } e : S \cdot e
\]

denotes the maximum value of the set \(S\).
- max (and min) is also used as an infix operator, i.e. \(a \text{ max } b\) denotes the maximum value of a
The specification should be read as giving both the necessary and sufficient condition. For example when it is stated that

\[ p_1 : \text{TD\_Point} \quad p_2 : \text{GEO\_Point} \quad \ldots \]

this means that for all points in the source model there is a point in the target model, but not more than one.

The conversion example has one parameter (view) specifying which view is to be converted. This means that for all instances in the source model it should be possible to determine whether the instance "belongs" to the specified view or not. To this end, the function scope is introduced. For some instance \( i \), \( \text{scope}(i) \) denotes the set of all instances which can be reached from \( i \) by using attribute relations. Example:

\[
\begin{align*}
#1=\text{TD\_MainView}(#2,(.),(),()); \\
#2=\text{TD\_Contour}((#3,#4,#5)); \\
#3=\text{TD\_Line}(#6,#7); \\
#4=\text{TD\_Arc}(#7,#8,#6,1); \\
#5=\text{TD\_Line}(#8,#6); \\
#6=\text{TD\_Point}(1.0,1.0); \\
#7=\text{TD\_Point}(2.0,1.0); \\
#8=\text{TD\_Point}(1.0,2.0); \\
#9=\text{TD\_Point}(7.0,8.0);
\end{align*}
\]

then \( \text{scope}(#2) = \{ #2, #3, #4, #5, #6, #7, #8 \} \). So point number 7 belongs to contour number 2.

Not all auxiliary functions are extensively specified (this is not difficult but a tedious task). Refer to the explanation of the conversion benchmark and to the implementation in C++.

As we have seen in the informal specification for this conversion problem, the conversion has one global parameter denoting the desired view, i.e. which view in the technical drawing model should be converted to the GEO model.

\[
\text{view} : \text{TD\_Entity}
\]

Next we specify the existence and attributes of the GEO\_File instance. There is exactly one such object. The attribute area should point to the GEO\_Window instance. The attribute geometry should be the set containing (references to) all instances of geometry elements. Remember that the set of all these instances is denoted by GEO\_GeometryElement (see above). The value of the attributes contours and point\_groups is determined analogously.
For all contours belonging to the desired view in the technical drawing model, there should be a corresponding contour in the GEO model. Checking whether the contour belongs to the desired view is done by checking whether the contour is in the scope of the view. The contour of the technical drawing has an attribute elements which is a list containing curves. Each curve (either a line or an arc) is mapped to a contour element in the GEO model. This is stated in the predicate specifying the existence and attributes of contour elements (line elements or circle elements) in the GEO model (see further). The fact that a certain curve is mapped to a certain contour element is kept track of by the set map. So the contour elements of a GEO contour can be determined by taking all contour elements which are mapped from all curves of the technical drawing contour, \textit{in the same order}. The specification of the values of the remaining attributes should now be clear.

\begin{verbatim}
_ c1 : TD_Contour • c1.scope(view) _
  _ c2 : GEO_Contour •
    _ i : INTEGER • 1 ≤ i ≤ SIZEOF c1.elements _
      (c1.elements(i), c2.elements(i)) _ map _
      ( _ p : GEO_Point •
        (c1.elements(1).start_point, p) _ map _
        c2.start_point = p.point_number ) _
      ( _ p : GEO_Point •
        (c1.elements(SIZEOF c1.elements).end_point, p) _ map _
        c2.end_point = p.point_number )
\end{verbatim}

For all lines belonging to the desired view of the technical drawing, there should be a corresponding line and line element (of a contour) in the GEO model. The line element refers to the line by its line number. The only tricky thing in this part of the specification is that for each line element the direction relative to the previous contour element of the contour to which the line belongs should be specified. The question now is: what is the previous element? This can be determined by first looking for the contour to which the line belongs, and then check its list of contour elements to find the previous element (if any). The fact that the line of the technical drawing is mapped to the GEO line element should be kept track of in the set map.

There is no calculation or function call to determine a unique line number. It is simply stated that all line numbers in the GEO model should be unique, i.e. if two lines have the same line number then they represent the same line.

Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ / 69
In the GEO model, there should be one window object giving the bounding box (in two dimensional space) for the desired view. This is done by taking into account all points and arcs (with their radius, which is calculated from the centre and the starting point) in the desired view. Only the value of the x coordinate is specified. The values of the y-, i- and j-coordinates can be specified analogously.

Finally, a specification should be given for a number of auxiliary functions. As an example, we give the specification of the function distance.

```plaintext
_distance(p1, p2) = sqrt((p1.x - p2.x)^2 + (p1.y - p2.y)^2)
```
5.3.4 Conclusions

Before implementing a conversion (in any language) it is good practice to first study the models involved and to specify the desired result. A requirement is of course the presence of sufficiently detailed and documented models, i.e. an EXPRESS schema should be available instead of more or less rough ATLIAM model, and the structure of the models, as well as the meaning of all entities and attributes should be explained in an accompanying document. A specification can either be formal or informal. Both formal and informal specifications concentrate on WHAT the result of a conversion should be, not on HOW to accomplish this result. A specification can serve as the basis for implementation in a programming language.

The advantage of an informal specification is that one does not have to worry about less relevant details, so it is not difficult to write such a specification. A disadvantage is that it is not clear how to write an informal specification because a framework such as a notation and a semantics is missing. Also, it is not easy to write an informal specification which is complete and not ambiguous, although there are some rules of thump, e.g. check if all entity and attribute names of both source and target model occur in the specification. The informal specification should be either brief or exhaustive; not somewhere in between. By brief we mean that the specification gives a global overview of the conversion and some points of attention. A brief specification suffices if the models are well documented and the programmer using the specification also has access to this documentation. If the latter is not the case, then it is better to make an exhaustive specification which can be implemented by a programmer without further knowledge of the models. As far as the notation is concerned, in order to obtain a clearly structured specification, it is advised not to make the specification too informal and to make use of tables, good layout, etc. instead of mere prose. For practical reasons it is best to place target entities and attributes on the left hand side, and source entities and attributes on the right hand side (contrary to the given examples). Example:

GEO_Point → TD_Point

• reference_point 0
• x_coordinate x
• y_coordinate y
• point_number "new point number"

instead of

TD_Point → GEO_Point

• reference_point := 0

Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ / 71
This is better and clearer from a layout point of view, and corresponds closer to other notations used (e.g. programming languages but also EXPRESS-M).

A formal specification has the following advantages. First, it is always written using a well defined language and it is unambiguous. Second, because of the formal semantics, a formal specification may be supported by using special software tools, which may lead automatically to an implementation. Because of the advantages mentioned before, a formal specification can serve as the basis for the implementation of a conversion, either in a conventional language or in a logical programming language. Disadvantages are that making a complete formal specification can be a very laborious task (although this can save a lot of time during implementation), and that because of the nature of a specific conversion problem it is sometimes easier to give a procedural solution (in terms of a programming language) than a declarative solution (in terms of predicate logic). Although predicate logic (extended with EXPRESS notation) proved to be suited for writing formal specifications, future research of other specification languages (such as RAISE or object-oriented extensions of Z) may be profitable because they may be supported by software which may lead to automatic implementation.

The necessity to make use of the set map gives an indication that the possibility to keep track of which instances in the source model are mapped to which instances in the target model is a requirement for a suited conversion language (and implementation).
5.4 Implementation in C++ and Prolog

5.4.1 Introduction

One of the possibilities to implement conversion is using an existing programming language. We have chosen to use two alternative programming languages: C++ and Prolog. C++ is used because it is an object-oriented language which can be used together with a number of STEP tools based on this language, such as a tool to translate EXPRESS schemas to C++ class definitions, together with a standard library to read and write STEP physical files. Prolog is used as an alternative because it is a declarative language, suited for representing expert knowledge about conversions in a more natural way, also corresponding closely to the formal specification using predicate logic. In this paragraph a description is given of the implementation of some conversions in C++ and Prolog. The implementation in C++ and in Prolog are given in 4.2 and 4.3, respectively, followed by some conclusions in 4.4.

5.4.2 Examples in C++

5.4.2.1 Conversion of VtM to AtM

Within the scope of ATLAS we worked out the conversion between the view type model for the architect in the Global Design stage (VtM) and the attribute type model (AtM) of Speedikon. The C++ source code of this conversion can be found in Annex D.

The implementation of this conversion started by working out the conversion in an informal specification as described in section 3.2. The informal specification describes how the source entities and attributes must be projected on the target entities and attributes. With the informal specification and the EXPRESS-code of both models we started the implementation of the conversion. We used PMshell (=Product, Project, Production or Process Modelling shell), a layered-modelling tool for the integration of computer applications (TNO, 1992). PMshell offers the possibility to parse an EXPRESS-schema and extend it by connecting C++-routines on the entities (classes). Next PMshell translates the EXPRESS-schema to C++-classes and joins the classes with the user-defined routines and some "standard" routines of PMshell. (The "standard"-routines of PMshell are e.g. routines to parse and write STEP Physical files, creating
objects and user-interface routines.) The result is a complete set of C++ code to generate (by compiling and linking) a program which works on the parsed data model. In case of the conversion between VtM and AtM, the EXPRESS-schemas of both models are parsed and routines added to convert the instances from VtM to AtM.

The implementation of this conversion was not so difficult because Speedikon uses references by name to define relations between entities. The advantage of references by name is that during the conversion you can create a string (using the ID of the entity to keep the references unique) with the name of the referenced entity without concerning if the referenced entity is already converted. Another advantage is that the VtM has one instance of "Building" and a relation with all the other entities, therefore it was possible to define the main "convert" at this entity. The routine for this conversion is printed below. It first converts the "Building" to a "Project", followed by the conversion of all the references to the entities in the model. Not all entities are directly connected with "Building" but by some hierarchical relations. For example a building has a relation with a roof, a roof has a relation with a roof surface and a roof surface has a number of relations with roof edges. For this the conversion starts with calling the routine to convert the roof in the main routine and the routine to convert the roof calls the routine for conversion of the RoofEdges. (The RoofSurface is not converted because there is no corresponding entity in the target model.)

```cpp
virtual void Building::map_to_project ()
{
    // require

    Project *project;

    project = new Project;

    project->set_project_number(int2string(building_ID));
    if (building_name != NULL)
    {
        project->set_project_name(building_name); 
    }
    else
    {  
        project->set_project_name("no project name");
    }

    project->set_proprietor1(architect);
    project->set_installation_date(version_date);
    project->add_text(building_description);
    project->set_supervisor(supervisor);
    project->set_project_reference(project_title);

    for ( grids->start(); !grids->finish(); grids->forth())
    { 
        grids->cursor()->map_to_griddefinition();
    }
    for ( storeys->start(); !storeys->finish(); storeys->forth())
    { 
        storeys->cursor()->map_to_floor();
    }
    for ( rooms->start(); !rooms->finish(); rooms->forth())
    { 
        rooms->cursor()->map_to_room();
    }
```

Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ / 74
for (walls->start(); !walls->finish(); walls->forth())
{
    walls->cursor()->map_to_wall_definition();
}
if (roof)
{
    roof->map_to_roof();
}
else
{
    printf("Error: No roof defined.\n");
}
for (slabs->start(); !slabs->finish(); slabs->forth())
{
    slabs->cursor()->map_to_roof();
}
for (ceilings->start(); !ceilings->finish(); ceilings->forth())
{
    ceilings->cursor()->map_to_roof();
}
for (columns->start(); !columns->finish(); columns->forth())
{
    columns->cursor()->map_to_columnsquare();
}
for (beams->start(); !beams->finish(); beams->forth())
{
    beams->cursor()->map_to_beamsquare();
}
for (openings->start(); !openings->finish(); openings->forth())
{
    openings->cursor()->map_to_opening();
}

project->set_name("PROJECT"); /* To visualise the entity */
working_model->add_object(project); /* in a table. */

The structure of the routines is first the creation of instances of the target entities and setting the values of the attributes by the routines "set_..." (single values) or "add_..." (by a set, list, bag or array). In some cases the value of an attribute is examined and depending on this value the value of the target attribute is set. A "Wall" and a "WallSegment" of the VtM are joined together to a "WALL_DEFINITION" in the AtM. Because a "Wall" can have one or more "WallSegments", it is necessary to copy some attributes of the "Wall" for every "WallSegment" to another "WALL_DEFINITION" with the values of one "WallSegment".

As well as the routines for conversion there are also routines defined to determine some values. One example is a routine to find the section with the lowest number (smallest_section_ID()).

virtual int Wall::lowest_storey (LinkList <Storey *> *storeys)
{
    // require

    Storey *lowest;

    lowest = storeys->access(1);

    for (storeys->start();
        !storeys->finish();
        storeys->forth())
    {
        if (storeys->cursor()->storey_level < lowest->storey_level)
        {
            lowest = storeys->cursor();
        }
    }

Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ / 75
5.4.2.2 Conversion of the benchmark

Another example of a conversion which can also be found in the Annexe, is the conversion of a technical drawing to the GEO format. A description of the conversion can be found in section 2.2. Just like the conversion described before the conversion starts in the top of the hierarchy (the TechnicalDrawing) in which the conversion of its attributes referencing other entities is carried out. In the conversions of these entities the conversions of its attributes are carried out and so on. This example shows also the use of global variables. The global variables x_min, y_min, x_max and y_max for example are used to determine the minimum and maximum coordinates of the contour in the TechnicalDrawing. At the end of the conversion a window is created and these global variables are used for this window.

```cpp
void TD_TechnicalDrawing::convert(String view) {
    // initialize global variables
    x_min = y_min = MAXINT;
    x_max = y_max = -MAXINT;
    point_count = line_count = circle_count = 1;

    // create GEO object
    geo = new GEO;

    // create GEO_File and connect to geo
    geo_file = new GEO_File;
    geo->add_object(geo_file);

    // convert desired view
    shape->start();
    while (!shape->finish()) {
        if (shape->cursor()->class_name == view) {
            shape->cursor()->convert();
        }
        shape->forth();
    }

    // make window
    GEO_Window *window = new GEO_Window;
    window->x_coordinate = x_min;
    window->y_coordinate = y_min;
    window->i_coordinate = x_max;
    window->j_coordinate = y_max;
}
```
The two schemas used in this example contain very different data because the semantic meaning of the models is different. To convert the data between the models a number of transformations are necessary, like simple additions and complex routines to find the direction of a circular arc.

5.4.3 Examples in Prolog

The conversion benchmark has been implemented in Prolog. Prolog will not be explained here; refer to for instance [CM87]. Because Prolog is a logical programming language and the formal specification of this conversion has been worked out in predicate logic, the formal specification will be used as a starting point for the implementation in Prolog. In this section, the implementation of conversions in Prolog will be explained, by using the example. A complete listing of the implementation in Prolog can be found in Annexe E.

The first thing to do when implementing a conversion in Prolog is to provide some translation from EXPRESS schemas and instances (e.g. a STEP physical file) to Prolog notation. There is no need to translate EXPRESS entities and types to Prolog because to Prolog, since Prolog is not a typed language. The translation of instances is explained using part of a STEP physical file.

STEP Physical file:

```plaintext
#1=TD_MainView(#2,(),(),());
#2=TD_Contour((#3,#4,#5));
#3=TD_Line(#6,#7);
#4=TD_Arc(#7,#8,#6,1);
#5=TD_Line(#8,#6);
#6=TD_Point(1.0,1.0);
#7=TD_Point(2.0,1.0);
#8=TD_Point(1.0,2.0);
#9=TD_Point(7.0,8.0);
```

Prolog:

```plaintext
td_MainView(1, 2,[,][,][]).
td_Contour(2, [3,4,5]).
td_Line(3, 6,7).
td_Arc(4, 7,8,6,1).
td_Line(5, 8,6).
td_Point(6, 1.0,1.0).
td_Point(7, 2.0,1.0).
```

Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ / 77
td_Point(8, 1.0, 2.0).
td_Point(9, 7.0, 8.0).

So every instance in the STEP file results in an entry (or clause or predicate) in the Prolog database. The name of the entity is used, where 'TD' was changed to 'td' because in Prolog predicate names start with a lower case letter, and variables with an upper case letter. The attribute values result in arguments in the Prolog clause. The instance number of a STEP file instance (e.g. '#1=') is added as the first argument of a clause. These numbers are also used to refer to other instances, as can be seen in the example. The instances of the source model are assumed to be present in the Prolog database (which can be accomplished by the Prolog consult function). The conversion program should result in the creation of the instances of the target model in the Prolog databases. This is done by using the Prolog function assertz, which adds some clause (at the end, hence the z) to the database.

Special attention should be paid to sub- and supertype relations. For instance, in the model for technical drawings we have

ENTITY TD_Arc
  SUBTYPE OF (TD_Curve);
  orientation : TD_ORIENTATION;
  centre : TD_Point;
END_ENTITY; -- TD_Arc

ENTITY TD_Line
  SUBTYPE OF (TD_Curve);
END_ENTITY; -- TD_Line

ENTITY TD_Curve
  ABSTRACT SUPERTYPE OF (ONEOF (TD_Arc, TD_Line));
  end_point : TD_Point;
  start_point : TD_Point;
END_ENTITY; -- TD_Curve

Curve is an abstract supertype of line and arc, so only instances of lines and arcs occur in a model instance, never instances of curves. But sometimes in our program we do want to refer to all curves (meaning all lines and arcs together). The problem is now that Prolog has no object-oriented facilities, so there is no way of telling that a line is also a curve. This problem is solved by explicitly telling Prolog about this in the following way:

td_Curve(N, Start_Point, End_Point) :-
  td_Line(N, Start_Point, End_Point) ;
  td_Arc(N, Start_Point, End_Point, __).

So now we can ask for all curve instances. This is used for instance to determine the starting point of a
contour, which is the starting point of the first curve (which can be either a line or an arc). Note that no curve instances occur in the Prolog database!

The set map : set of MAP_Entity is represented in Prolog by a number of entries in the database, i.e. for every (s,t) _ map, there is an entry map(s,t) in the database. Of course the value of map is only relevant during the mapping process, so it is temporary. The correspondence to the two different uses of map in the formal specification are:

<table>
<thead>
<tr>
<th>Formal specification</th>
<th>Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s,t) _ map</td>
<td>map(s,t)</td>
</tr>
<tr>
<td>map(s,t)</td>
<td>assertz(map(s,t))</td>
</tr>
</tbody>
</table>

The desired view to be converted is indicated by a clause view(n) where n is the instance number of some view entity, for instance td_MainView(n,...). The function scope is implemented in a straightforward way and should be self explaining. Example:

\[
\text{scope}(2,S). \text{ results in } S = \{2,3,4,5,6,7,8\}. \text{ in the example above.}
\]

Before we start explaining the Prolog implementation, a few more remarks are in place.

- The generation of unique numbers for instances, lines, circles and points is implemented in a straightforward way, where the highest available number is kept in the database. All numbers are initialized to one.
- A number of auxiliary functions is used. In order to be able to understand the given examples, some comments are given here.
  - match(L1,L2) means that lists L1 and L2 match, i.e. if X is the n-th member of L1 and Y is the n-th member of L2, then map(X,Y) holds. Typically, L1 serves as an input argument and L2 serves as an output argument. match is used to determine the contour elements belonging to a GEO contour.
  - nth(N,L,X) means that X is the Nth member of the list L.
  - index(X,L,N) means that N is the index of member X of the list L.

All other functions are standard functions or should be self explaining.

- The functions distance, direction and angle are not implemented for the same reason why they were not formally specified. The implementation makes use of dummy implementations.

Probably the biggest difference between the formal specification in predicate logic and the implementation in Prolog is that the implementation in Prolog is after all a computer program, i.e. it
should be executable by a computer. This implies that the conversion, which ultimately consists of a large number of small steps, should be carried out in some order. In predicate logic the order is not important; the end result is the only thing that matters. In the Prolog program we are forced to specify some order in which to evaluate some predicates, even if the order is not significant. For instance, line and circle elements in the GEO model should be created before contours, because contours refer to existing line and circle elements. It does not matter, however, whether point groups are created before or after contours. The order of creating target instances is "bottom up", i.e. the lowest elements in the model hierarchy are created first, then the ones above them, and so on. The results in the main predicate of the program map_td_to_geo:

```prolog
map_td_to_geo :-
    findall(__map_Point_to_Point       _,_),
    findall(__map_Line_toLine_LineElement _,_),
    findall(__map_Arc_toCircle_CircleElement_,_),
    findall(__map_Contour_toContour     _,_),
    findall(__map_Arc_toPointGroup_Hole _,_),
    map_to_Window,
    map_to_File.
```

This means that first all technical drawing points are mapped to GEO point, second all technical drawing points are mapped to GEO lines and GEO line elements, and so on. There is only one GEO Window instance which should be mapped before the only GEO File instance.

The most important issue is to "translate" parts of the formal specification to Prolog clauses. We will start with the GEO file.

```prolog
map_to_File :-
    geo_Window(W_,__,__,_),
    findall(GE,geo_GeometryElement(GE,__,_),SGE),
    findall(C,geo_Contour(C,__,__)         ,SC ),
    findall(PG,geo_PointGroup(PG,__)       ,SPG),
    new_instance_number(N),
    assertz(geo_File(N,W,SGE,SC,SPG)).
```

Looking at the formal specification and the explanation given above this should be self evident by now. Let's look at the mapping or conversion of contours now. First the existence of a technical drawing contour is checked, resulting in the instance number N and the set of elements (curves) SE. Then the condition that the contour is in the scope of the desired view is checked. Before a new instance of a GEO contour can be created, the value of its attributes must be determined. The first attribute is the list of contour elements, denoted by the variable SCE. This is done by matching the technical drawing contour elements (SE) to the GEO contour elements (SCE), i.e. a reference is found to all GEO contour elements which are mapped from all technical drawing contour elements for this particular contour. The second
attribute is the starting point of the GEO contour, denoted by the variable SP. To determine its value, we take the first curve of the technical drawing contour, and find out to which GEO point its starting point is mapped. Then we take SP to be the point number of this GEO point. The value of the third attribute is determined analogously. Finally the instance of the GEO contour is created with a new instance number and all appropriate attribute values.

map_Contour_to_Contour :-
   td_Contour(N,SE),
   view(V), scope(V,R), member(N,R),
   match(SE,SCE),
   nth(1,SE,First),
   td_Curve(First,Start_point,_),
   map(Start_point,N1),
   geo_Point(N1,_,_,_,SP),
   length(SE,L),
   nth(L,SE,Last),
   td_Curve(Last,_,End_point),
   map(End_point,N2),
   geo_Point(N2,_,_,_,EP),
   new_instance_number(M),
   assertz(geo_Contour(M,SP,EP,SCE)).

The next example is the conversion of technical drawing lines to GEO lines and GEO line elements. The most complex task is to determine the value of the direction attribute, giving the direction of the line element relative to the previous contour element. Referring to the formal specification and the explanation given above, this should now be clear. Note that a map from the technical drawing line to the GEO line element is added to the database.

map_Line_to_Line_LineElement :-
   td_Line(N,Start_point,End_point),
   view(V), scope(V,R), member(N,R),
   td_Point(Start_point,X,Y),
   angle(Start_point,X,Y),
   td_Curve(C,SE),
   td_Contour(C,SE),
   index(N,SE,I),
   (I > 1, J is I - 1, nth(J,SE,Previous), direction(Previous,N,D) ;
    I = 1, D = no_direction),
   new_line_number(K),
   new_instance_number(M1),
   new_instance_number(M2),
   assertz(geo_Line(M1,0,X,Y,A,K)),
   assertz(geo_LineElement(M2,D,K)),
   assertz(map(N,M2)).
The implementation of map_to_Window is straightforward. Note that it is not possible to take into account the instances of arcs, because their radius cannot be calculated (the function distance is not implemented).

```prolog
map_to_Window :-
% arc instances are not processed (radius cannot be calculated)
    findall(X,(td_Point(N,X,_),view(V),scope(V,R),member(N,R)),S1),
    findall(Y,(td_Point(N,_,Y),view(V),scope(V,R),member(N,R)),S2),
    min(X,S1),
    min(Y,S2),
    max(I,S1),
    max(J,S2),
    new_instance_number(M),
    assertz(geo_Window(M,X,Y,I,J)).
```

### 5.4.4 Conclusions

Using an informal specification of a conversion proved to be a useful starting point for implementation in C++. The implementation of a conversion using C++ offers a lot of possibilities to realise a conversion, using the powerful statements and routines defined in the programming language. Some important advantages of programming languages are the possibility to store data in memory, to use local and global variables, powerful predefined routines (like sin, cos, abs, power) and the possibilities to search for instances. The most important disadvantage of C++ is that a programmer is forced to think in an operational way, where he has to consider all kinds of implementation details. Also, every programmer has the liberty to work out the conversion in his own way which does not contribute to a uniform solution of the conversion problem.

Using Prolog as an implementation language we can use a formal specification as a starting point. There is an almost one-to-one correspondence between the formal specification in predicate logic and the implementation in Prolog. It follows that Prolog is a language well suited for implementing conversions. Comparing Prolog to C++, in Prolog conversions can be implemented in a declarative way which naturally follows the formal specification, even when implementing the map relation between mapped instances. In Prolog the programmer can think on a more conceptual level without having to bother about implementation details. This also makes it easier to construct correct implementations. A disadvantage (and this is also the only main difference between formal specification and Prolog) is that in Prolog we are forced to state the order in which parts of the conversions have to be carried out. This disturbs the
declarative nature of the implementation, but this is a small price to pay for all the advantages.
5.5 EXPRESS-M

5.5.1 Introduction

EXPRESS-M is a schema conversion language, which means that it is used to describe how entity instances should be converted between schemas in order to facilitate the transfer of data between applications described by those schemes [BAI94a]. EXPRESS-M is developed to be a part of the International Standard for the computer-interpretable representation of product data (ISO 10303) also known as STEP. The objective of STEP is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product, independent from any particular system. The nature of this description makes it suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and archiving. The data definition language EXPRESS is defined by STEP to specify the aspects of product data in an unambiguous way. EXPRESS is used to define schemas (models) which describes data structures. EXPRESS-M is defined to convert instances of one schema to another schema. The syntax of EXPRESS-M is quite similar to EXPRESS.

EXPRESS-M is developed by CIMIO Ltd and Brunel University, who are also developing the EXPRESS-M software.

5.5.2 Possibilities and restrictions

EXPRESS-M offers the possibility to convert every instance of a source entity to zero, one or more instances in the target model. It is also possible to use constraints on entity conversion, e.g. depending on a value of an attribute a source-entity will be converted to one of the subtypes of a target-supertype. Most of the functionality of EXPRESS can also be used in EXPRESS-M like mathematical processing, most data types, REPEAT-, IF-, CASE-statements, user defined functions and standard functions (e.g. sin, cos, abs). DERIVED attributes can also be converted because the value of the attribute is calculated before the conversion will take place. In EXPRESS is it possible to define rules/conditions which are used to check the values of the instances which are stored in a model. These EXPRESS WHERE-rules are not converted.
EXPRESS-M also has some problems (restrictions) which possible will be solved in a future definition of EXPRESS-M.

- It is not possible to convert two or more not directly related source entities to one or more target entities.
- It is not possible to create relations between entities which are converted.
- EXPRESS-M does not allow to define more than one conversion for one entity.
- It is not possible to create instances in the target model which have no source entities.
- The declaration of GLOBAL variables is not (yet) possible in EXPRESS-M.
- There is no way to influence the order in which instances are mapped. For instance, to refer to the point_number of the start_point of a line (when mapping a contour), the point should be mapped first.
- It is not possible to pass parameters to a called mapping. Example of the conversion benchmark: when mapping a curve (line or arc) the previous curve should be known (passed as a parameter) because the direction relative to the previous curve must be calculated.
- It is not possible to map only instances belonging to some specific part of a model (e.g. a specified view), without extensive programming.

At the moment CIMIO are working on an implementation of EXPRESS-M. Doing so they also found a number of problems. The definition of EXPRESS-M will be improved in the future.

5.5.3 Examples

The conversion between the view type model for the architect in the Global Design stage (VtM) and the application type model (AtM) of Speedikon as defined in ATLAS and the conversion benchmark are worked out in EXPRESS-M and can be found in the Annexe. The examples in this section are parts of the conversions from the Annexe and give an idea of the notation of EXPRESS-M.

The first example gives shows the conversion of a Building to a PROJECT. The conversion is a 1:1 conversion and contains no difficult constructions. In the conversion two routines are used: the routines int2string to convert an integer into a string and the routine length to determine the length of a string.

MAP PROJECT <- Building;
ESP RIT 7280 - ATLAS - D302a: Data Abstraction, Generalisation and View Conversion

project_number := int2string(building_ID);
IF (LENGTH(building_name) > 0))
    project_name := building_name;
ELSE
    project_name := "no project name";
ENDIF;
proprietor1 := architect;
installation_date := version_date;
text[1] := building_description;
supervisor := supervisor;
project_reference := project_title;
END_MAP;

EXPRESS-M is able to determine if an attribute is a part of the source or target entity, so it is not necessary to use the name of the entities. For example the next two lines have the same meaning for EXPRESS-M.

PROJECT.proprietor1 := Building.architect;
proprietor1 := architect;

Sometimes it is necessary to use the entity name if the attributes names are equal and it is not clear which attribute is used. The second example is a more complex conversion which shows the resemblances between EXPRESS-M and a programming language. This conversion transforms a RectangularCeilingOpening into an opening and a number of roof edges. The conversion contains some statements and calculations to generate the correct data for the target entities.

MAP (OPENING, ROOF_EDGE[1:?]) <- RectangularCeilingOpening;
roof_opening_name := "CEILINGOPENING" + int2string(opening_ID);
depth := in_ceiling.ceiling_thickness;
roof_opening_type := 1;
section_number :=
    smallest_section_ID(in_ceiling.involved_sections);
IF EXISTS (in_ceiling.involved_storey) THEN
    floor_number := in_ceiling.involved_storey.storey_ID;
ELSE
    floor_number := 0;
ENDIF
REPEAT i = 1 TO 4;
    ROOF_EDGE[i].roof_edge_name := "CEILINGOPENING_EDGE" +
        int2string(opening_ID) +
        int2string(index);
    ROOF_EDGE[i].roof_number := roof_opening.floor_number;
    ROOF_EDGE[i].section_number := roof_opening.section_number;
    ROOF_EDGE[i].roof_edge_angle := 90;
    ROOF_EDGE[i].roof_edge_angle_type := 0;
CASE (index) OF
    case 1:
ROOF_EDGE[i].x_start := opening_position.x_coordinate;
ROOF_EDGE[i].y_start := opening_position.y_coordinate;
ROOF_EDGE[i].x_end := opening_position.x_coordinate + opening_length;
ROOF_EDGE[i].y_end := opening_position.y_coordinate;
case 2:
ROOF_EDGE[i].x_start := opening_position.x_coordinate + opening_length;
ROOF_EDGE[i].y_start := opening_position.y_coordinate;
ROOF_EDGE[i].x_end := opening_position.x_coordinate + opening_length;
ROOF_EDGE[i].y_end := opening_position.y_coordinate + opening_width;
case 3:
ROOF_EDGE[i].x_start := opening_position.x_coordinate + opening_width;
ROOF_EDGE[i].y_start := opening_position.y_coordinate + opening_width;
ROOF_EDGE[i].x_end := opening_position.x_coordinate;
ROOF_EDGE[i].y_end := opening_position.y_coordinate + opening_width;
case 4:
ROOF_EDGE[i].x_start := opening_position.x_coordinate;
ROOF_EDGE[i].y_start := opening_position.y_coordinate + opening_width;
ROOF_EDGE[i].x_end := opening_position.x_coordinate;
ROOF_EDGE[i].y_end := opening_position.y_coordinate;
END_CASE;

ROOF_EDGE[i].z_start := opening_position.z_coordinate + 0.5 * depth;
ROOF_EDGE[i].z_end := opening_position.z_coordinate + 0.5 * depth;

ROOF_OPENING.roof_edge_names[index] := ROOF_EDGE.roof_edge_name;
END_REPEAT;
END_MAP;

The last example shows a recursive mapping which is a part of the conversion benchmark. In this example defines a mapping of a contour which calls the mapping of lines and circles by type casting.

MAP GEO_Contour <- TD_Contour;
LOCAL
  i : INTEGER;
  previous_curve : GEO_Curve;
END_LOCAL;
previous_curve := NULL;
REPEAT i := 1 TO SIZEOF(elements);
  -- using a cast, another map is called;
  -- previous_curve should be passed as a parameter
  IF elements[i] IS TD_Line THEN
    elements[i] := (GEO_Line,GEO_LineElement)elements[i];
  ELSE
    elements[i] := (GEO_Circle,GEO_CircleElement)elements[i];
END_CASE;
5.5.4 Conclusions

The definition of EXPRESS-M is not definitive yet and CIMIO are still working on it, as well as on the implementation of the language. During our evaluation of EXPRESS-M we contacted CIMIO to ask for some explanation on unclear parts of the manual and restrictions of the language. It appeared that they were experiencing problems similar to ours, working out conversions for the PI-STEP project. This has led to changes in the language definition, resulting also in a new language manual [BAI94b]. Our examples and evaluations are based on a earlier version of the manual, but on studying the new manual most problems still seem unsolved.

Sometimes problems in EXPRESS-M (caused by lacking functionality) can be overcome by using EXPRESS-M functionalities like those of a programming language. For instance, it is impossible to link instances created in two separate MAPs, which can be solved by having a single MAP which uses a REPEAT loop to create and link instances. This can result in obscurity as small pieces of program text will replace the real conversions.

We think EXPRESS-M is a promising option for realising conversions, provided that the restrictions as mentioned in 5.2 will be overcome. We do, however, have some doubts about the possibilities to implement the language with the proper functionality based on the current ideas.

5.6 Transformr

5.6.1 Introduction

Transformr is a prototype tool for migrating STEP exchange files between different versions of the same EXPRESS schema (NIST, 1992). This migration differs from the mapping which must be carried out in
ATLAS, which involves two totally different EXPRESS schemas. In spite of this we analyzed Transformr because there was a prototype available, and for completeness sake. The inputs of Transformr are source and target EXPRESS schemas and a specification of the transformations which must be applied to convert a model from the source to the target schema. Transformr then reads exchange files corresponding to the source schema and writes files corresponding to the target schema.

5.6.2 Possibilities and restrictions

Transformr (officially TCSL, Transformr Correspondence Specification Language) offers two commands to convert data from the source to the target schema: COPY and BUILD. COPY establishes a direct conversion from instances of one entity in the source schema to instances of one entity in the target schema. BUILD specifies a construction for instances of one entity in the target schema based on tuples of instances from the source schema. Such kind of option is missing by EXPRESS-M. The COPY-command offers the possibility to skip/drop an attribute during the conversion or transform the values of the attributes. The restriction to the COPY-command is that for every instance in the source schema one instance can be made in the target schema. If you like to create two or more instance of different entity-types you must use the BUILD-command. The BUILD-command specifies that instances of the named target entity are to be built from one or more source instances. If necessary it offers the possibility to define a condition on the source instances to carry out the command or not.

The BUILD-command of Transformr has great resemblances with the MAP-command of EXPRESS-M but Transformr has some restrictions. The most important restrictions are mentioned below and also described in the manual of Transformr.

- TCSL uses the syntax of EXPRESS to define expressions but the current Transformr parser does not yet accept everythig. Some unimplemented parts are: aggregate constructors, aggregate operations (including indexing), entity instance constructors, group qualifiers, references to derived attributes and most function invocations (only built-in arithmetic functions of a single argument are currently implemented).
- There is no way to create user-defined functions.
- There is no way of converting an aggregate value from one class (array, bag, list or set) into another, short of writing a function.
- There is no way of referring to the instances constructed by a BUILD command.
5.6.3 Examples

The conversion between the view type model for the architect in the Global Design stage (VtM) and the attribute type model (AtM) of Speedikon as defined in ATLAS and the conversion benchmark are worked out in TCSL and can be found in the Annexe. The TCSL parser accepts a part of the syntax of TCSL. The examples in the Annexe and mentioned in this section contain some statements which are currently not supported by the parser. The unsupported statement are printed in italics. The example in this section is a part of the conversions from the Annexe and give an idea of the notation of TCSL. The example shows the conversion of a Building to a PROJECT. In this simple conversion two lines are not supported by the parser of Transformr. The first line shows a call of a routine which is not possible. The fifth line is not allowed because it is not possible to use one element of an aggregate.

BUILD PROJECT FROM Building;
DERIVE
   PROJECT.project_number := int2string(Building.building_ID);
   PROJECT.project_name := Building.building_name;
   PROJECT.proprietor1 := Building.architect;
   PROJECT.installation_date := Building.version_date;
   PROJECT.text[1] := Building.building_description;
   PROJECT.supervisor := Building.supervisor;
   PROJECT.project_reference := Building.project_title;

5.6.4 Conclusions

The available version of Transformr (and the parser) is only usable for simple conversions and has a great number of restrictions. This is partly due to the fact the only a small part of the defined language has been implemented. But even the full language does not offer sufficient expressive power to solve the ATLAS conversion problem (see the restrictions of section 6.2). These problems especially relate to coping with the different structures of the source and target models, e.g. connecting created instances. This is perfectly understandable because Transformr was designed for another purpose. Only if the TCSL will be expanded and the restrictions of section 6.2 are solved then Transformr will be a realistic option to use for complete conversion.
We contacted NIST about Transformr. Transformr is not currently supported. Plans are underway for further development, probably late in the year. NIST suggest looking at EXPRESS-M, but in the future also EXPRESS V2.

5.7 Miscellaneous

5.7.1 Introduction

This paragraph describes various other possibilities to convert data from one model to another. These possibilities are joined together in one paragraph because they are not analysed so extensive as the possibilities described in the previous paragraphs ((in)formal specification, programming languages, EXPRESS-M and Transformr). Section 7.2 describes the XVcM convertor which is a rule based tool developed by CSTB. In section 7.3 an option is described which is based on a federated database management system and developed by the University of Amsterdam. Section 7.4 describes Operation Mapping, a tool for conversion between different databases and developed by the Dutch company Metaform Software b.v. The last section (7.5) describes a mapping language for views which is being developed by the Computer Science Department of the University of Auckland (New Zealand).

5.7.2 The XVcM Convertor

5.7.2.1 Introduction

As defined in Paragraph 4 CSTB has developed a rule based approach to define the conversion of a given semantic model into another (target) semantic model. A rule based language has been defined and also a tool (the XVcM Convertor) implements this language. In this section a short description of the possibilities and restrictions of the rule based language is given according to the description of other tools.

As mentioned before the way to define conversion is based on ruled based languages. In this languages a number of rules are defined which act on a data structure. A rule tries to find a specific sub-structure somewhere in the data structure which meets the conditions defined in the rule. If a structure is found an
action (in the context of conversion a conversion) is carried out by the rule. Below the general description and two simple examples of a rule are given.

RULE rulename
IF condition-for-a-sub-structure
THEN action

RULE map_to_child
IF person.age < 18
THEN map_person_to_child()

RULE map_to_boss_relation
IF person1.department = person2.department AND
    person1 <> person2 AND
    department.boss = person2
THEN create_boss_relation(person1, person2)

5.7.2.2 Possibilities and restrictions

The expressiveness of the rule based language looks very extensive and powerful. It is possible to use complex constructions to define the conditions for a sub-structure of model and carry out conversion or creating relations. Most of the other tools/techniques for conversion have problems with the conversion of relations or creating relations because they convert entities/objects separately instead of sub-structures. Another advantage of the XVcM convertor will be the possibility to use the link between a source entity and its target entity. With this it is possible to create a relation during a conversion with an entity which was created before by a conversion in another rule.

5.7.3 PEER

5.7.2.1 Introduction

PEER is a federated, object-oriented database management system designed and implemented primarily to support industrial automation application environments [UVA93]. PEER is developed by the University of Amsterdam. The design of PEER is consists of a loose federation of autonomous, heterogeneous distributed database systems without a global schema. In the scope of conversion the significance of PEER lies in defining a schema derivation and integration mechanism for a federated database architecture to support the specification of interdependencies among activities in industrial
automation applications. The schema management mechanism is based on an object oriented data model and language. Complex definition of derived types and derived conversion are supported by a set of derivation/integration primitives. Remote referencing among agents can be established and their referential integrities are supported through the extension of local schemas.

### 5.7.3.2 Possibilities and restrictions

The integration part of PEER gives only the possibility to create relations between some entities and relations between some attributes in different data structures. The restriction of this relations is that the instances of an entity or values of an attribute can be converted only directly without a transformation. It is not possible for example to store the sum of two source attributes into one target attribute. It is only possible to make a selection with queries which is very cumbersome. PEER does not offer the functionality which is necessary for the conversion in ATLAS. Some missing functionalities are: the possibility to do transformations on values and the possibility to make selections (e.g. with IF's and/or CASE's) and the creation of objects. The possibilities of creating relations between different entities (more than two) does not look like to meet the needs of the ATLAS' conversion.

### 5.7.4 Operation Mapping

#### 5.7.4.1 Introduction

The Dutch company *Metaform Software b.v.* is developing software for the conversion (or mapping) between different databases [BIJ94]. The problem is stated as follows. Suppose we have two information systems, system A and system B. Both systems can handle the same kind of information. However, they store this information in different ways, using different file formats. So if we want to transfer information from system A to system B (or vice versa), the information has to be converted.

In order to solve this problem, first a distinction is made between the logical (or conceptual) level of the information, and the physical level. On the logical level, the meaning and structure of a file format is described. This can be done with a data modelling language such as EXPRESS. On the physical level, the exact representation of the information in the file is described. The can be the physical file format of
STEP, but also other formats such as DXF are supported. Solving a conversion problem now takes place only at the logical level, thus abstracting from any details of the physical representation. This approach is shared with most other methods which are developed to solve the conversion problem. The software developed is known under the name Operation Mapping. One more detail, Operation Mapping consists of two programs: the Mapper and the Converter.

The Mapper is an interactive graphical tool which should be operated by a user who is a conversion expert, i.e. who knows exactly how to relate information from systems A and B. The Mapper shows graphical representations of the logical schemas of both systems (although unfortunately not based on EXPRESS-G), and assists the user in relating entities and attributes from the source system to entities and attributes in the target system. This can be done by selecting first an entity or attribute in the source schema (by clicking a mouse button) and then selecting the corresponding entity or attribute in the target system. A schema is viewed as a directed graph, so each entity or attribute can be identified by the path from the "root" of the schema.

Example:

\[
\begin{align*}
A/drawing & \rightarrow B/drawing \\
A/drawing/lines[n] & \rightarrow B/drawing/lines[n] \\
A/drawing/lines[n]/label & \rightarrow B/drawing/lines[n]/label \\
A/drawing/lines[n]/color & \rightarrow B/drawing/lines[n]/color/red_value \\
& \quad \rightarrow B/drawing/lines[n]/color/green_value \\
& \quad \rightarrow B/drawing/lines[n]/color/blue_value \\
A/drawing/lines[n]/point[0]/point/x & \rightarrow B/drawing/lines[n]/x_start \\
A/drawing/lines[n]/point[0]/point/y & \rightarrow B/drawing/lines[n]/y_start \\
A/drawing/lines[n]/point[0]/point/z & \rightarrow B/drawing/lines[n]/z_start \\
A/drawing/lines[n]/point[1]/point/x & \rightarrow B/drawing/lines[n]/x_end \\
A/drawing/lines[n]/point[1]/point/y & \rightarrow B/drawing/lines[n]/y_end \\
A/drawing/lines[n]/point[1]/point/z & \rightarrow B/drawing/lines[n]/z_end
\end{align*}
\]

A mapping is checked on completeness and ambiguity. The result of a Mapper session is a set of translation rules, also referred to as a key.

The Converter can be used to actually convert files from one file format to another. To this end, a set of translation rules as produced with the Mapper is read and interpreted by a simple recursive algorithm. See the following figure.
5.7.4.2 Possibilities and restrictions

Operation mapping offers a nice solution for the conversion problem. A conversion is specified interactively on the logical level using a graphical user interface, so one does not have to bother about details of a physical file format. It is possible to use EXPRESS as a data modelling language. STEP physical file format is supported, but also other formats such as DXF.

It is possible to define the conversion of entity to entity, but also from entity to attribute and vice versa. In addition to this, it is possible to create, delete, split and combine entities and attributes. Also the use of arithmetic functions is foreseen.

The Mapper and Converter are available on both PC/DOS and Apple Macintosh platforms. They are sold separately. It is possible to buy some predefined keys for DXF, which can be adapted easily. These keys can be used for instance to change DXF entity attributes like layer and color.

The software is still under development. The Converter seems finished but work remains to be done on the Mapper. A number of successful tests have been performed (e.g. mapping a large DXF file to another DXF file, where lines are translated to lines and vice versa) but the examples used were fairly straightforward, so it is yet uncertain if more complex conversions like the ATLAS examples can be solved. The conversion benchmark from paragraph two was presented to Metaform. They claim they can solve this problem but time and budget was insufficient to support this claim, so further investigation is needed.

5.7.5 A Mapping language for views

5.7.5.1 Introduction

At the Computer Science Department of the University of Auckland (New Zealand), a mapping language for views is being developed [AMOR94]. The context for this project is a system with multiple views on the same (instances of) data. Each view represents its data according to a specific schema. Several views can be “active” at the same time, so a change of data in one view should be dynamically propagated to all other views. Doing so, it might be the case that representations of information differ, hence the conversion problem. The conversion problem is defined identically to the ATLAS problem, i.e. given two EXPRESS schemas, how can we convert entity instances (e.g. in a STEP physical file) of one schema to
the other? A so far unique property of the mapping language for views is that no distinction is made between a source and a target model, so a defined mapping is, in principle, bi-directional. Of course this is not always possible (think of an area which should be mapped to a width and height) and in those cases the user should interactively provide additional information.

The mapping language is based on Prolog, so it used a Prolog style of notation. It can handle instances of EXPRESS schemas by using some object-oriented extensions of Prolog. A mapping consists first of an inter_view description, describing the schemas to be mapped, and whether the mapping is complete (i.e. if all information in both models can be mapped) or not.

\[
\text{inter_view(Schema}_1, \text{Schema}_2, \text{complete}).
\]

Next, a mapping has a number of inter_class descriptions, describing how entities from both schemas should be mapped onto each other.

\[
\text{inter_class([\text{Schema}_1\text{\_Entities}], [\text{Schema}_2\text{\_Entities}],}
\]
\[
\text{\quad \text{invariants(\text{Invariant\_Definitions}),}}
\]
\[
\text{\quad \text{equivalences(\text{Equivalence\_Definitions})}).}
\]

So it is possible to relate one or more entities from the first schema to one or more entities from the second schema. The invariant section is optional, and describes the conditions under which this mapping is valid. The equivalence section describes the actual relationship between data in the named entities.

Some examples:

\[
\text{inter_class([\text{person}], [\text{male}],}
\]
\[
\text{\quad \text{invariants(\text{gender = male}),}}
\]
\[
\text{\quad \text{equivalences(\text{name = name,}}}
\]
\[
\text{\quad \quad \text{age = age,}}
\]
\[
\text{\quad \quad \text{unity = masculinity})}}
\]
\[
\text{\quad \quad \text{).}}
\]

\[
\text{\quad \quad \text{).}}
\]

\[
\text{inter_class([\text{person}], [\text{female}],}
\]
\[
\text{\quad \text{invariants(\text{gender = female}),}}
\]
\[
\text{\quad \text{equivalences(\text{name = name,}}}
\]
\[
\text{\quad \quad \text{age = age,}}
\]
\[
\text{\quad \quad \text{unity = femininity})}}
\]
\[
\text{\quad \quad \text{).}}
\]

(The example is taken from [NIST92].) Next to entities from two schemas, it is possible to use temporary entities which can be used in very complex mappings.

An interesting part of [AMOR94] is the comparison of the view mapping language with database theory. Equivalence of the language is shown to the five basic operands from relational databases (project, select, union, set difference, cartesian product), the higher order operands from relational databases
(intersection, join, natural join) and the ten operands defined for the definition of superviews for multiple databases (meet, join, fold, rename, combine, connect, aggregate, telescope, add, delete). In the remainder of [AMOR94], all examples from [NIST92] and [BAI94a] are described in the mapping language for views, together with some new examples from the author.

5.7.5.2 Possibilities and restrictions

The mapping language for views is one of the few languages for conversion which has a theoretical foundation, at least as far as the expressive power of the language is concerned. Attractive features of the language are the possibility to use temporary entities for complex conversions, and the fact that a mapping specification is in principle bi-directional. The fact that a Prolog style of notation is used, and that an object-oriented version of Prolog is probably used for an implementation, closely corresponds to our results from paragraph 4.

The mapping language for view is a promising development. No implementation is available yet, however, so it is not yet known whether the language defined can be successfully be implemented. Also, some more complex examples have to be used to test both the language and the future implementation, so a final conclusion cannot be given.

5.8 Classification of conversions

5.8.1 Introduction

The aim of this paragraph is to give a classification of conversions and to give an overview of the useful conversions. The results can be used to formulate requirements for conversion languages and tools. A number of examples are added to make clear what a conversion class looks like. The last section gives some conclusions.

5.8.2 A framework for classification

This section contains a classification of conversions which is based on the experiences during this
The aim of this classification is to show which types of conversions are theoretically possible and which are needed in practice. The classification is based on two selection criterions: the type of data (entities or attributes) and the number of source and target elements. The selection criterion "type of data" offers four options: conversion from entity to entity, from attribute to attribute, from entity to attribute and from attribute to entity. The other selection criterion "number of source and target elements" has more options which need an explanation. Most times there are many relations between the different entities in a model and it is not so clear to determine the number of entities which are involved in a conversion. For example: one source entity is converted to one target entity but the source entity used one attribute of a related entity. It is possible to call this conversion a 1:1 conversion or a 2:1 conversion depending on the criterion used. In this project the selection criterion is that a conversion is called a 1:N (or 1:C, N is variable number and C is a constant number) conversion if the meaning of the source element is described by N target elements. The same criterion is used in the other direction. Examples 1 and 2 show 1:1 conversions and a C:1 conversion.

Example 1, 1:1 mappings

Conversion Line -> Line
Conversion Point_2D -> Point_3D

ENTITY Line
  Start :  Point_2D ;
  end :  Point_2D ;
END_ENTITY ;

ENTITY Point_2D
  x :  REAL ;
  y :  REAL ;
END_ENTITY ;

ENTITY Line
  Start :  Point_3D ;
  end :  Point_3D ;
  length :  REAL ;
END_ENTITY ;

ENTITY Point_3D
  x :  REAL ;
  y :  REAL ;
  z :  REAL ;
END_ENTITY ;
ENTITY Line
    Start : Point_2D ;
    end : Point_2D ;
END_ENTITY ;

ENTITY Point_2D
    x : REAL ;
    y : REAL ;
END_ENTITY

ENTITY Line
    start_x : REAL ;
    start_y : REAL ;
    start_z : REAL ;
    end_x : REAL ;
    end_y : REAL ;
    end_z : REAL ;
END_ENTITY ;

Example 2, Example of a C:1 mapping

The following table shows all the possible conversions and indicates by an asterisk or nothing whether a conversion is realistic. The indication if a conversion is realistic is based on the experiences of the authors with the different conversions which are worked out and the conversion tools which are analyzed. In the next section a number of examples are given of the different kinds of conversions. At first, we thought that a N:M conversion was a realistic conversion but it seems possible to split the conversion into a number of 1:N and/or N:1 conversions. The conversion of an entity or an attribute to N attributes is also not realistic because this means that an entity must have a variable number of attributes which is impossible. On the other hand the conversion of one attribute to a number of entities is realistic, a nice example is the creation of entities for every element of an aggregate type. Because there is a strong relation between entities and their attributes, there will be a combination of the conversions as mentioned in the table. For example if one entity is converted to one entity then there usually exist a number of one to one attributes conversions.

<table>
<thead>
<tr>
<th>Conversion</th>
<th>Entity _ Entity</th>
<th>Attribute _ Attribute</th>
<th>Entity _ Attribute</th>
<th>Attribute _ Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>0:1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1:1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1:C</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>C:1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1:N</td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>N:1</td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>N:M</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.8.3 Examples

ENTITY Storey :
    Storey_ID : INTEGER ;
    Storey_name  : OPTIONAL STRING ;
    Storey_origin : Point_3D ;
    storey_height : OPTIONAL REAL ;
    storey_level  : REAL ;
    involved_sections: SET (1 : ?) OF Section ;
END_ENTITY ; -- Storey

ENTITY Point_3D :
    x_coordinate : REAL ;
    y_coordinate : REAL ;
    z_coordinate : REAL ;
END_ENTITY ; -- Point_3D

ENTITY FLOOR :
    floor_number : INTEGER ;
    height       : REAL ;
    level        : REAL ;
    origin       : SET (1 ; ?) OF REAL ;
END_ENTITY ; -- FLOOR

MAP FLOOR <- Storey :
    /* E -> E, C : 1 */
    floor_number : = storey_ID
    /* A -> A, 1 : 1 */
    level        : = storey_level ;
    /* A -> A, 1 : 1 */
    IF EXISTS (storey_height) THEN
        height : = storey_height ;
        /* A -> A, 1 : 1 */
    ELSE
        height : = 0 ;
        /* A -> A, 0 : 1 */
    ENDIF
    origin (1)  : = storey_origin. x_coordinate ;
    /* E -> A, 1 : 1 */
    origin (2)  : = storey_origin. y_coordinate ;
    /* E -> A, 1 : 1 */
    origin (3)  : = storey_origin. z_coordinate ;
    /* E -> A, 1 : 1 */

Example 3a

The first example (example 3a) shows a part of the conversion between the view type model for the architect in the Global Design stage (VtM) and the attribute type model (AtM) of Speedikon describes with EXPRESS-M. The classifications of the conversions are mentioned as comments. The attributes "storey_name" and "involved_sections" are examples of A->A and 1:0 conversions. The conversion of
the Storey to a FLOOR is called a C:1 conversion because the origin of the Storey is described by a Point_3D and the Storey together with the Point_3D is converted to the FLOOR. The conversion of the origin is also a conversion of an entity to an attribute. In this case the attribute is a SET and the conversion is 1:1. If the origin were defined as three attributes then there would be a 1:C conversion (see example 3b).

The conversion 1:C (or C:1) occurs more often than 1:N (N:1). In example 4 there is a nice example of a N:1 attribute conversion. In the target model there is an attribute material which contains a code (INTEGER). This code is used for a combination of a number of skins, the thickness of the skins and the material of the skins. In the source model all this data is stored in different attributes belonging to different entities. In the conversion a number of selections on the source attributes are carried out to determine the material code. There is also a C:1 conversion of attributes at the end of this example.

ENTITY Storey;
  storey_ID : INTEGER;
  storey_name : OPTIONAL STRING;
  storey_origin : Point_3D;
  storey_height : OPTIONAL REAL;
  storey_level : REAL;
  involved_sections : SET (1 : ?) OF Section;
END_ENTITY; -- Storey

ENTITY FLOOR;
  floor_number : INTEGER;
  height : REAL;
  level : REAL;
  x_origin : REAL;
  y_origin : REAL;
  z_origin : REAL;
END_ENTITY; -- FLOOR

ENTITY Point_3D;
  x_coordinate : REAL;
  y_coordinate : REAL;
  z_coordinate : REAL;
END_ENTITY; -- Point_3D

MAP FLOOR <- Storey;
  /* E -> E, C : 1 */
  floor_number : = storey_ID;
  /* A -> A, 1 : 1 */
  level : = storey_level;
  /* A -> A, 1 : 1 */
  IF EXISTS(storey_height) THEN
    height : = storey_height;
    /* A -> A, 1 : 1 */
  ELSE
    height : = 0
    /* A -> A, 0 : 1 */
  ENDIF
  origin_x : = storey_origin. x_coordinate;
  /* E -> A, 1 : C */
  origin_y : = storey_origin. y_coordinate;
  /* E -> A, 1 : C */
Example 3b

MAP WALL_DEFINITION (1 : ?) <- Wall ;
  REPEAT i = 1 TO SIZEOF (wall_segments) ;
  ...;
  CASE sizeof (wall_skins) OF
  1 : IF (wall_skins (1). skin_material = "HLZ12") THEN
    WALL_DEFINITION (i). material := 131 ;
    ELSE
    ERROR ;
    ENDIF
  2 : IF ((wall_skins (1). skin_material = "VMZ"). AND.
    (wall_skins (1). skin_thickness = 115). AND.
    (wall_skins (2). skin_material = "Poroton"). AND.
    (wall_skins (2). skin_thickness = 175)) . OR.
    ((wall_skins (2). skin_material = "VM3"). AND.
    (wall_skins (2). skin_thickness = 115). AND.
    (wall_skins (1). skin_material = "Poroton"). AND.
    (wall_skins (1). skin_thickness = 175)) THEN
    WALL_DEFINITION (i). material := 301 ;
    ELSE
    ERROR ;
    ENDIF
    OTHERWISE : ERROR ;
  END_CASE ;
  ...
  IF (wall_segments (i). wall_segment_start_height >
    wall_segments (i). wall_segment_end_height) THEN
    WALL_DEFINITION (i). wall_height := wall_segments (i).
    wall_segments_start_height ;
  ELSE
    WALL_DEFINITION (i). wall_height := wall_segments (i).
    wall_segments_end_height ;
  ENDIF ;
  ...
END_REPEAT ;
END_MAP ;

Example 4

5.8.4 Conclusions
Classifying conversions is not an easy task. One difficulty is to choose the classification criteria. The criteria we used (cardinality of the relation between mapped entities together with entity/attribute-ness) seem natural, but other criteria are conceivable. Using the cardinality criterion, it is not always clear how to classify a specific example, e.g. is it a real N:1 conversion or is it 1:1, where some auxiliary information is used. We had to introduce the informal notion of meaning (or concept) to solve this dilemma. Also, working with a number of examples one can never be sure to give a theoretically complete classification because new examples can always enter the picture. We did try, however, to generalise our results. The results of this paragraph can be seen as a first step towards a full-scale classification that may influence the future development of conversion tools. The result so far is used to evaluate existing tools and to give some recommendations.

5.9 Conclusions and recommendations

5.9.1 Introduction

This paragraph summarizes some of the work done, and tries to draw conclusions and give recommendations on how to tackle the conversion problem within the ATLAS project. A number of evaluations will be made. In the next section, the approach which was followed (which has been explained in section 1.4) will be evaluated, followed by an evaluation of the work in relation to the ATLAS project. An evaluation of the investigated possibilities to solve the conversion problem is given in section 9.4. In section 9.5, finally, some recommendations are given.

5.9.2 Evaluation of the approach

As described in section 1.4, the approach was to take three conversion example and use them as a means to investigate the conversion problem. Next to possible ways to informally and formally specify a conversion and ways to implement conversions using existing programming languages, several languages and tools designed especially to solve the conversion problem have been examined. Also, the examples have been used to try to classify conversions. The following table shows which combinations were exactly investigated.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>VtM _ AtM</th>
<th>VtM _ BPtM/LSE PtM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informal spec.</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ /103
The conversion benchmark proved to be a very useful example to test the alternatives. Although a relatively small and easy example, from a conversion point of view it shows sufficient complexity. Using the example, we were able to pinpoint a lot of deficiencies in most conversion alternatives. Using the conversion from VtM to AtM is less interesting from a conversion point of view, because (although the models are much bigger) the structure of both models is very similar, so no real problems turned up. Of course there was another reason to implement this conversion (using C++), namely it is part of the first ATLAS demonstration in June 1994. The third example (from VtM to BPM / LSE PtM) was not of much use because sufficiently detailed versions of the latter two models were not available in time. We did, however, try to take this conversion into account for the classification of conversions, because the nature of this conversion is different from the conversion from VtM to AtM.

The practical approach to use (real life) examples instead of a more theoretical approach proved to yield some interesting results (see the following sections). Because of the complexity of the conversion problem in the most general sense (e.g. what type of models can be involved in a conversion?), it is a useful way of research, for it can lead to requirements for conversion tools, and can be used to evaluate these tools. In addition, it is a way of evaluating the models involved (i.e., do the models contain the right (level of) information?, is an automatic conversion possible?, etc.). In the end, of course, a theoretical foundation of a solution remains always necessary.

9.3 Evaluation of the project

During the execution of this task, a number of problems were encountered:

- **Availability of the models**
  The main problem was the availability of the ATLAS models. It took more time to get stable versions of these models than originally planned. Some of the models (BPM and LSE PtM) are still not sufficiently detailed to use for a realistic conversion. This problem was partly solved by using a third, non-ATLAS conversion example.

- **Documentation of the models**
  Availability of ATLIAM and/or EXPRESS schemas of a model is not sufficient. Additional documentation on how to interpret the model is absolutely necessary to be able to implement a
conversion.

- Availability of software

Some of the conversion software was not available in time (e.g. XVcM, EXPRESS-M). This is a pity, because promising conversion software could not be evaluated.

As a result, the execution of the work was somewhat hindered. Sometimes creative solutions had to be found like devoting some of our time to help to develop the models involved. Most of the problems, however, could be solved in the end, because of the close collaboration between the project partners. For instance, when problems with the involved models were reported. Also, when model documentation was unclear or missing, it was no problem to obtain additional explanation of the models.

Conversion is a very popular problem at the moment. Only recently, developments in this area have started, and most documentation is currently under development. Because of the preliminary status of the documentation, several authors were contacted (CIMIO, NIST, University of Amsterdam, University of Auckland, CSTB, Metaform Software) which resulted in a useful exchange of information.

5.9.4 Evaluation of the alternatives for conversion

In paragraphs 4, 5, 6 and 7 a number of alternatives are given for implementation of conversions. A short summary of the different alternatives is given below, followed by a general conclusion. The table on the next pages summarizes our results. It shows a number of requirements, resulting from the research using the conversion examples. For each alternative we indicate whether the alternative meets the requirement.

5.9.5 Recommendations for conversion within ATLAS

It is not possible to give a single conclusion. For the practical implementation of conversions in ATLAS, using C++ (after making an (in)formal specification) is a reliable option. Depending on commercial versions of Prolog, the combination of a formal specification and Prolog may also be used. Of course using a special purpose conversion tool is to be preferred, because then specification and implementation can be obtained in one step. Of all tools evaluated, XVcM seems most promising, provided some minor details are solved and an implementation becomes available soon.

On a longer term, it is recommended that the advantages of for instance EXPRESS-M (using an
EXPRESS-like notation, and being the official STEP conversion effort) and XVeM are combined. A graphical specification (like Operation Mapping) using EXPRESS-G is also something to consider. For theoretical fundamentals, EDI and database theory may offer some useful concept and solutions.
<table>
<thead>
<tr>
<th></th>
<th>C++</th>
<th>Prolog</th>
<th>EXPRESS-M</th>
<th>Transfo rmr</th>
<th>XVcM</th>
<th>PEER</th>
<th>Operation Mapping</th>
<th>Mapping for views</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPRESS based</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>SPF/SDAI based</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>software available</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>?</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>supports all</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cardinalities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>functions +</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arithmetic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conditional</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conversion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>keep map source-target</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>partial</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conversion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>instance search</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>link target instances</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>examples evaluated</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bi-directional</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>procedural /</td>
<td>p</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>p</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>declarative</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conceptual /</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>physical</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>order can be</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>specified</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>graphical</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>specification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ /107
6. AN IT VENDOR VIEWPOINT

6.1. Objectives

In this section we position the objectives of the task 3200 in the perspective of an IT vendor: SNI. Let us indicate that four partners in the ATLAS project, CSTB, SNI, IEZ and TNO, are required to contribute to this task with regard to its integration within the LSE Application Platform. SNI represent one of the two IT vendors participating in the ATLAS Project (the other being IEZ) and have the responsibility of providing an LSE Application Platform based on SIFRAME.

The objectives stated by SNI were listed at an early stage of the task development process and the presentation is rather generic and only presents the IT vendor viewpoint. An attempt has been made to describe an overall picture of Computer Integrated LSE and discuss how it can meet the expectations of the end-user organization.

An Annex on Computer Integrated LSE is included which provides an introduction to the subject and covers the semantic integration framework used. View Conversion forms an integral part of the semantic integration framework and the way in which it contributes to the overall environment is a key motivation for the SNI effort.

6.2. Scope of Work

One of the key responsibilities of SNI is to provide the LSE Application Platform with a Data Manager equipped with SDAI interface, STEP-Processors, ATLAS Data Browsers etc [D404a]. Moreover, the Data Manager will incorporate the results of task 3200 by means of their further integration into the LSE Application Platform.

One of the key results of the task 3200 is the DAVOS modelling and programming paradigm and the XVoM and XVcM prototype service modules. The functionalities described in section 2 and experimented by means of XP-NIAM, XVoM and XVcM should be made available to the final end user of the SIFRAME platform, at the conceptual and at the operational (i.e. implementation) levels.

The modelling support could be granted by the integration of Application Tools such as the final version
of XP-NIAM or XP-XPRESS-G into the SIFRAME environment but this would not bring a decisive advantage as compared to their use as standalone modelling systems - given the fact that these systems communicate on the basis of neutral STEP based technologies. We should even consider that the use of the XPDI software facilities would grant the user with the required functionalities to perform a full scale modelling enterprise including of course the definition of his/her views.

On the other hand, the XVoM and XVcM systems only prototype the functionalities required at the operational level to take charge of the management and conversion of views associated to STEP repositories.

SNI will carry out further evaluation to determine whether similar functionalities, envisaged as extensions to the normal operations supported by the Object Management system (OMS) of SIFRAME, could be provided to the end-user of the LSE Application Platform.

It is quite clear that these view facilities can only be taken in charge at the implementation level if they operate on the data storage facilities maintained by SIFRAME. As this requires a careful assessment on SNI side to adjust the resources to the effort, the next deliverable will report on the feasibility of a full integration into the LSE Application Platform. In case this would be deemed unrealistic, the entire DAVOS modelling and implementation paradigm would be demonstrated by means of the XPDI software facilities.

6.3. Initial SNI results

The initial work carried out, by SNI, was to conduct a preliminary study, at first to analyse the needs and then explore various possibilities based on currently available tools and methodologies. Inputs received from Work Package 1 were consistently used to refine these ideas. As a result, the possibility of using various tools and facilities, currently available within SIFRAME, were reviewed. The initial findings are presented hereafter.

6.3.1. Use of Object Oriented Paradigm

Data abstraction and data generalization are concepts which are naturally supported within Object-oriented paradigm. The fact that the architecture of the LSE Application Platform and the Data Manager
are actually based on an object-oriented database management system should make implementation somewhat easier.

The objective of Task 3200 to provide a mechanism for data abstraction and generalization and these mechanisms are essentially needed in the modelling stage and in further implementation stages. Development and Specification of the LSE Project type Model and of the View type Models are based on both top-down and bottom-up approach. This is explained in the Annexe in section 2.2.

The object-oriented paradigm supports concepts such as abstraction, inheritance, encapsulation and polymorphism. Abstraction principle provides the ability to look at something as a whole without being concerned with internal details and it is possible to implement functional abstraction and data abstraction. Messages can be examples of functional abstraction and hiding the specific implementation details of data types or structures used is an example of data abstraction. Use of class definition in TIDL and ENTITY in Express is the mechanism to support data abstraction.

Inheritence provides a mechanism to relate objects having similarities in their characteristics. Using the inheritence property the generalization and specialization can be done. In Express this is taken care by the SUPERTYPE and SUBTYPE definitions and in TIDL this is implemented by using base and derived class definitions. This discussion on the possibilities offers by OO programming and the differences identified with the DAVOS modelling and implementation paradigm should refer to the sections 2.2. where the overall methodology is presented and 4.2 where its use for supporting the AEC framework is envisaged.

6.3.2. Adopting a strategy for developement of Information Models

One of the requirement set behind the development of information models is that their implementation will be user friendly or ease of use. This means the LSE Project Database should be a comprehensive repository of project-wide information. It should be supplemented by appropriate Viewing mechanisms because the amount of information stored in an LSE Project Database can be gigantic in quantity and very neutral in format.

The requestor of information shall be a user or an actor who will use one of the application tool to access the LSE project database. For the time being assuming that there exists a suitable mechanism for the tool to access the LSE Project Database (via direct database access or file transfer), we have to still provide a
mechanism to extract relevant data and appropriate mechanisms so that every request should not result in filtering the entire database.

The approach envisaged by SNI is to implement a number of pre-defined views based on type of user, life-cycle stage and level of granularity and application tools are accessing the LSE Project Database. Each view is expressed in terms of its information requirement as View type Model and is generic or neutral to a set of application tools. For example, for an architecture design view, application tools such as Speedikon and AutoCAD AEC can be supported.

A view represents a subset of the complete Project type Model and thus has pre-defined links to relevant portion of it. Both the PtM's and VtM's are developed and specified within the scope of the ATLAS Project. The motivation for the development of the VtM's can be summarised as a number of benefits:

- The neutral database is large. Access via a VtM removes the need to scan the entire data base for every data access;
- Allows for the creation of actor dependent relationships to speedup data access;
- Makes filtering of data easy.

With these motivations in mind a strategy can be adopted to influence the model definitions so that the mapping complexity can be reduced for the View Conversion Module. Initially several strategies can be considered:

1) The PtM is split into view-independent kernel and supplemented by view-dependent part. A conversion module must look for all kernel data and view-dependent data relevant to a particular view;

2) While it may not be possible to precisely divide the view-dependent part of a PtM into mutually exclusive portions (VtM's), some amount of substructuring can at least ease the problem for the view convertor. Resolving overlaps, as in the case where basic geometry will be common to both the architecture and structural designers views, will however require different mechanisms.

3) Between any VtM and its corresponding view-dependent portion within the PtM, a link can be defined so as to denote the total extent of the view. One way to implement this is to have some kind of mapping table which links starting points, end point and in-between points (important references). The most desired way will be to link all data-items via mapping table but this may
lead to the problem of table management overhead.

6.3.3. Other ways to support View Conversion

The other implementation choices available include the extension of the SDAI itself, the provision of a library on top of the SDAI or as an independent tool:

1) Views can implement functions (as object methods) to do some filtering based on pre-defined criteria. The requestor can also provide a key, or keys, to obtain either a summary or detail information;

2) Filtering algorithms implemented as library of routines (e.g. matrix operations) can be made available as a separate facility;

3) KB View Conversion facility can provide a dynamic mechanism to abstract views taking the View type Models (or combination of them) as an input. This will also support filtering of information to the required level of granularity.

Besides these concepts, some implementation related issues need to be addressed. One of the essential questions is: how the concepts introduced by the DAVOS paradigms and how far the functionalities prototyped and demonstrated by means of the XVoM and XVcM prototype service modules can be integrated into the OMS layer of SIFRAME to support the view mechanisms atop the OMS data repositories?

It is quite clear that the encapsulation of the XVoM and XVcM prototype service modules into SIFRAME would not provide direct view mechanisms on the internal SIFRAME OMS repositories, a functionality which would be needed if view mechanisms were to be supported by the LSE Application Platform.

These possibilities will be assessed by SNI and reported in the next release of the deliverable.
7. CONCLUSIONS

This deliverable has addressed the issue of data abstraction, generalization and view conversion. A meta methodology - meta in the sense that this methodology is developed atop the modelling methodologies adopted by the ATLAS project - has been presented and the way it can support the definition of conceptual models at different levels of abstraction and corresponding to different views on the products has been described.

A number of CASE tools have been developed to support the modelling methodologies of the ATLAS project (ATLAS, D104) and these systems have been further adapted to the meta methodology proposed to handle complex conceptual spaces. The AEC modelling framework proposed by (Gielingh, 1994) is one example of such modelling spaces where the models are distributed at different levels of abstraction along a generalization / specialization axis.

Abstraction mechanisms are supported at the conceptual level by the supertype / subtype relationships and by appropriate structural overloading mechanisms and by multiple inheritance functions developed at the implementation levels, so that the products can be addressed at different levels of abstraction.

The other axis of the modelling framework aforementioned (Gielingh, 1994) is mainly systems oriented. Of course, other modelling frameworks could have been used as well, such as the ones from (Böhms, 1994) where the models fall into matrix cells, from (Tolman, 1994) with combined systems and discipline approach or from (Wix, 1994) where many more dimensions appear. The view mechanism proposed in this deliverable, with the very abstract understanding of the concept of view, is adapted to model the other dimensions proposed by these framework (i.e. actors, disciplines, etc.).

The physical separation at the conceptual level of the various models supposed to be later considered as views on the same product model (e.g. systems, actors, disciplines) is the first stage in the meta methodology proposed. Moreover, the external NOLOT concept enables the modeller to check the correct assembly of these views (located at the same abstraction level) with models located at an higher abstraction level along the generalization / specialization axis.

At the implementation level, these models (or their implemented form) can be considered as views defined on the same product model. The appropriate structural overloading mechanisms have been
presented and developed in the XVoM system so that specific views could be managed on the project models (i.e. the data populating real projects). These views can also serve the purpose of the neutral communication of partial project models by means of SPFF files depending on the actors / disciplines concerned.

Moreover, an experimental system has been prototyped, i.e. the XVcM service module, so that conversion between models at the same abstraction level could be envisaged. An approach based on a first order rule language has been used to demonstrate how the problem of view conversion could be addressed.

All these concepts and the implementation mechanisms required to support data abstraction, generalization and view conversion have been called the DAVOS (Data Abstraction and View Oriented System) modelling and programming paradigms. The way the DAVOS approach could help solve the inter-operability problems encountered in complex modelling frameworks has also been addressed.

Future work will be required to refine these concepts and the corresponding implementation strategies, and a careful assessment should be made to determine the possibility to offer these functionalities to the end users of the LSE Application Platform.
8 REFERENCES

(Amor, 1994) "A mapping language for views", Robert Amor, Departmental report, Computer Science Department, University of Auckland, Auckland, New Zealand (to appear)


(Bijnen, 1994) "Exchangeability of digital information, up to an easier data transmission", A.H.J.M. Bijnen (to appear)


(IEZ, 1992) "Speedikon High Level Interface", IEZ, Bensheim, 1992


(Neuteboom, 1994) "ATLAM model Speedikon Architecture", Johan Neuteboom, Michel B_hms, ESPRIT 7280 - ATLAS, 1 February 1994


Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ / 115


(Stepney,1992) "Object orientation in Z", Susan Stepney, Rosalind Barden, David Cooper (eds.), Springer-verlag, Berlin, 1992


(Spivey, 1992) "The Z Notation, a reference manual (2nd ed.)", J.M. Spivey, Prentice Hall, New York, 1992


Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ /116
ESPRIT 7280 - ATLAS - D302a: Data Abstraction, Generalisation and View Conversion


Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ /117
ESPRIT 7280 - ATLAS

- D302a: Data Abstraction, Generalisation and View Conversion


Issue Date: 15/10/94 - Authors: CSTB / TNO / SNI / IEZ / 118
9 List of the Figures

Figure 1. COMBINE as a first experience of DTP-oriented View type Models............................................. 4
Figure 2. XP-NIAM - Overloading of External NOLOTs with various VtMs..................................................... 12
Figure 3. XP-NIAM - Simplistic VtM1............................................................................................................. 13
Figure 4. XP-NIAM - Simplistic VtM2............................................................................................................. 14
Figure 5. XVoM - Multi Views (Schemas) implementation Platform................................................................. 15
Figure 6. XVoM - Building entity in GPM........................................................................................................ 16
Figure 7. XVoM - Building entity in VtM1......................................................................................................... 17
Figure 8. XVoM - SPPF for VtM1 and GPM schemas......................................................................................... 19
Figure 9. XP-SDAI a multi-repositories implementation platform.......................................................................... 19
Figure 10. A syntactical example of the expressiveness of the rule language..................................................... 25
Figure 11. Rule mapping a COMBINE building on an ATLAS building............................................................. 26
Figure 12. Rule mapping between "spatial systems" (COMBINE -> ATLAS)......................................................... 27
Figure 13. Rule mapping C-space->A-elementary-space-of-section-space......................................................... 28
Figure 14. Rule mapping C-building-spatial-system->A-elementary-space-group-of-storey-space................. 29
Figure 15. Rule mapping C-space->A-elementary-space-of-storey-space......................................................... 30
Figure 16. Rule mapping C-building->A-space-enclosing-system-and-space-dividing-system....................... 30
Figure 17. The AEC IARs / AICs and APs Framework (Gielingh, 1994)............................................................. 39