A PRODUCT MODEL BASED ARCHITECTURE FOR ENGINEERING APPLICATIONS SOFTWARE

Names of Authors. Alastair Watson[1], **Wanjau Wambugu** [2]
[1] CAE group, School of Civil Engineering, Leeds University
[2] CAE group, School of Civil Engineering, Leeds University

ABSTRACT: Some progress has been made in the establishment of agreed product models as the basis of a more open information-centric approach to engineering processes, both in the construction sector and beyond. Specific construction related initiatives, such as the CIS, the IFC and AP225 have been implemented in software and are now beginning to impact on industry. In common with other engineering sectors, these standards have drawn heavily on ISO 10303 (STEP), particularly the EXPRESS data definition language.

The paper builds on this background and identifies some of the barriers to the adoption of product model based standards. The particular problem of information mapping is described and discussed, both in the current context of building data exchange translators for existing applications software and, more specifically, the implications for creating new more interoperable software. A conceptual architecture that addresses this problem by means of a hierarchy of view windows is proposed. This architecture is described, its likely strengths and weaknesses identified, and a prototype implementation reported on. The paper concludes with some speculations on the possible implications of such an architecture for future engineering applications software.

KEYWORDS: Product Model, System Architecture, Applications Software.

1. INTRODUCTION

Product model based data exchange standards are only now beginning to make an impact on the construction industry. However, their importance is likely to grow as the industry becomes increasingly information-centric. Vendors of engineering software are already experiencing difficulties in designing and implementing the data exchange translators necessary to comply with such standards. Unless a data architecture can be provided that will enable future applications software to be more closely aligned with agreed product models, these difficulties are likely to become acute.

2. PRODUCT MODEL BASED STANDARDS TODAY

Although the origins of today's product model based standards are firmly rooted in ISO/STEP (Owen, 1997; STEP, 2000), the latter is virtually unknown within the mainstream construction industry. While engineering sectors such as the automotive and aerospace have been very active in developing appropriate STEP "Applications Protocols", the STEP Building and Construction group[1] has had an honourable but relatively unproductive history. To date, the only building and construction Application Protocol to have been ratified is ISO10303 Part 225 (DIS) "Building elements using explicit shape representation".

[1] ISO TC184/SC4/WG3/T22, Building & Construction Group

The development of AP225 was led by Wolfgang Haas with substantial financial support from the German government and active participation from Japan (in particular). Although AP225 data exchange translators have been implemented for a number of 3D CAD applications, it is perhaps significant that the German construction industry appeared not to be ready to embrace 3D model-based working. Since 1997 Wolfgang Haas has also led a newer initiative, known as STEP-CDS "Construction Drawing Subset" (Csavajda, 1998), which is based on subsets (Conformance Classes) of two existing Application Protocols:

- ISO10303 Part 202 (IS) "Associative draughting"
- ISO10303 Part 214 (DIS) "Core Data for Automotive Mechanical Design Processes"

This initiative, which was instigated by German-based automotive manufacturers (VDA) and includes several major CAD vendors, will enable more traditional drawing-centric data exchange to take place within a STEP context. Japan currently has a similar initiative that is known as SCADEC.

One reason why the STEP Building and Construction group is failing to attract a critical mass of active Application Protocol development activities was the launch of the International Alliance for Interoperability (IAI, 2000). The origins of IAI lie in an initiative by AutoDesk, and some of its major clients, to provide models for use with its then emerging object oriented CAD software. This initiative was subsequently reformulated as the IAI. In 1996 it was launched globally, with very ambitious objectives and timetables, on an unsuspecting (but essentially receptive) construction industry. That the IAI prospered is probably due to the fact that it focused on cross-disciplinary co-ordination, something that a STEP Application Protocol is inherently less able to do. IAI has made slow progress – as the full magnitude of the task was recognised – and has wisely borrowed heavily from STEP both in developing and in implementing its Industry Foundation Classes (IFCs). Although the IFC has considerable longer-term potential, the current state of software implementation is confused (LU, 2000), with software vendors spread over several different releases of the IFC. Hard data on the availability of commercial implementations is difficult to obtain, and this strongly suggests that industrial usage is still largely limited to trials (Leeds University, 2000).

An outcome of the CIMsteel project, comprehensive product model based standards also exist to support the engineering and the realisation of structural steel frames. Known as the CIS, the development of these standards was instigated in 1987 when STEP was still very immature. They thus draw heavily but pragmatically on STEP. To simplify its implementation in software, CIS/1 was deliberately simplified prior to its release in 1985. Although CIS/1 (Leeds University, 1995) has been quite widely implemented, effective industrial deployment has only been achieved by the more committed end users. CIS/2, which removes the limitations of CIS/1, was formally published early in 2000 (Crowley, 2000), with software implementations commencing in 1999 (Leeds University, 2000).

Many reasons why the construction industry has been slow to adopt product model based standards can be put forward, but these can be conveniently collected under three headings:

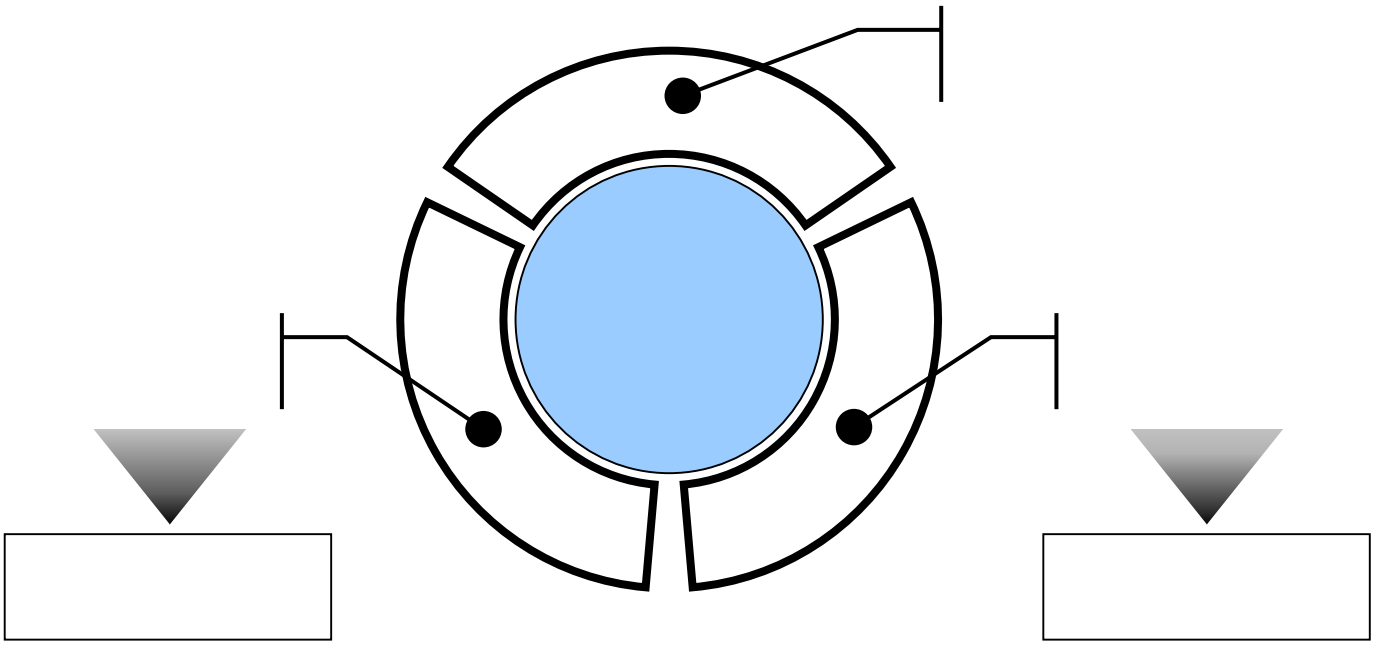
Time and effort to develop: That such standards are costly to develop is partly due to the intrinsic technical difficulty of the task. Additionally, the nature of today's construction industry requires globally applicable standards that also span the different disciplines. Differences in engineering practice, together with political and

commercial factors, conspire to increase the development cost of such standards. At the same time, there is still no effective means of funding this work.

Investment to implement: For a standard to be usable by industry, it must first be implemented in software (initially to prove the standard, and then as a commercial product) in the form of data exchange translators for applications software. Because of the inherent technical complexity of this task, the required investment is both significant and difficult to quantify. It can be difficult for a software vendor to make the business case, particularly for those vendors who are the first to implement.

Industrial changes to fully exploit: The existence of appropriate and implemented standards can be seen as a strong agent for change. While typically they can be deployed to support the existing industrial processes, there is already clear evidence that industrial change will follow to more fully exploit the potential of the standard. For example, the most successful adopters of the CIS have been designers who have accepted the overall benefits of working in 3D, and have then sought to re-engineer their business relationship with steelwork fabricators.

Clearly these barriers (see Figure 1) are not unique to construction, but they do present more of a constraint in this context. Two significant technical constraints can be identified. That of developing appropriate product models, and the so called "information mapping" problem encounter during software implementation. It is the latter problem that underlies this paper.



3. FOCUS OF THE ARCHITECTURE

3.1 Introduction

The conceptual architecture that is introduced in this paper can be seen as a means of reducing the information mapping problem encountered when developing data exchange translators for existing engineering application software. This problem arises because application software each has its own (particular) data structure, while the product model that underpins the data exchange standard has a different (more generic) data structure. Thus a

data exchange translator must map information between two different information domains, and typically the differences between these domains are considerable.

Although the architecture provides a means for reducing the current information mapping problem, its longer-term objectives are more profound. The goal is to investigate the implications of the architecture for the structure of future engineering application software.

3.2 The Information Mapping Problem

Each of the product models that have been developed for current data exchange standards are the result of sustained efforts to understand the information requirements of a given domain. The resulting data structures represent a best attempt to provide coherent support for identified information flows within that domain. The choice of data structures depends upon many factors, including who was involved in the modelling. There is no unique correct solution. A major objective is the avoidance of ambiguity in the model, "efficiency" is generally less important (as the criteria is not obvious).

Product modelling has achieved a level of pragmatic maturity. Almost all models are now formally defined using the EXPRESS (Schenck, 1994) data definition language, and most seek to avoid re-inventing the wheel by making use of previously established data structures. For example, both the IFC and the CIS draw heavily on generic resources defined by STEP. This has resulted in areas of similarity between different product models. However, product models have also grown to become large and complex. They now demand considerable time to understand. Additionally, as interest grows in data sharing rather than simple data exchange, product modellers are increasingly providing data structures to support the management of the engineering data. For example, the CIS/2 product model includes extensive support for the maintenance of meta-data.

While the development of a product model is generally an open (or partly-open) collaborative process, the internal data structures of an engineering application are frequently commercially confidential. They may have evolved over several generations, and may be well (or less well) documented. There is no standard way of defining these data structures – which may be physically realised as a database, or as live data structures within the program. In contrast to a product model, they are developed to support particular engineering functions and are likely to have well defined efficiency criteria.

Mapping information between the *generic* domain of the product model and the *specific* domain of an application is inevitably a difficult task. It requires the program states under which data can be exchanged to be identified, and demands a good understanding of both data structures. Prior to any software implementation, a mapping table is frequently developed to fully specify how the information should be mapped (in one particular direction). For each potential new data entity in the sink domain, this table identifies where the necessary data can be found in the source domain. The required mapping can be simple when similar (corresponding) data entities exist in both domains. However, in other cases, a complex query on the source domain may be required to collect data from many individual entities and thus acquire the data necessary to create a single new entity in the sink domain. Where different representations are employed in the two domains, complex numerical procedures may also be required to convert engineering data between them.

In addition to these direct manifestations of the information mapping problem, the degree of indirection gives rise to other important consequences. These include:

- A substantial increase in the scope for misunderstanding, confusion and errors relating to which of the standard's conformance classes (valid implementation sub-sets) could-be, or have-been, fully and correctly implemented by a translator. This is the single most critical aspect to achieving successful data exchange.
- Fundamental difficulties in realising data management, both in the context of data exchange and data sharing. Existing data management schemes depend on the preservation of a unique ID for data entities, and the maintenance of meta-data about these entities, as they flow into, through, and out of an application. This can be extremely complex to realise for a legacy application with more than a nominal amount of indirection between the two information domains.

Formal mapping technologies, such as EXPRESS-X, can offer a partial solution to the more direct manifestations of the information mapping problem. They provide a declarative means of specifying and automatically generating data mapping logic. However, both information domains must first be defined formally, and technical limitations still require more complex mappings to be hand coded. By enabling active data links to be specified between individual information models, mapping technologies could be employed in the future creation of large federated product models. Thus, they may well assist in overcoming the problem of developing product models.

3.3 The objectives of the Architecture

The architecture is based on a presumption: that growing pressure from end-users to improve interoperability will result in engineering software developers implementing new applications software based on accepted product model(s). Thus the goal is to provide developers with a practical means of working with an externally imposed model.

The broad objectives of the architecture are to:

- Maintain the underlying data in strict compliance with a standard product model (to facilitate data exchange or other form of interoperability).
- Allow each application programmer to work with a smaller and more appropriate schema, with scope for additional validity constraints.
- Enable the application programmer to specify what sub-set of the data is to be made visible to them.
- Provide a coherent framework for the delivery of these capabilities, including the delivery to the application programmer of persistent data objects with in-built context dependent behaviour.

4. THE PROPOSED ARCHITECTURE

4.1 Description of the Conceptual Architecture

The architecture builds on the availability of generic industrial-strength STEP implementation technology. This technology delivers a data repository capability, within which product model data is held in compliance with an EXPRESS schema, an interface through which the data entities can be manipulated, plus the capability to import or export data in the form of a STEP Part 21 exchange files Such technology substantially simplifies

the task of writing translators for existing applications software, but does not specifically address the information mapping problem.

By employing concepts found in the view mechanism of many database management systems, the proposed architecture delivers similar benefits to the user. For example, understanding is increased by reducing the scope of the visible data, data is presented to suit the use, and usability is maximised through devices such as providing derived data. This is achieved by enabling the definition of a hierarchy of **windows**, each of which may provide a different **view** onto the same underlying product model data. The usage of these multiple "interfaces" also parallels that of database management systems. Typically an engineering application program would employ an appropriate window to access the underlying data. Thus the software developer is regarded as the primary user of the architecture. However, equipped with an appropriate browser, an end-user might also select an appropriate window for the same purpose.

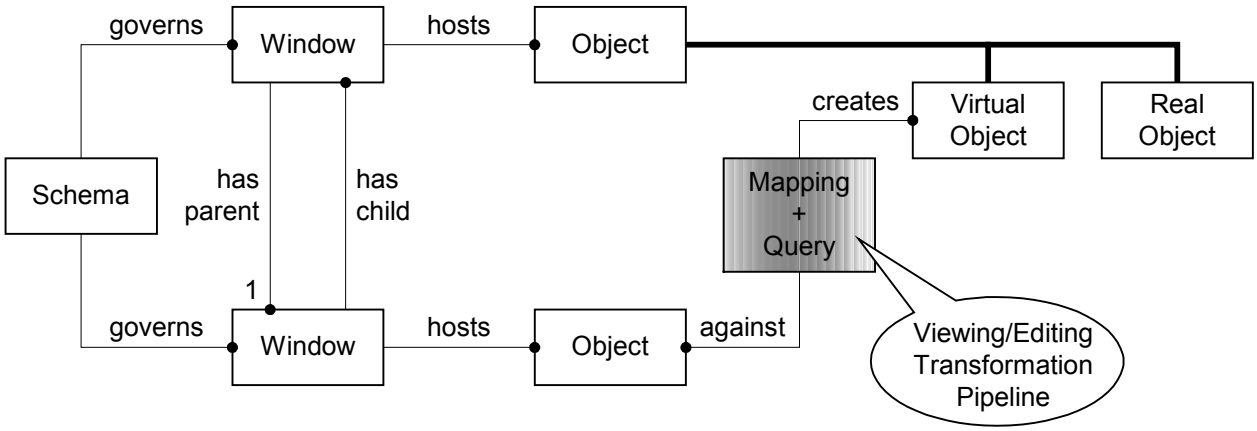


Figure 2. Window and Parent Window

A key feature of the architecture is that the view provided by a window is defined relative to its parent window as illustrated in Figure 2. The window at the root of the hierarchy of windows is embedded in the repository and is called the **base window**. This window presents the most direct view to the instances of data entities stored the repository. The other **non-base windows** each employ a sequence of one or more **transformation pipelines** to generate the required view. Although each window presents the data as **objects**, the architecture allows for the schema employed by any window to be defined using EXPRESS. As this implies, through schema evolution the architecture allows a restructuring of the data view between a non-base window and its parent window.

As is illustrated in Figure 3, each non-base window has a transformation pipeline. This is used to build that window's view by creating virtual objects that correspond to the objects in the view of its parent window. Each pipeline typically has two stages: a **query** against the objects in the parent view (providing a selection mechanism), and a **mapping** of these objects onto the window (allowing schema evolution). The transformation pipeline of each window is specifically designed to suit the intended use of that window. The queries are generally parameter driven, while the mapping can be. Where appropriate, the pipeline may omit the query (meaning all the parent objects are mapped) and/or the mapping (meaning that there is no schema evolution).

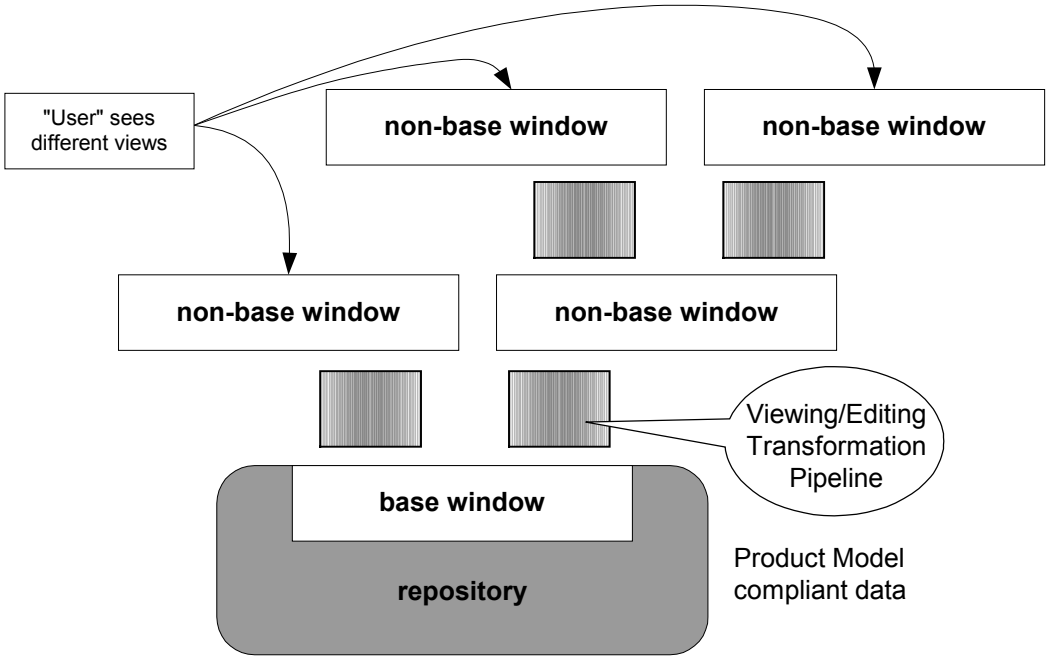


Figure 3. Components of the window hierarchy

Non-base windows normally employ **virtual objects** (which carry no data themselves), but may include **real objects** (which do carry data). The role of real objects is to increase flexibility by the introduction of local data into the window hierarchy. This enables schema evolution to introduce completely new object types (relative to the parent window). Real objects are given persistence by their host window. In the special case of the base window, the view is presented by **base objects** which directly manipulate the persistent data in the repository. Figure 4 illustrates how the various elements of the architecture interrelate.

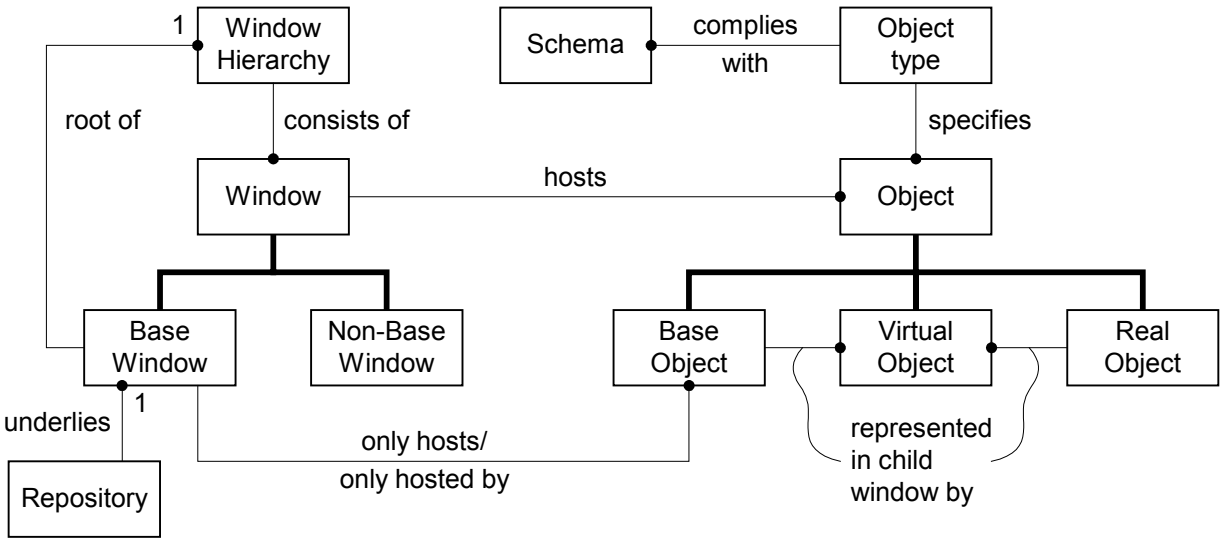


Figure 4. Components of the Architecture

To activate a particular view, the corresponding chain of windows within the hierarchy needs to be opened in sequence, and the virtual objects created, starting at the base window. Similarly, to edit the resulting objects, the effects must be cascaded window-by-window

down to the base window and the data in the repository changed. Before the revised data can be accepted it must first be validated against the EXPRESS schema of the governing product model. Then, as each view in the window-chain is re-established, each window may itself perform further validation of the objects. Pre-creation validation may form part of the query in the transformation pipeline. Post-creation validation might include constraints specified in the relevant schema and/or specific engineering constraints. For an edit to be accepted, the new objects must re-appear in the active view (indicating that the validation cycle has been satisfied).

The mapping within a transformation pipeline is thus a two-way process. Object existence is mapped-up from the parent window, resulting in the creation of virtual objects to build the view. In response to subsequent editing of that view (create object, delete object, edit attribute), it is the effects of such changes that are mapped-down on to the parent window. The architecture raises several interesting implementation issues, these are addressed in part by a transaction model (see Figure 6).

4.2 Some potential strengths and weaknesses

The *strengths* of the proposed architecture include:

- Founded on a product model, the architecture offers the user the option to work with a more appropriate view of the underlying data.
- Employing standard industrial-strength technology (where appropriate) for data validation against EXPRESS schemas.
- The integrity of the data can readily be guaranteed via transactions.
- Persistent local data can be incorporated seamlessly, providing additional scope to users.
- The data can be presented at different abstraction levels in different windows, thus facilitating the application of alternative manipulations to the same data.
- Data filtration may be hard-wired into a window by excluding object types through schema evolution, and soft-wired into the transformation pipeline query.
- The window concept is modular, flexible and readily extensible. The window hierarchy can be extended incrementally as the need arises simply by defining new windows. Maintenance is inherently simple.
- The window hierarchy provides an effective context for embedding and delivering object functionality (management and engineering methods) to the user.
- The notion of specialisation is inherent in the architecture. The base window would normally have the most complex schema and the greatest data scope with the windows located towards the tips of the hierarchy becomming increasingly specialised (with simpler schemas and reduced data scopes). The inclusion of embedded methods within this notion substantially amplifies the potential advantages to the software developer.
- The architecture allows specified branches of the window hierarchy to act as a library, making available object definitions and object functionality to other locations in the hierarchy.

Potential *weaknesses* of the proposed architecture include:

- Concurrency of access by different users and/or via different views has not been addressed. This is seen as being of critical importance.
- Where the correspondence between object types is not one-to-one, the definition of robust transformation pipelines may be difficult.

- Within the hierarchy there is the possibility of constraint conflicts between windows, such that one window can make changes which would cause the same data to be invalid in another window. Such conflicts must be avoided through careful design of windows.
- Depending upon how the architecture is implemented, once an edit transaction has started, the data may not be accessed until re-validation is completed.

5. AN INITIAL PROTOTYPE

5.1 Introduction

Prototype software has been written as an exploratory proof-of-concept for the architecture. This required that the conceptual architecture be mapped, for the first time, into a viable implementation form. An available product model was selected and appropriately restricted engineering scenarios identified. To support these scenarios a two-window hierarchy was designed, each with a governing EXPRESS schema specifically designed to support the intended use of the window.

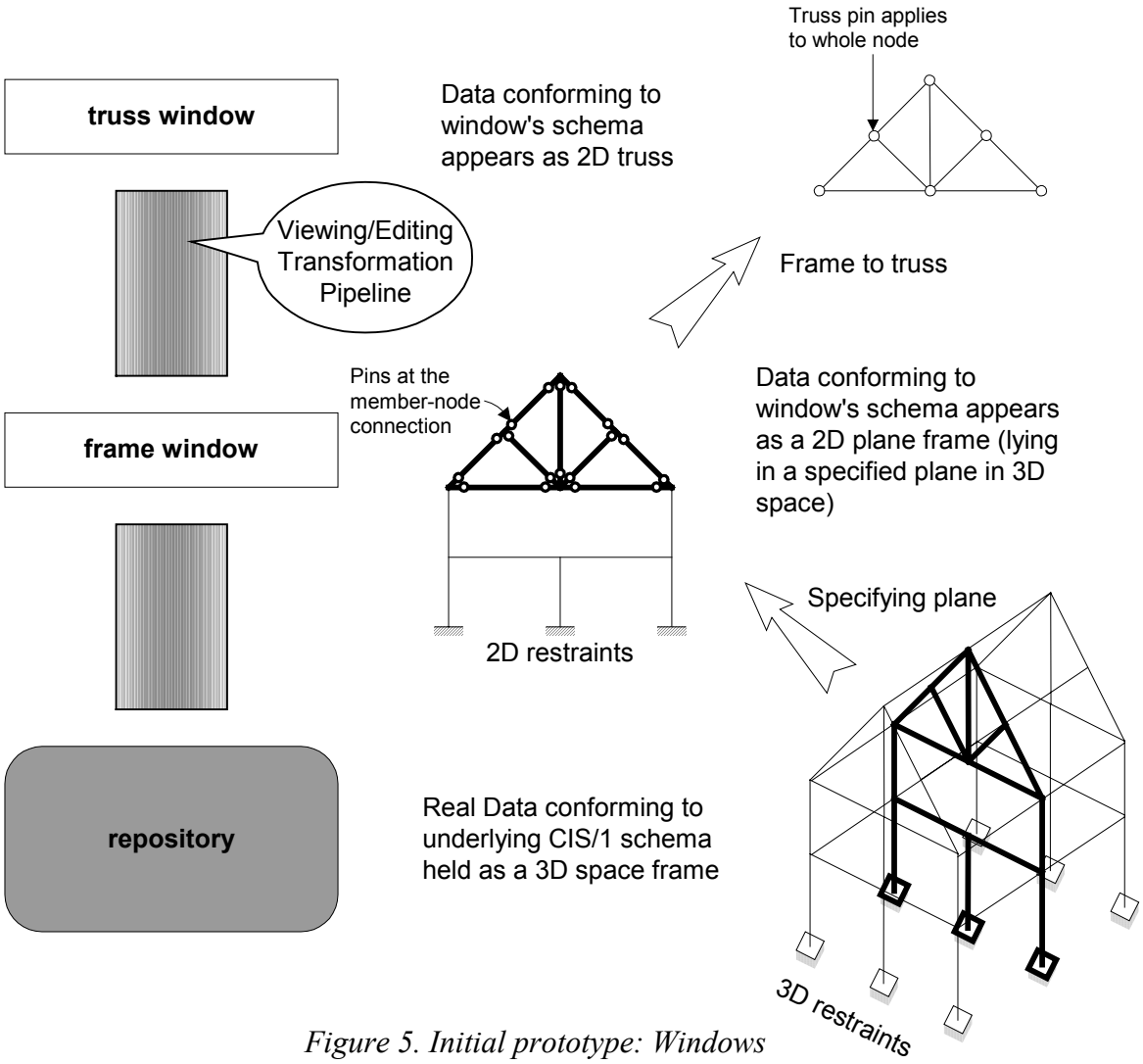


Figure 5. Initial prototype: Windows

5.2 The Implementation

The prototype was implemented using Visual C++ version 5 and ST-Developer version 7 (STEP Tools, 1999). The latter is a fairly typical example of STEP implementation technology, although the implementation makes heavy use of the native ROSE late bindings in preference to the more generic SDAI C late bindings.

The prototype is concerned with structural analysis, more specifically with the input of structural model data. As illustrated in Figure 5, the software comprises:

- a truss window where data is viewed as a 2D truss
- a frame window where data is viewed as a 2D frame
- a repository in which data is stored using the CIS/1 product model

To save time the base window was not formally implemented, and STEP technology was used more than envisaged in the conceptual architecture. The three schemas were unified into a single EXPRESS implementation schema, which was used to configure the repository and to guide the creation of corresponding C++ object types. Each instance of a product model entity thus corresponds to a pseudo base object, while each instance of the additional window entities corresponds to either a frame or truss virtual object. Every instance of a window entity has attribute(s) that point at instance(s) corresponding to those object(s) in their parent window from which they obtain data. Functions were written to act on the objects plus their corresponding repository instances.

Access to the architecture, including the opening and closing of the repository and the windows, is provided through a series of functions written in C++. The implementation schema is structured in such a way that functions can also simulate data navigation and validation. The functions use the STEP toolkit to navigate between instances (except via inverse attributes) and for data storage, Part 21 file creation and partial data validation (uniqueness). The rest of the functionality was hand coded.

To implement the prototype more detailed consideration had to be given to the system states and the transaction model as illustrated in Figure 6. These have not been fully implemented, each function is simply regarded as a discrete transaction.

The windows encapsulate the input and editing of node, member, support and release data. As illustrated in Figure 5, the transformation pipeline of the frame window features a parametric query and a parametric mapping. This enables the selection of those data elements that lie in a specified plane in 3D space, and their mapping into 2D space. The more specialised truss window employs a simple manual selection of the required data elements, and a mapping that converts between frame and truss data. Both windows include constraints to ensure that the entered data is valid.

The software provides ancillary functions to allow the end-user to create the skeletal object framework for a frame or truss in the repository. The object set can then be viewed and new data can be entered via the windows.

5.3 Some issues raised by the Prototype

Relative to the underlying product model, the schema used for the frame window demanded a relatively complex mapping. Some product model objects are merged, and others are split, in the frame window. Similarly the pin and support frame objects are existence dependent on

attribute values in the product model. It took some time to work out how to implement the mapping for the transformation pipeline, and that used in the prototype is no longer seen as the best solution. Since all mappings must be completely robust, our current preference is for each mapping to be kept relatively simple.

The frame window pipeline also exposed a situation were the query needed to incorporate a validation filter. Because the repository may contain analysis model data that is valid against the (more generic) product model, a filter is required to prevent the transformation pipeline creating objects that might not be viewable by the (more specific) display logic attached to the frame window.

Several of the methods used to implement the truss window where inherited from the parent frame window. Because the toolkit does not support the EXPRESS inverse clause it was necessary to write code to simulate this relationship. Unique entity names where preserved within the implementation schema by prefixing the names of the entities from the two windows with the name of the window.

The initial prototype did not reveal any major flaws in the architecture, and resulted in further refinement of the concepts – particularly relating to update transactions. Although this prototype was not designed to investigate questions such as the run-time efficiency of the architecture, it did suggest means by which a relatively efficient implementation might be created. While a robust environment to implement the architecture could probably be created relatively easily, the major investment would lie in the subsequent development of windows and their associated transformation pipelines.

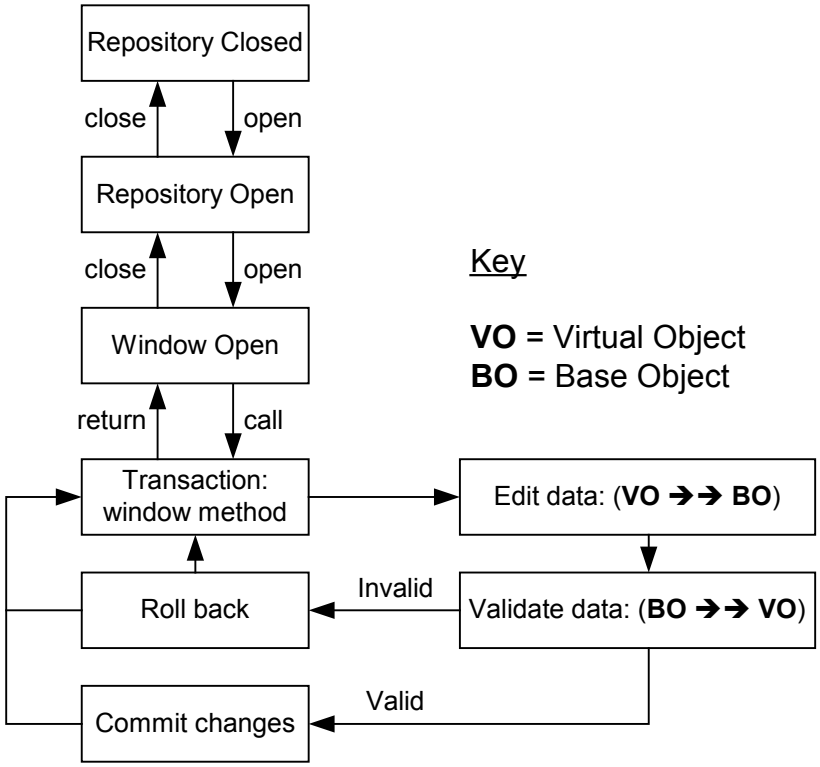


Figure 6. Initial Prototype: System states

6. CONCLUSIONS

The adoption of standard information models (to provide more effective interoperability between applications software) is likely to impact significantly on the future architecture of such applications software, and probably on the industry that provides the applications software.

The view-based conceptual architecture described in this paper appears to be broadly feasible. It offers the potential to address some of the challenges posed to software developers by product models. In the shorter-term it could be used to make more appropriate views available to translator developers. However, it is the longer-term vision of compliance being achieved by developing new software around specific views of the product model which is attractive. Ultimately, the feasibility of this approach will hinge on balancing the cost of realising the required views against the benefits to the application developer. Some grounds for optimism are provided by the inherent flexibility that an environment based on the architecture could deliver. The window hierarchy can readily be assembled from modular components, and it seems likely that a longer chain of simple (standard) windows will be more flexible and cost-effective than a short chain of more complex windows. This leads to the possibility of a future market for window components, with substantial in-built engineering functionality, based on the concept of libraries plug-in into a standard environment. In such a context, it seems likely that engineering applications software would evolve into smaller functional applets integrated via the data environment.

In reality the preceding is highly speculative as it is almost impossible to predict the direction of software technology. Many other technical solutions are possible, even within the domain of models defined in EXPRESS. For example, ISO TC184/SC4 is currently pursuing a route by which EXPRESS models are encoded as UML classes – thus enabling UML (Booch, 1998) development technologies to be leveraged.

REFERENCES

Booch, G., Jacobson, I. and Rumbaugh, J (1998). The Unified Modelling Language User Guide, Addison Wesley Publishing Co.

Csavajda, P., Haas, W., Kalpakidis, G. (1998). STEP and its implementation in factory design - the STEP-CDS initiative, Proceedings of ECPPM '98.

Crowley A.J. and Watson A.S. (2000). CIMsteel Integration Standards Release 2: Volume 1 – Overview, Steel Construction Institute.

IAI (2000). Welcome to the International IAI web site, http://iaiweb.lbl.gov/

Leeds University (1985). CIS Homepage, http://www.cae.civil.leeds.ac.uk/past/cis/cis.htm

Leeds University (2000). Data Exchange Translators: Commercial Implementations http://www.cae.civil.leeds.ac.uk/information

Owen J. (1997). STEP An Introduction, Information Geometers.

Schenck, D.A. and Wilson, P.R. (1994). Information Modeling the EXPRESS way, Oxford University Press.

STEP (2000). Welcome to ISO TC184/SC4 Developers of International Industrial Data Standards, http://www.nist.gov/sc4.

STEP Tools (1999). ROSE Library reference manual, STEP Tools Inc.