Manipulating IFC model data in conjunction with CAD

M. Nour & K. Beucke
Chair of Construction Informatics, Bauhaus University, Weimar, Germany.

ABSTRACT: The IFC model is used as a means of information exchange between AEC software applications. Currently, workflow aspects based upon IFC models still reside in a gray area between applications. This paper reports ongoing research work on a new approach that uses an online dynamic continually updated construction product data source for IFC based Building Information Models (BIMs). It focuses on different algorithms for parsing, interpreting and writing STEP ISO 10303 P-21 files. In between these stages various instantiation, deletion and updating process on the IFC model take place. The paper also investigates the ability of current software applications to work on the IFC model in a sequential order and points out some workflow management problems that were encountered during this process.

1 INTRODUCTION

The paper focuses on the operations that are needed to carry out on the IFC model in order to introduce product data from an external source to the model. An important objective of the undergoing research work is to supply the IFC model with information about construction products; not only life cycle information but also information allover the lifecycle of the product. Another objective is to enable carrying out operations needed to modify and keep the IFC model up-to-date. Among these operations are the instantiation of new property sets, property definitions, and classifications of construction products. Moreover, it is more often than not required to carry out changes to the model like changing the values of the above-mentioned aspects or even deleting them. This is envisaged to respond to the changes that a construction project undergoes in the design, specification and value engineering stages and hence can be used for procurement aspects like conduction of parametric searches in electronic product catalogs. Furthermore, this is considered to be the means by which the construction product's life cycle properties could be mapped to the IFC model all over the life cycle of the product itself. An important goal in this process is not to cram the IFC model with all the available life cycle information for each product in the construction project. However, the goal is to supply and support the model with up-to-date information from a continually updated data source (Nour et al, 2004). This implies that only needed information is mapped and instantiated in the model. In this way, the size of the IFC model and its exchange format (STEP ISO 10303-P21) can be minimized i.e. a fat free communication model can be reached. This model exists in parallel to its life cycle information and can reference and import any of its contents according to the user's (client's) need.

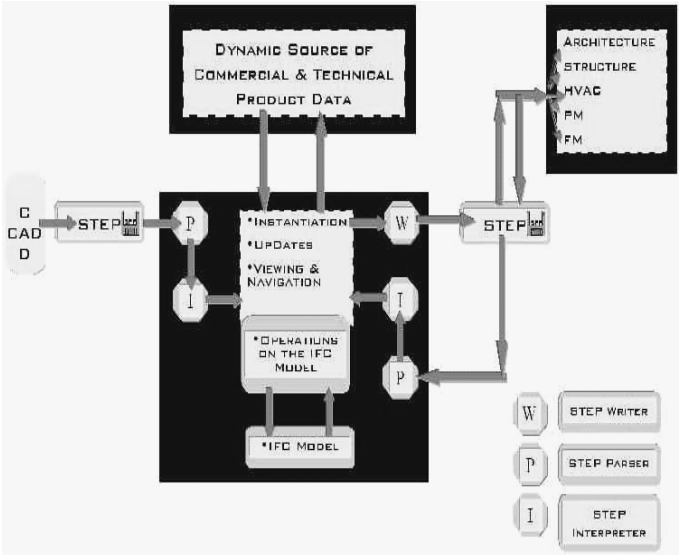

Figure 1.1 An Overall view of the operations on the IFC model

Figure 1.1 shows a map view of this research work, where the whole system consists of distributed platforms represented in a source of dynamic construction product data and a user (client) side represented in the CAD software and other multidisciplinary applications. The core of this system is the operations undergone on the IFC model. This in-

cludes importing the model to a space where new data can be instantiated, and old data can be updated or deleted. At the end the model can be exported to an arbitrary number of multidisciplinary applications. In other words, this means that construction product data can be mapped -from a relational model- to the IFC object oriented model and newly instantiated or merged to it. The latter is done in two main scenarios. First is when the user or specifier needs to define the search parameters of the product and second is later during the whole life cycle of the product whenever a need for product data or updates arises. The paper begins with a brief description of the STEP ISO 10303-P21 parser, then moves to the IFC2x Interpreter, where various alternatives for the interpretation process are discussed. It shows how various modifications and updating operations that are performed on the model in addition to exporting the model in the form of a STEP ISO 10303 P-21 file and finally some workflow management aspects that were encountered.

2 STEP ISO 10303-P21 PARSER

A major problem facing the implementation of IFC in university research projects is the process of parsing STEP files and the instantiation of the IFC Model, which is defined in EXPRESS ISO-10303-P11. In industry contexts, there are several EXPRESS based object oriented databases that are capable of reading, updating, writing and mapping STEP files. However, the costs of such relatively new technologies, at the time of writing this paper, are extremely high.

```
ISO-10303-21;

HEADER;
.....
ENDSEC;

DATA;
.....
ENDSEC;

END-ISO-10303-21;
```

Figure 2.1 Constituents of a STEP file, Nour 2004

For many researchers, IFC is considered to be not more than a means for data exchange between commercial software applications. It was found that one of the biggest barriers standing between researchers and the IFC model is how to push the model itself from the theory in the IAI[1] documentation to the practice of implementation. In the meantime, the tools that can facilitate the instantiation of the model and manipulating its elements are extremely expensive. Since the research work is entirely independent of any commercial software application, it presents therefore a simple approach to parsing (STEP ISO-10303-P21 2004) files using available parser generator technologies.

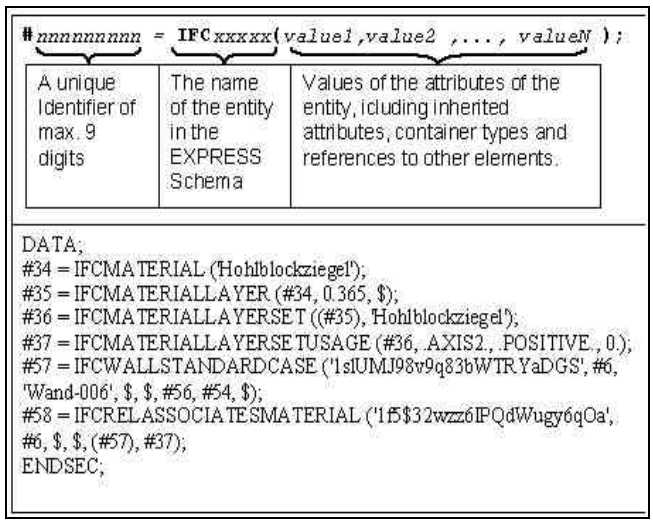

Figure 2.2 The Analsis of the STEP file, Nour 2004

The first step in developing the parser is identifying the structure of the STEP file, then defining the grammar. The first step was determined as a result of the analysis process of the STEP physical file's main constituents that are shown in figure 2.1, starting with the HEAD section and moving to the body or DATA section and ending with the END_ISO_STEP statement. The second step is done by writing down a grammar of the step file. The ifcElement starts with a line number identifier and the '=' sign, the name of the Element 'IFCxxxxx', open bracket, the argument list, a closing bracket, then an EOC (End Of Line Command) symbol which is the ';' symbol as shown in figure 2.2.

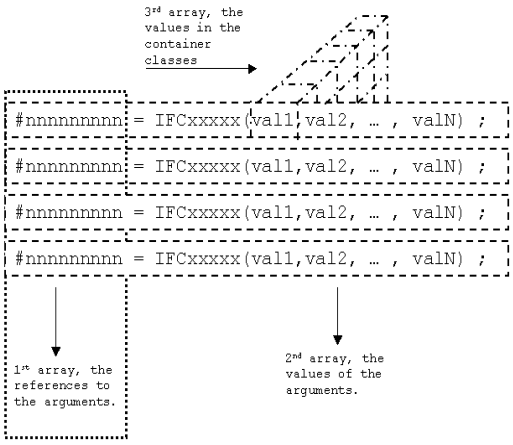

Figure 2.3 The instantiated three arrays from the parsing process

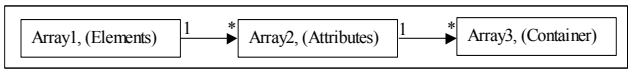

Figure 2.4 UML for the output 3D Array, Nour 2004

At the end of the parsing process a three dimensional array is obtained, as shown in figures 2.3 and 2.4. The first dimension of the array contains all the

1 IAI International Alliance for Interoperability, www.iai-international.org/iai_international/

IfcElements (1st array), each IFC element points to an array containing its arguments (2nd array) and finally some argument values are references to container classes or other elements by them selves, i.e. They are represented through the third optional dimension (3rd array). Now the code should have already been parsed and is ready for interpretation by Java.

3 IFC2X INTERPRETER

The interpreter tries to map the IFC2x model entities to Java classes. The fact that STEP is a kind of serialization of objects defined in EXPRESS cannot be ignored. This more often than not results in many problems like the absence of some attributes in the STEP file, e.g. the optional, derived and inverse EXPRESS attributes. Moreover, Java is a programming and modeling language, whereas EXPRESS IS NOT a programming language. There are lots of differences that can be pointed out between the two languages.

Among these differences are the support for multiple inheritance, different types of container classes, logical, optional and Inverse attributes. STEP physical files are tightly bound to the EXPRESS schema they were written against. Because the ordering of attribute values is determined from the EXPRESS schema, changes to the schema may cause problems with files written against the original version. In the context of this paper only important issues that are encountered through the process of creating the interpreter are very briefly discussed.

A first step in mapping EXPRESS entities to Java classes is building an SDAI (Standard Data Access Interface). The SDAI is a STEP API for EXPRESS defined data. The SDAI protocols contain a description of the operations and functionalities that should be satisfied by the mapped entities. The SDAI is described by several ISO standards documents. STEP Part 22 (ISO 10303-22 SDAI, 1995) contains a functional description of the SDAI operations, while Parts 23 (ISO 10303-P23, 1995) and 24 (ISO 10303-P24, 1995) describe how these operations are made available in the C++ and C language environments. Bindings for CORBA/IDL and Java are also available. As a general rule, all mapped EXPRESS entities should implement the SDAI interface. The only purpose of this interface is the definition of rules that the generated Java classes must implement to get access to the inner attributes (Loffredo, 2004). There are two main types of bindings available:

SDAI Late Binding — In this approach, no generated data structures are used. Only one data structure is used for all of the definitions in an EXPRESS model. The Inner attributes are usually collected in a container class, e.g. Vector or List. Moreover, access to the objects is provided at runtime (ibid).

SDAI Early Binding — An early binding approach makes the EXPRES S information model available as specific programming language data structures for each different definition in the EXPRESS model (Schwarz, 2004). For example, an early binding such as the Java SDAI would contain specific Java classes for each definition in the IFC2x Schema. One major advantage to this approach is that the compiler can do extensive type checking on the application and detect conflicts at compile time. Special semantics or operations can also be captured as operations tied to a particular data structure. Early bindings are usually produced by an EXPRESS compiler. The compiler will parse, resolve, and check the EXPRESS model, then passes control to a code generator to produce data structures for that model. EXPRESS entity definitions are usually converted to Java or C++ classes where type definitions are converted to either classes or typedefs, and the EXPRESS inheritance structure is mapped onto the Java / C++ classes. Each class should have access and update methods for the stored attributes, possibly access methods for simple derived attributes, and constructors to initialize new instances. It should be also noticed that Java does not support multiple inheritance. At any rate, this problem is not encountered in this work due to the fact the EXPRESS definition of the IFC model does not use any multiple inheritance.

Another Approaches — The early and late bindings are not the only possible approaches. In the scope of this work a mixed approach is implemented. This approach provides the advantages of an early binding (compile-time type checking and semantics as functions in a class) without the complexity introduced by modelling a huge number of classes in the IFC model (there are more than three hundred and seventy leaf classes, in addition to eighty nine defined types, twenty three select types and one hundred and seventeen enumerations). It should be mentioned that in the the early binding approach there is a restriction to predefined classes. This means that if we need to interpret Ifc2x compliant STEP files, we have to model all the elements of the IFC2x model to Java classes. At the meantime, if we need to change to IFC2x2, then we have to do the same again with the whole model to produce new Java binding classes. A mixed binding takes advantage of the observation that applications rarely use all of the structures defined by the IFC2x EXPRESS Schema. The subset of structures that are used, called the *working set,* can be early-bound, while the rest of the Schema is late-bound (idem, 2004). All data is still available, but the application development process is simplified. The number of classes and files that are needed are reduced dramatically, resulting in quicker compiles, simpler source control, and more rapid development.

In the scope of this work the mixed approach is implemented, in the early binding parts (working classes) a more labour-intensive approach has been used to hand-generate an early binding for the IFC2x model. Such a binding is not 100% compliant to the IFC EXPRESS model, due to the fact that there are EXPRESS data types that can not be mapped 1:1 to Java data types in addition to the strong rules that are imposed by the EXPRESS language.

Although this approach might provide a simplified programming interface, there are some drawbacks to be aware of. Aside from the increased labour involved in defining and implementing the binding, this method requires that the user should understand the EXPRESS schema API completely, and be able to predict how it will be used. (Loffredo, 2004).

3.1 *Mapping Express Data Types*

The EXPRESS language includes TYPE and ENTITY declarations, CONSTANT declarations, constraint specifications and algorithm descriptions. Only EXPRESS primitive data types, TYPE, ENTITY and aggregations declarations, are mapped to the exchange structure (STEP-P21). Other elements of the language are not mapped to the exchange structure and consequently are not mapped to Java types. Table 3.1 shows the mapping from EXPRESS to STEP and Java types. The first two columns in the table are taken from the ISO 10303-21 Specifications. The third column is developed by the author.

Table 3.1 Mapping EXPRESS to STEP & Java

EXPRESS element	mapped in to STEP-P21:	Mapped into Java
ARRAY	list	List
BAG	list	List
BOOLEAN	boolean	boolean
BINARY	binary	binary
CONSTANT	NO INSTANTIATION	NO INSTANTIATION
DERIVED ATTRIBUTE	NO INSTANTIATION	NO INSTANTIATION
ENTITY	entity instance	Class
ENTITY AS ATTRIBUTE	entity instance name	Reference to object
ENUMERATION	enumeration	Class
FUNCTION	NO INSTANTIATION	NO INSTANTIATION
INTEGER	integer	integer
INVERSE	NO INSTANTIATION	NO INSTANTIATION
LIST	list	LIST
LOGICAL	enumeration	Class
NUMBER	real	double
PROCEDURE	NO INSTANTIATION	NO INSTANTIATION
REAL	real	double
REMARKS	NO Inst.	NO Inst.
RULE	NO INSTANTIATION	NO INSTANTIATION
SCHEMA	NO INSTANTIATION	Package (early binding)
SELECT	As an entity	Class (early binding)
SET	list	Set
STRING	String	String
TYPE	As an entity	Class (early binding)
UNIQUE rules	NO INSTANTIATION	NO INSTANTIATION
WHERE RULES	NO INSTANTIATION	NO INSTANTIATION

The EXPRESS Type, Enumeration and logical values are mapped to Java classes that try to immitate the behaviour of the EXPRESS entities. However, there is a drawback that modelling the rules and ristrictions imposed by the EXPRESS modelling language is not fully achievable.

3.2 *Interpreting STEP ISO 10303 P-21*

After parsing the STEP file, a three dimensional array - described earlier – is obtained. The following algorithm describes the process of interpreting the parsed code to IFC2x Java classes, where a mixed early and late binding approaches are used together. The author did not build an EXPRESS compiler that automatically does the mapping between EXPRESS entities and Java classes but depended on a good understanding of the IFC2x model in manually creating the mapping.

Step One — is the building of Java classes that are mapped from the IFC2x EXPRESS Schemas' entities i.e. an early binding approach. This is done for about more than seventy seven working classes and more than three hundred and twenty abstract and super classes. Each IFC EXPRESS Schema is mapped to a Java package and each mapped class implements the SDAI interface that provides the functionalities that insure reaching the inner attributes of the class.

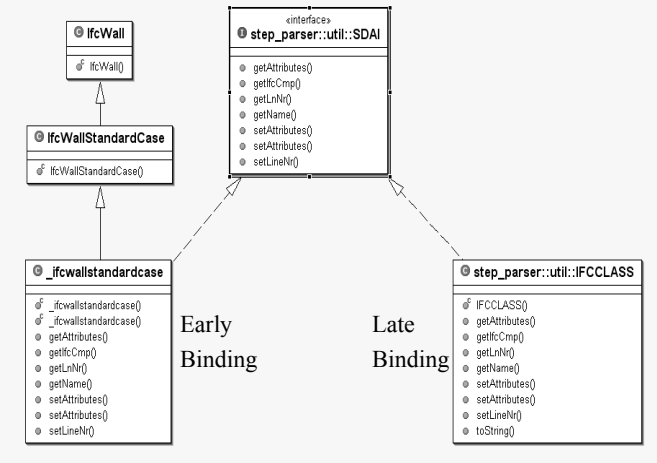

Figure 3.1 UML Diagram showning the implementation of the SDAI interface for early & late bindings and the separation between the IFC model and its implementation

In the early binding approach the EXPRESS entities are mapped to Java classes with no implementation, only as attributes. The implementation is then determined by a subclass that takes the name of the superclass preceded by a "_" as shown in the UML diagram in figure 3.1 This keeps the Java Ifc2x model pure and away from the influence of any implementation. This is envisaged to enable other researchers to use the model and provide their own implementation without any limitation to the author's use of the model. The figure also shows the use of both early and late binding approaches at the same time. Both approaches implement the SDAI interface.

In the late binding approach, one class is used for all EXPRESS entities. This class contains an attribute that is an array that contains all the attributes of the EXPRESS entity. This approach does not perform any attribute type checking on the countrary to the early binding approach that performs a type checking at run time and throws a class cast exception, whenever an incompatibility is encountered.

Step Two — Sorting the parsed array in an ascending order according to each element's identifier number is done by changing the array to an ArrayList and building a comparator class that is capable of sorting the list. The interpreter iterates over the parsed array of elements. If the element iterated upon already exists in the IFC2x model (early binding), then a new instance is created with the given parameters. If not, it will be instantiated as a late binding class. In both cases, before the instantiation takes place, the interpreter iterates over the arguments and makes an argument checking for each element in the 2nd dimension of the array i.e. the attributes of the IFC STEP entity. If it is a "$", then it is substituted by a null value. If it is a "#nn", then the identified element is sought from an identifiers HashMap that keeps references to the interpreted IFC2x Java objects and uses the STEP numerical identifier(#nnn) as a key. If it already exists i.e. already interpreted, then a reference to it replaces the identifier and if not, then it is added to a remainings list, where it will be later referenced to the correct element at the end of the interpretation process. In case where the argument is a container class (a Set or a List and so forth), the interpreter iterates over its elements (in this case as the 3rd dimension of the Array) and treats them as normal arguments. In general, if the argument is not an identifier, a "$" or a container class, then the value of the argument is taken as a parameter for the construction of the new instance of the Ifc2x Java class, bearing in mind the mapping rules between EXPRESS data types and Java data types. At the end of the interpretation process, the elements in the remainings list are re-instantiated, where any identifier reference should be replaced with a reference to an element (IFC2x Java Object) obtained from the identifiers Hash Map. In this way, any violation to referencing conventions is rectified. In other words, the instantiation of such argument is postponed till the end, where the references to the elements already exist in the identifiers HashMap. At the end of the interpretaion process we should already have an array of Ifc2X Java objects.

4 VISUALISATION OF THE IFC MODEL

After interpreting the parsed STEP file it is visualised in the form of a tree that represents the project hierarchy of the IFC model. This enables the navigation through the model and exploring its entities.

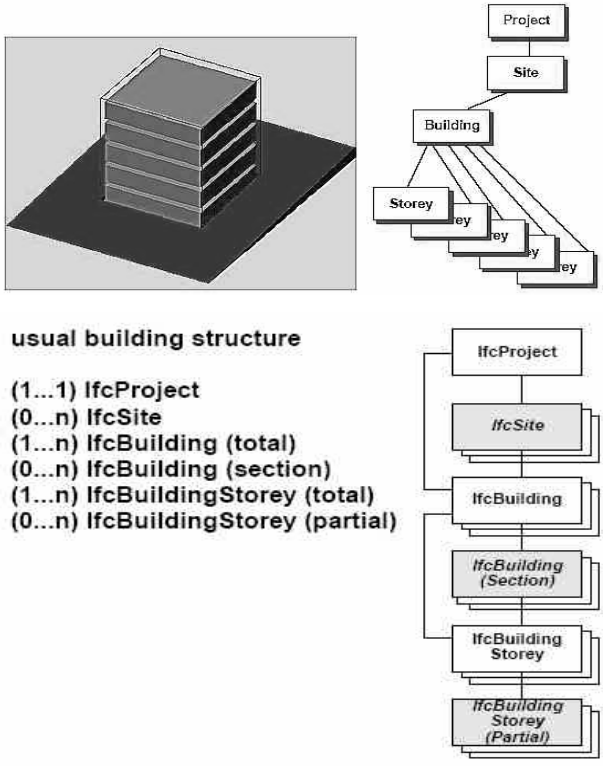

Figure 4.1 The Project hierarchy in the IFC model, IFC2x Model Implementation Guide (IAI, 2002).

This also facilitates the selection of construction products in the IFC model and carrying out various processes on the model such as any instantiation, modification or deletion of the products' properties and so forth.

Figure 4.1 shows the hierarchical structure and spatial arrangement of the IFC model. The root of this tree is the Project entity, where it is a single unique object that contains zero or more sites. Each project contains one or more buildings and each building contains one or more stories. The mandatory and optional levels of such a tree structure are shown in the same figure. Whereas the IfcProject, IfcBuilding and IfcBuildingStorey are mandatory levels for the exchange of complex project data, the IfcSite and IfcSpace represent optional levels (which may be provided, if they contain necessary data).

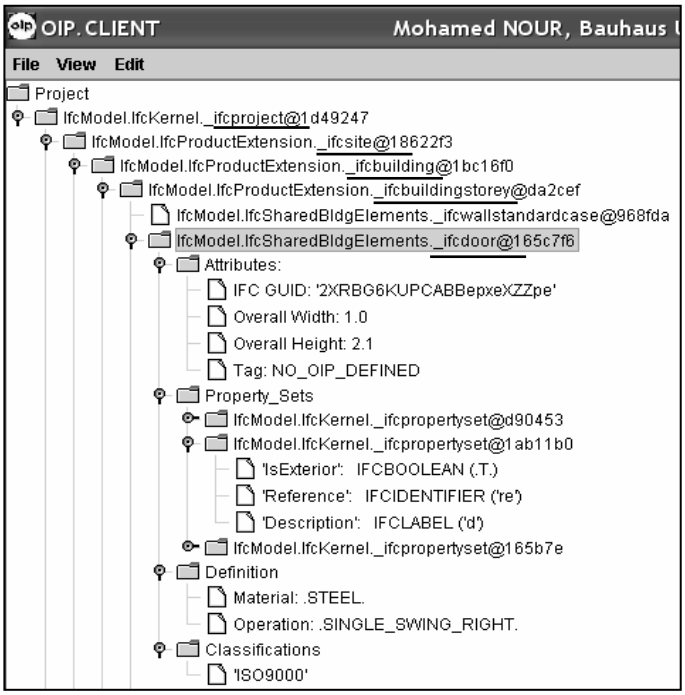

Figure 4.2 The IFC Project Tree hierarchy

Figure 4.2 shows the navigation tree in the software application, starting from the project instance moving down to the construction product instance. The project tree is extended to include other aspects such as the product's attributes like the dimensions of a door or a window, the construction materials included in such elements and so forth. In other words, the attributes that contribute to the construction of parametric searches in electronic product catalogs are included. It could be navigated through these parameters as desired by just expanding the product's tree node. And hence enables carrying out information queries and updates on the selected elements in the information model.

5 OPERATIONS ON THE IFC MODEL

In the scope of the research work the operations needed to be done on the IFC model are limited to the instantiation, updates and deletion of construction product properties in addition to exporting the modified model in the form of a STEP ISO 10303 P-21 file.

5.1 *Instantiation of new property sets and classifications*

In general and as shown in figure 5.1, before carrying out any instantiation process to a property set, property definition, a classification or even a new construction material, we should find first the elements (IfcBuildingElement) in the IFC model to which the new instance is related. In the scope of this research work, this is envisaged to be done by selecting a product from the project's tree view. Once we have a selected product, then we can query the model to find the objectified relationship classes instances (IfcRelxxx) that link the product with other parts of the model – like a cross reference table in a relational database- and consequently add to the

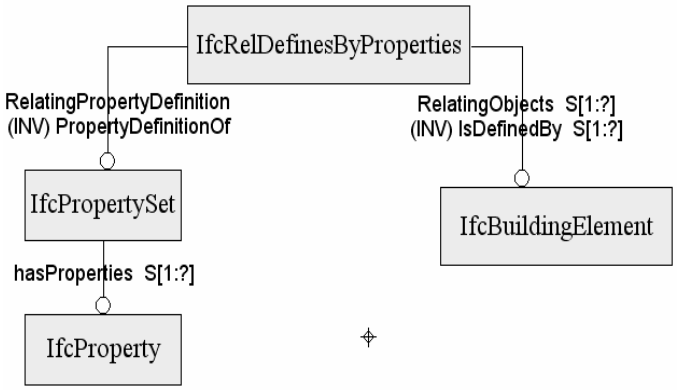

Figure 5.1 An EXPRESS-G diagram showing the relation between properties & construction products through the definition relationship

product an arbitrary number of new properties classifications or materials. Furthermore, this enables carrying out any required update or delete operations on the model.

The instantiation process is done in this way due to the fact that the IfcPropertySet and IfcBuildingElement EXPRESS entities are both linked to the IFC relation by inverse attributes – as shown in figure 5.1 - that are absent in the STEP file. Thus, searching the relations is the only means through which the property set and the construction product could be matched together.

The process of adding a new property to a product begins by searching the model and examining if the product is already linked to a property set that this property belongs to or not e.g. PsetDoorCommon which is one of the many published property sets in the documentation of the IFC2x model. In case this property set is found, the property is simply added to the property set. If the property set does not exist then a new property set and a new relation instance are instantiated from scratch, linked to the product and the property is added to the property set.

In both cases of instantiation of the property sets and the constituent properties, new instances of the properties themselves are created with the necessary parameters, added to the property set and then added at the end of the instantiation process to the array (ArrayList) of IFC elements. It should be mentioned that figure 5.1 is an EXPRESS-G diagram that represents an abstraction by the author that ignores inheritance of the EXPRESS entities and some of the attributes. This is done for clarification reasons and to make the diagram as simple as possible.

5.2 *Updates*

The updates to the model are carried out in the same manner like the instantiation. During the navigation in the project tree, a product is selected, a query in the model is executed to find its attributes and properties. Once the values are changed from the user interface as shown in figure 5.2, the new values replace the old ones.

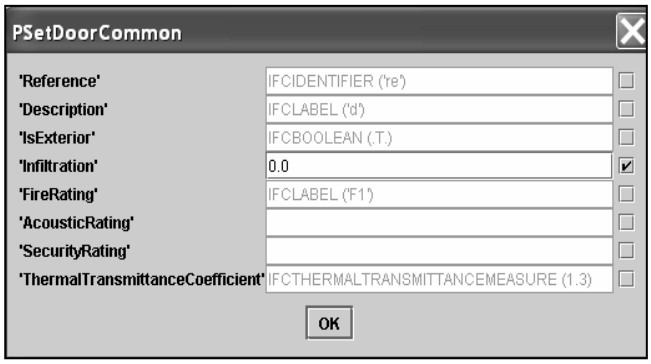

Figure 5.2 The GUI used for the updates

5.3 *Deletion*

Deletion is done either by blanking the text filed in the GUI or deleting the property explicitly. It is important here to mention that it is not only necessary to delete the references to and from the deleted object, but it also important to delete the object itself. This is done by changing the array of elements to an ArrayList, removing the deleted element and then returning the array back again.

5.4 *EXPORTING STEP ISO 10303-P21 files*

Exporting the model in the form of a STEP ISO 1030-P21 is easily done once the model is converted to a tree structure according to its relationship references and not according to its project hierarchy as it was done earlier in the visualization process. To build this tree the Java JTree and DefaultMutableTreeNode classes were used. At the root of the tree are always the aggregation relationships that have references to different parts of the model and acts as the aggregation elements. On the other hand, elements that have no references to other elements are situated at the leaf ends of the tree. The more the element has references, the more it is nearer to the root of the tree.

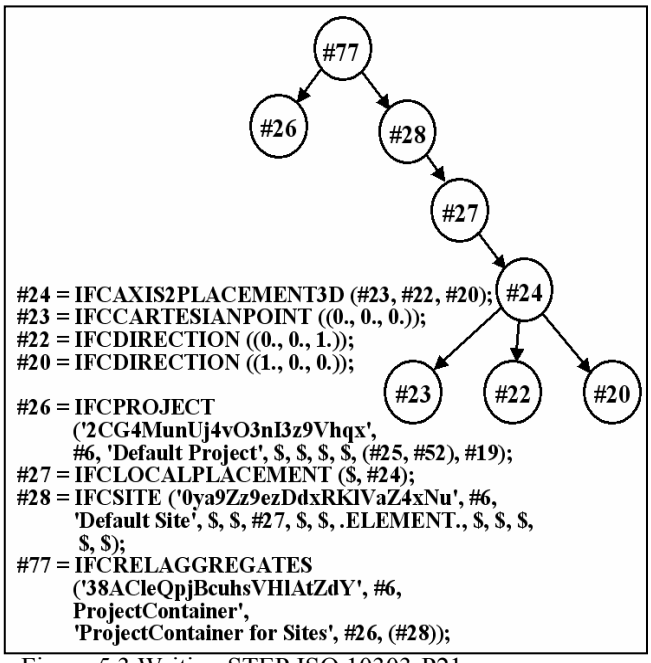

Figure 5.3 Writing STEP ISO 10303-P21

Figure 5.3 presents a simple example that shows the way to build such a tree. The STEP writer iterates over the array of elements and each of the element's arguments, where the Java Types are mapped to STEP. All elements are allocated new identifier numbers according to their position in the tree. From figure 5.3 we can notice that elements with an identifier #xx references another element with an identifier #yy, where xx > yy. The STEP writer iterates on the IFC model elements and their arguments and replaces the null attributes by a "$" and the references to other elements by their newly allocated identifiers. Furthermore, the same procedure is done with elements residing inside collections or container classes together with adding extra parenthesis as shown in #77 and its reference to #28 in figure 5.3.

One major problem during the building of the tree was the mutation of the nodes. In the IFC model tree a node can be referenced from more than one parent node, thus the node jumps from the old position to the new position. However, the nodes should keep their position where they are first referenced, i.e the old position. Hence, the major task to overcome this problem was to prevent the referencing of nodes that have already been referenced before. This was done by keeping a record of the referencing in a hash map and to allow referencing only in cases where the node has never been referenced before.

After building the tree structure, the HEADER part of the STEP file is instantiated, and the tree is traversed in a post order recurring manner, where the leafs of the tree are iterated upon before the parents and hence given a smaller identifier number and written first to the STEP file.

6 WORKFLOW MANAGEMENT ASPECTS

The IFC model is mostly used for the transfer of information from one software to another. There is always a gray area between software applications that enables the information to be mapped from one software application to another. However, there are more often than not functionalities that are supported by one software and not by the others. This often results in inevitable information loss; when the model is saved by an application that imports an IFC model and does not support the functionalities of the software that originally produced it. Further more, some times it has nothing to do with functionalities, the software serializes the model to IFC according to the information content of its objects and ignores the information that was originally imported within the model. This is exactly the case that the author encountered when the IFC model was instantiated by product data and re-imported by CAD software (ArchiCAD 7.0, Students version and ADT 3.3) the software could show the newly instantiated information as in figure 6.1, but when asked to re-export the model, the software serialized its objects to a STEP file and took no care of the extra information in the model. Hence, the IFC model loosed its advantage as an independent non proprietary building information model that is capable of transmitting multidisciplinary AEC information.

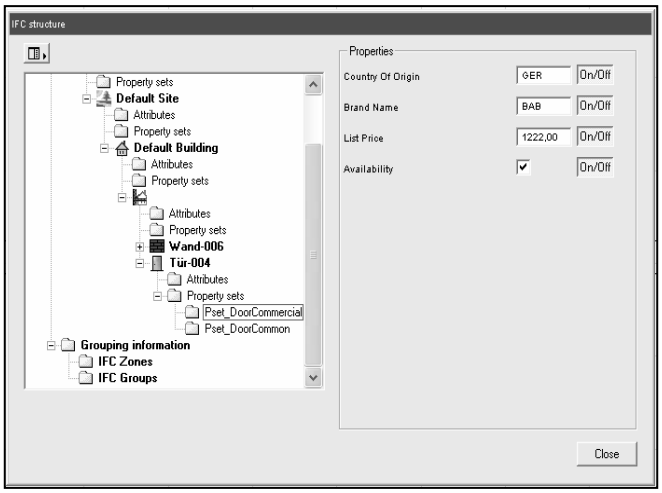

Figure 6.1 A snap shot from ArchiCAD 7.0 showing the Instantiation of the Property Set Pset_DoorCommercial

7 CONCLUSION

The paper has discussed various steps necessary for the implementation of the IFC model in scientific research using simple programming techniques and away from using relatively expensive commercial software. The paper showed an algorithm for parsing STEP ISO 10303-P21 files and its interpretation to IFC2x Java classes, in addition to carrying out instantiation, updates and deletion operations on the model and finally exporting the model in the form of a STEP physical file that can be used by AEC software applications.

Some shortcomings of the CAD software applications were encountered in the process of exporting the imported model. The shortcomings are represented in the loss of information that was originally imported into the model, despite the fact that the CAD software could read it and represent it in its own environment. It is worth also mentioning that the lost information does not relate to the functionalities of the importing CAD software. It is most probably commercial information that is related to construction products. In other words, only the information that is related to the imported software is exported.

If the above mentioned shortcoming is rectified by AEC software applications - which will happen no doubt sooner or later – this would give a real technical push to the use of Building Information Models and its ability to be worked upon by multidisciplinary AEC software applications.

REFERENCES

Loffredo, D. T. 1998. Efficient Database Implementation of EXPRESS Information Models, PhD Thesis, Rensselaer Polytechnic Institute Troy, New York, USA.

IAI 2002, IFC2x Model Implementation Guide, V 1.4, Liebig, T.(ed)International Alliance for Interoperability.

ISO 10303-11 EXPRESS 1994. Industrial automation systems and integration – Product data representation and exchange – part 11: Description methods: The EXPRESS language reference manual.

ISO 10303-22 SDAI, 1995 Industrial Automation Systems and Integration — Product Data Representation and Exchange — Part 22: STEP Data Access Interface, ISO Document TC184/SC4 WG7 N392, July 1995.

ISO 10303-23 C++ language binding 1995. Industrial Automation Systems and Integration — Product Data Representation and Exchange — Part 23: C++ Language Binding to the Standard Data Access Interface Specification, ISO Document TC184/SC4 WG7 N393, July 1995.

ISO 10303-24 c language late binding 1995. Industrial Automation Systems and Integration — Product Data Representation and Exchange — Part 24: Standard Data Access Interface — C Language Late Binding, ISO Document TC184/SC4 WG7 N394, July 1995.

ISO 10303- 21 STEP 2002. Industrial automation systems and integration – Product data representation and exchange – part 21: Implementation methods : Clear text encoding for exchange structure.

Nour, M. M. 2004. A STEP ISO-10303 Parser, 16th Forum Bauinformatik . In Jan Zimmermann, Sebastian Geller (eds), Braunschweig. Shaker Verlag, Aachen. October 2004, pp. 231 – 237. ISBN 3-8322-3233-4

Nour, M. & Beucke K. 2004 IFC supported distributed, dynamic & extensible construction products information models. In: Dikbas A. and Scherer R. (eds) : Proceedings of ECPPM 2004 – e-Work and e-Business in Architecture Engineering and Construction.8-10 Sept. Istanbul, Turkey. 2004 Taylor & Francis Group, London, ISBN 04 1535 9384

Schwarz, S. 2004. Java as a language for STEP-based product and process modeling, University of the Federal Armed Forces Munich, Germany.