ITcon
www.itcon.org - Journal of Information Technology in Construction - ISSN 1874-4753

PERFORMANCE OF DIFFERENT (BIM/IFC) EXCHANGE FORMATS WITHIN A PRIVATE COLLABORATIVE WORKSPACE FOR COLLABORATIVE WORK

PUBLISHED: October 2009 at http://www.itcon.org/2009/48
EDITORS: Rezgui Y, Zarli A, Hopfe C J

Mohamed Nour, Dr.,
Bauhaus-Universität Weimar, Germany;
mohamed.nour@daad-alumni.de

SUMMARY: *This paper reports on research work done to incorporate an end user private workspace for each stakeholder in a collaborative BIM (Building Information Modelling) environment using a central model server. It focuses on reducing the complexity and improving the performance of obtaining IFC early binding runtime objects from stored persistent models. The paper discusses and describes the various types of object-relational mappings between the IFC express schemata and relational databases. The paper also presents the results of examining the possible serialization and de-serialization of IFC objects. An IFC ISO 10303 STEP-21, IFC XML 2X3 parsers and interpreters in addition to STEP-21 writer were developed by the author to be able to conduct the analysis in this paper. Comparative results and developed techniques to reach better performance are described. In order to inform the reader about the context of this research work and the underlying motivation, the paper presents the main interoperability aims and the rationale behind this work.*

KEYWORDS: *BIM, IFC, Model Server.*

REFERENCE: *Nour M (2009) Performance of different (BIM/IFC) exchange formats within private collaborative workspace for collaborative work, Special Issue Building Information Modeling Applications, Challenges and Future Directions, Journal of Information Technology in Construction (ITcon), Vol. 14, pg. 736-752, http://www.itcon.org/2009/48*

1. INTRODUCTION

A lot of efforts in the past decade have been directed towards using a central repository or a model server that acts as a base for interoperability between various AEC-FM disciplines and their software applications (IMS, 2002, Eurostep Share a Space, 2009, EDM Model Server, 2009). In the meantime, collaborative parallel working and change management necessitate the use of Object Versioning to enable the management of a central data repository that supports long term transactions.

Thus, this research work is done as a part of the ongoing research work within the scope of the “InPro” project financed through the 6th EU Framework Program for Research and Development (www.inpro-project.eu). The research project addresses the problem of early design management and collaborative work, where the IFC model (ISO/PAS 16739, 2005) is used by a common model server for supporting the exchange of BIM (Building Information Model) related data.

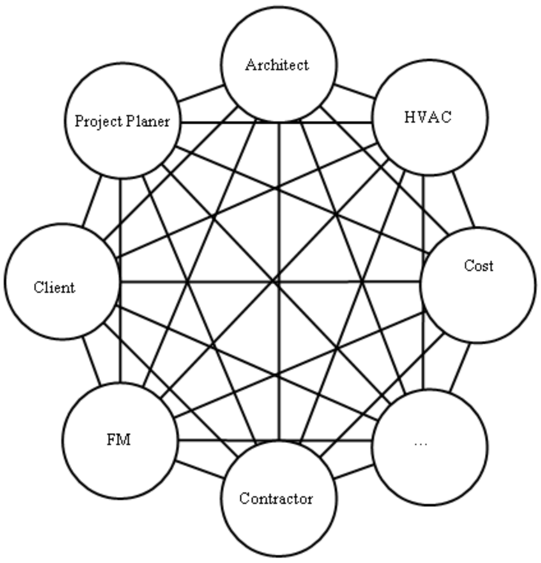


FIG. 1: The current proprietary file based exchange of data in the AEC-FM industry.

The main efforts are directed towards abandoning the proprietary file based data exchange between stakeholders as shown in *FIG. 1* to the more interoperable approach in *FIG. 2*. Meanwhile, in the BIM based approach as shown in *FIG. 2*, at the client sides, where the main stakeholders (e.g. Architects, project planners, cost estimators, HVAC, etc.) are situated, each stakeholder is allowed to optionally use a private workbench. This workbench acts as a local domain and as a sandbox, where team members inside the same organization can exchange their own local private domain data among themselves. At certain development stages of the design, a release version can be uploaded to the central project's server in a "Commit" transaction to be communicated to other domains. The use of the private workbench enables the stakeholders to keep their unshared information and local versions of the design within the boundaries of their organizations and enables them to use any type of software or developing platform.

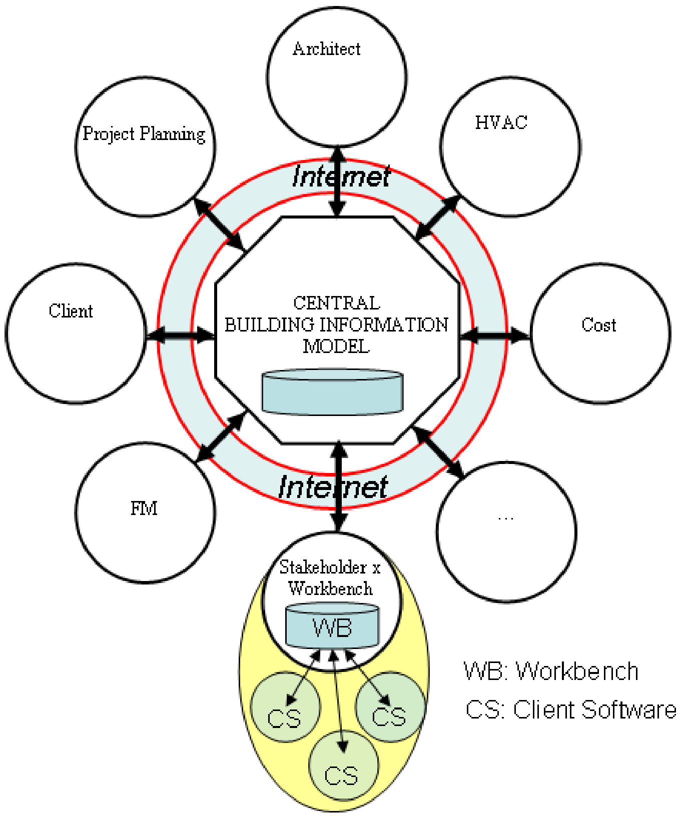


FIG. 2: The main architecture of the solution approach

The main idea behind the workbench concept is to be able to exchange shared data with the central data repository (model server) within an object versioning environment, where there is a need to manage different design versions that are produced during the design development process.

Before describing any of the solution approaches to the problem, it is worth mentioning that the main objectives of the solution approach are:

- To enable conduction of queries on the IFC model data.
- To be able to shift and navigate between object versions in relation to various (project) model versions.
- To be able to create valid partial IFC/STEP models as a result of conducting queries in the local workbench's database.

FIG. 2 shows the main architecture of the suggested solution approach. A main model server acts as a data hub in the middle, where all stakeholders can exchange information through it. This necessitates that the model server should be able to exchange data with all AEC-FM domains that integrate with it through a plug and play scenario. Thus, there is a need to follow certain protocols for the data exchange, especially when considering versioning management.

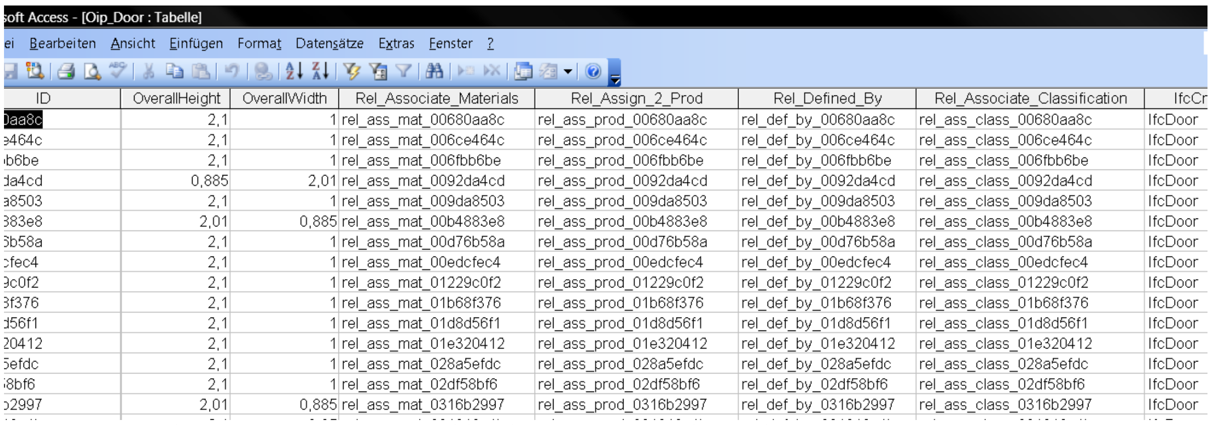
It is difficult to force all software vendors to adhere to a certain exchange protocol. Hence, the workbench approach would offer a good opportunity in organizing the relation between the various stakeholders and their software tools on one side and the central model server on the other side. In addition, it helps to regulate internal workflows, approvals and data privacy issues.

The following sections describe the efforts done to establish a client (stakeholder) workbench that should be capable of both communicating with the central model server and serving the local collaboration needs and data privacy for each stakeholder's organization.

2. STATEMENT OF PROBLEM

Several solution approaches were investigated to satisfy the objectives mentioned in the introduction section of this paper. The main two approaches that achieved results are discussed within this paper. They are the mapping of the entire IFC model to a relational database at the client's (Stakeholder) side and the second is the filtering of the imported IFC model data against domain criteria to create "Domain Objects" or "Business Object".

The first problem that faced the research work was dealing with the IFC/STEP (ISO 10303-21, 1994) exchange format and its EXPRESS (ISO 10303-11, 1994) definitions. The EXPRESS definition of the IFC2X-3 model had to be bound to a programming language. This was done through creating an early binding library of classes to the Java programming language. This was achieved by using the Java Compiler Compiler technology (JavaCC, 2005) for creating an AST (Abstract Syntax Tree) of the EXPRESS-Schema and then generating the corresponding Java classes by traversing the Tree.


soft Access - [Oip_Door : Tabelle]
ei Bearbeiten Ansicht Einfügen Format Datensätze Extras Fenster ?

ID	OverallHeight	OverallWidth	Rel_Associate_Materials	Rel_Assign_2_Prod	Rel_Defined_By	Rel_Associate_Classification	IfcCr
Daa8c	2,1	1	rel_ass_mat_00680aa8c	rel_ass_prod_00680aa8c	rel_def_by_00680aa8c	rel_ass_class_00680aa8c	IfcDoor
e464c	2,1	1	rel_ass_mat_006ce464c	rel_ass_prod_006ce464c	rel_def_by_006ce464c	rel_ass_class_006ce464c	IfcDoor
b6be	2,1	1	rel_ass_mat_006fbb6be	rel_ass_prod_006fbb6be	rel_def_by_006fbb6be	rel_ass_class_006fbb6be	IfcDoor
da4cd	0,885	2,01	rel_ass_mat_0092da4cd	rel_ass_prod_0092da4cd	rel_def_by_0092da4cd	rel_ass_class_0092da4cd	IfcDoor
a8503	2,1	1	rel_ass_mat_009da8503	rel_ass_prod_009da8503	rel_def_by_009da8503	rel_ass_class_009da8503	IfcDoor
383e8	2,01	0,885	rel_ass_mat_00b4883e8	rel_ass_prod_00b4883e8	rel_def_by_00b4883e8	rel_ass_class_00b4883e8	IfcDoor
6b58a	2,1	1	rel_ass_mat_00d76b58a	rel_ass_prod_00d76b58a	rel_def_by_00d76b58a	rel_ass_class_00d76b58a	IfcDoor
cfec4	2,1	1	rel_ass_mat_00edcfec4	rel_ass_prod_00edcfec4	rel_def_by_00edcfec4	rel_ass_class_00edcfec4	IfcDoor
9c0f2	2,1	1	rel_ass_mat_01229c0f2	rel_ass_prod_01229c0f2	rel_def_by_01229c0f2	rel_ass_class_01229c0f2	IfcDoor
3f376	2,1	1	rel_ass_mat_01b68f376	rel_ass_prod_01b68f376	rel_def_by_01b68f376	rel_ass_class_01b68f376	IfcDoor
d56f1	2,1	1	rel_ass_mat_01d8d56f1	rel_ass_prod_01d8d56f1	rel_def_by_01d8d56f1	rel_ass_class_01d8d56f1	IfcDoor
20412	2,1	1	rel_ass_mat_01e320412	rel_ass_prod_01e320412	rel_def_by_01e320412	rel_ass_class_01e320412	IfcDoor
5efdc	2,1	1	rel_ass_mat_028a5efdc	rel_ass_prod_028a5efdc	rel_def_by_028a5efdc	rel_ass_class_028a5efdc	IfcDoor
8bf6	2,1	1	rel_ass_mat_02df58bf6	rel_ass_prod_02df58bf6	rel_def_by_02df58bf6	rel_ass_class_02df58bf6	IfcDoor
2997	2,01	0,885	rel_ass_mat_0316b2997	rel_ass_prod_0316b2997	rel_def_by_0316b2997	rel_ass_class_0316b2997	IfcDoor

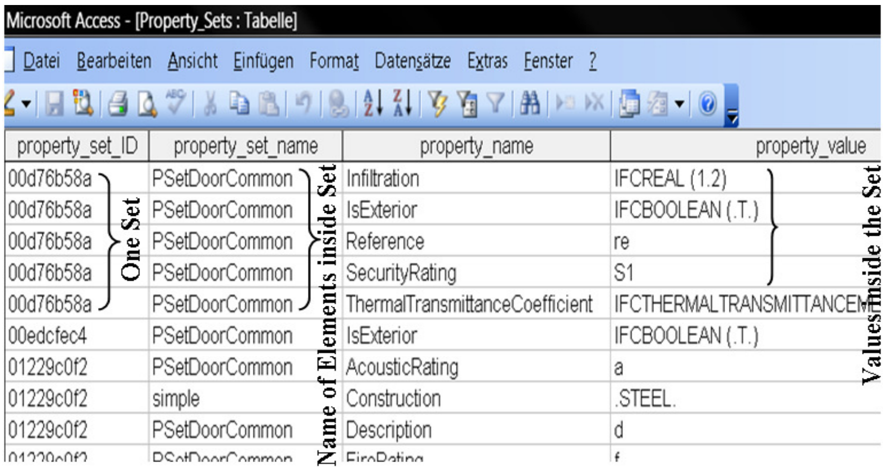
FIG. 3: A snapshot of a cross reference tables that maps IFC Relationship intacnes

The second step was to parse an IFC STEP-P21 file and to interpret it to the corresponding instances from the Java early binding library. This was also achieved using the Java Compiler technology and the Java reflection package (McClusky, 1998).

The third step was to create a connection with a relational database – MS Access - (Microsoft, 2007) using the JDBC ODBC interface and to map the IFC entities to it as shown in *FIG. 3* and *FIG. 4*.

In all the above steps the Java programming language was used due to its compatibility with other software tools developed by the author for visualization of the IFC model in addition to the platform independence of the language. Similarly, other programming languages like C# could have been used for more convenience with particular environments and operating systems.

The IfcRelationships were mapped to the database as cross reference tables. New identifiers were created for managing the EXPRESS-P11 aggregates (List, Set, Bag, Array) and IFC elements that do not descend from the abstract entity IfcRoot as shown in *FIG. 4*. As soon as the first few trials took place several problems were discovered; among theses problems were the management of (GUIDs) Global Unique Identifiers in the Object Versioning Processes, especially with regards to GUIDs of the IFC Relationships which are not preserved within the majority of main stream applications. However, the most outstanding problem was a deficiency in performance of the database, especially with models that are greater than 2MB. It took a lot of time to carryout a simple query that would go beyond the patience of the user.


Microsoft Access - [Property_Sets : Tabelle]
Datei Bearbeiten Ansicht Einfügen Format Datensätze Extras Fenster ?
One Set
Name of Elements inside Set
Values inside the Set

property_set_ID	property_set_name	property_name	property_value
00d76b58a	PSetDoorCommon	Infiltration	IFCREAL (1.2)
00d76b58a	PSetDoorCommon	IsExterior	IFCBOOLEAN (.T.)
00d76b58a	PSetDoorCommon	Reference	re
00d76b58a	PSetDoorCommon	SecurityRating	S1
00d76b58a	PSetDoorCommon	ThermalTransmittanceCoefficient	IFCTHERMALTRANSMITTANCEM
00edcfec4	PSetDoorCommon	IsExterior	IFCBOOLEAN (.T.)
01229c0f2	PSetDoorCommon	AcousticRating	a
01229c0f2	simple	Construction	.STEEL.
01229c0f2	PSetDoorCommon	Description	d
01229c0f2	PSetDoorCommon	FireRating	f

FIG. 4: Modelling EXPRESS aggregate types in the relational database

TABLE 1 shows the statistical results from observing the type of data that filled up the database. Three different models representing different sizes (from 7KB to 8MB) were mapped to the database. It was found that the IFC elements that descend from IfcRoot, which is the common super type of all IFC entities other than those defined in the IFC Resource schema are ranging from 0.3% to 11.4% with an average of 5.4% only. The average of the IfcBuildingElements is not more than 0.91%. The average of the relationships elements descending from IfcRelatioship was found to be 2.78%. The rest is data coming from the resources layer of the IFC model e.g. geometrical data about products' representations, orientation data about the location of elements within the coordinate system and data that define the context of the IFC model (e.g. units of measurement, tolerance values, true north… etc.). It was also found that the ratio between the IfcElements and the geometrical resources describing the representation of these elements is dependent on the type of geometrical representation. For example, in models 2 and 3 the Breps descriptions used a lot of IfcCartesianPoints, IfcPolylines, IfcFaces more than the swept area solids. Thus the percentages of IfcCartesianPoint in models 2 and 3 were found to reach 32% and 25% of the entire elements of the IFC model respectively.

TABLE 1: Statistical analysis of the IFC model

IFC Element Type	Model1 7 KB	Model 2 8 MB	Model3 161 KB	Average
	357 elements	589464 elements	13046 elements	
IfcRoot	11.4 %	4.6 %	0.3 %	5.4%
IfcRelationship	5.7%	2.5 %	0.14 %	2.78%
IfcBuildingElement	< 1%	1.7 %	0.03 %	0,91%
IfcDirection	7%	3.2 %	0.16 %	3,45%

IfcCartesian Point	13%	32 %	25 %	23,3%
IfcAxis2Placement3D	7%	4.8 %	0.08 %	4%
IfcSIUnit	8.5%	0.001 %	0.3 %	2,9%

The analysis of the statistical results in *TABLE 1* has drawn the attention to very important questions.

1. Do we really need to map all the geometrical and context information to the database (ranging 30: 60 % of the IFC model, depending on the types of geometrical description)?
2. Do we really need to map all the IfcRelationships (about 3 %)?
3. Do we really need to map the objectified relationship classes (IfcRelationships) into cross reference tables in the relational database; the matter that makes the structure too complex and therefore causes much difficulty in formulating SQL queries?
4. Does the database structure need to be complicated in order to accommodate the entire IFC model?

Although the above mentioned ratios can change when dealing with different sizes and complexities of IFC models, they are still an indication for the questions to be asked and answered.

The above mentioned questions were answered by the underlying business processes (Stakeholders) within the InPro project. They have shown that a relatively smaller subset of the data is sufficient to enable their processes to take place. This subset is known as the domain model subset. It contains a set of “domain objects” or “business objects” that satisfy the business needs. Business Objects represent any object related to the domain for which the developer is creating business logic. It is an abstraction that contains attributes, values and metadata related to the underlying business process. The term “business objects” or “domain objects” is generally used to distinguish between objects the developer is creating or using related to the domain and all other types of objects he may be working with (IBM, 2008).

In the meantime, irrelevant data is considered to be a heavy burden for the processes. Furthermore, the business process only needs accurate simple information that is easy to reach and that can be trusted.

The rather complex database structure that is designed to map the IFC/STEP-21 model 1:1 does not add any value for the business process needs. They need to be able to conduct simple queries that deliver the required information away from the complexity of the IFC model and its EXPRESS definition.

Moreover, the business processes need to conduct the queries and communicate the results as partial IFC models with the outer world. Thus, there is a need to interpret the results of the queries to partial IFC models that can be compared and integrated with other models. Consequently, there is a need for a two way communication of the IFC model. Hence, the database has to include information that enables the creation of valid IFC partial models when needed.

The process modelling tasks within the InPro project have delivered a description of their business objects. They are flat business objects where their attributes contain data and not references to other objects or any inheritance hierarchy. These attributes were used to create a simple and flat relational database schema. This schema satisfies both the query and versioning needs for each stakeholder’s domain.

The main problem is now how to manage the database efficiently in parallel with the different versions of the IFC models or partial models. There is a need to be able to get the right information from the database in the form of (GUIDs) as a Result Set (Sun, 2008) and load the relevant IFC model and create a partial model that contains the elements of the Result Set in a valid IFC STEP P-21 file. This partial model can be further communicated with other team members or with the main repository for the use by other stakeholders.

Consequently, the main focus of this paper is on how to create runtime objects as fast and as efficient as possible from the stored IFC STEP-P21 files.

3. SOLUTION APPROACH

The main solution approach is described in *FIG. 6* and *FIG. 5*. It depends on four layers architecture for managing the client’s (Stakeholder’s) workspace. It is worth mentioning that the client can optionally use the workbench or can use the direct communication with the model server depending on the type of software and its partial model exchange capabilities. Thus, there are two possible configurations for the client, either by direct communication to the model server as shown by application "B" in *FIG. 6* or by the use of the workbench functionality as shown in application "A" in *FIG. 6*.

3.1 Application Layer

At the application layer as shown in *FIG. 6*. The user is free to use any type of software that has an IFC API or where the output data can be converted to IFC (e.g. text). In cases, when processes do not include a round trip journey of data, then there is no need to have the output exported to IFC.

3.2 Optional Workbench Layer

At the workbench layer a simple and flat database structure is created and hosts the attributes of the domain objects that are necessary for the stakeholder's business processes in addition to references to any type of proprietary file formats.

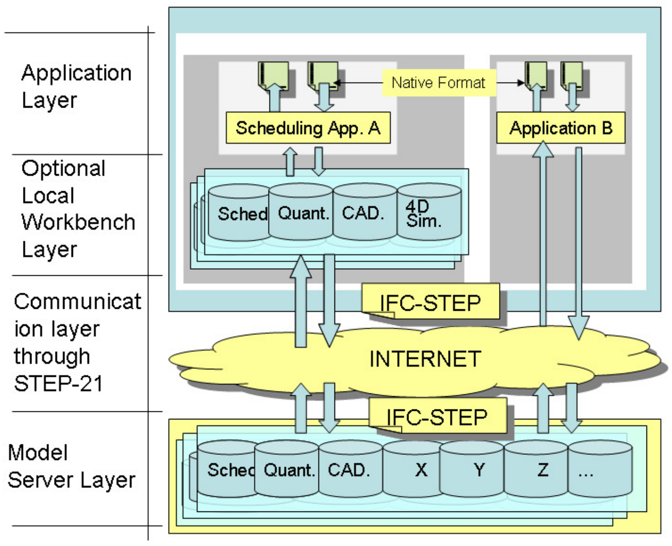


FIG. 6: The overall architecture of the developed system

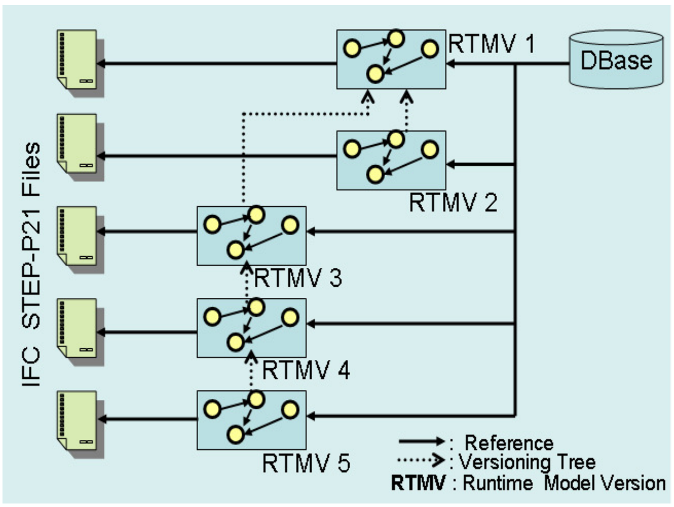


FIG. 5: The communication between the versioning database and STEP-P21 files

The workbench enables collaborative teamwork to take place inside each stakeholder's organization. All internal workings and intermediate stages can be versioned and stored in the database. The Object Versioning Management system enables the saving of the differences (deltas) between the versions of the domain objects and thus succeeds in saving a lot of space in the database, which do impact the database performance. Moreover, the comparison between the Object Versions or different states or variants of the model in relation to the domain aspects can be easily achieved. In the meantime, any information that is not included in the database, due to not being a part of the domain objects attributes can still be obtained through the reference to the relevant version of the saved IFC STEP file. Selected information can be uploaded to the central model server at certain development stages of the work (i.e. Synchronization to model server).

If any design changes take place, the influences on the domain's processes can be easily pointed out by the database due its ability to compare the domain objects' attributes of several design versions.

The workbench can also control the communication with the central model server through several protocols that map the different model versions on the model server to their counterparts on the local workbench. Therefore, some unimplemented IFC concepts like the IfcOwnerHistory and Ownership concepts can be implemented.

Another important aspect for the majority of the stakeholders is data security and privacy. The workbench enables each organization to determine the shared and unshared data subsets of information and consequently gives the organization the possibility of hiding its own knowhow and confidential details.

The Object Versioning Management System enables the tracing of design development together with contexts of changes and hence offers a good chance for organizational learning from previous projects.

FIG. 5 shows the internal structure of the local workbench layer. It consists of a relational database that depends on a flat schema representing the business objects. The main aim of the database is to enable the client to carryout queries that serve the domain needs in addition to the object versioning requirements.

Each design model version in the database has a reference to a URL where the corresponding IFC STEP ISO 10303-P21 model is stored. Whenever needed the IFC file can be parsed, interpreted and changed to a Java (RTMV) Runtime Model Version.

Each model version contains a set of objects that enable versioning at the object level. *FIG. 5* shows an example of a versioning tree, where design variants can be represented as a branching in the tree. Both RTMV2 and RTMV3 are variants of the same parent model RTMV1.

This structure of the local workbench enables domain specific queries to take place with minimum memory needs and does not affect the database performance. If the need for any extra information that goes beyond the content of the abstracted domain model in the database arises, the corresponding IFC model version can be immediately parsed and interpreted to Java IFC early binding objects at runtime. Hence, the main challenge for this database functionality is to create a Java runtime object oriented model of the saved IFC STEP ISO 10303-P21 file to:

1- Satisfy the query needs.

2- Create valid IFC partial models that represent the Result Set obtained from the relational database.

3.3 Communication layer

The communication between the client's workbench and the central model server takes place over the internet in the form of STEP ISO 10303-P21 files as shown in *FIG. 2* and *FIG. 6*. Thus, the ability to split models (create partial models), merge (update), and compare models is essential at both the workbench side and the central model server's side.

There is also a need to establish data exchange protocols between the workbench and the central model server, especially in relation to the management of model versioning.

3.4 Model Server Layer

The model server layer is expected to offer the same functionalities of the workbench, but with more capacity to handle bigger sizes of data sets and multiple domains. This should be based entirely on the IFC data structure and the underlying STEP technology. Thus, an EXPRESS based database server would be recommended as a central data hub that manages the communication between all stakeholders.

4. COMPARING DIFFERENT BIM EXCHANGE FORMATS

As stated in the previous sections one of the main challenges facing the local workbench layer (shown in *FIG. 6* and *FIG. 5*) and the main focus of this paper is on how to create IFC early binding Java objects as fast and as efficient as possible from IFC persistent data (stored files). Therefore, various types of serialization and de-serialization of the IFC model were tested. Among these types are the IFC STEP ISO 10303-P21 relevant to the IFC2X3 schema (text), IFCxml (IAI, 2007) version 2X3 (text) and the Java Object serialization (binary format).

The main criteria for measuring the test results are the size of files and the speed of de-serialization and obtaining runtime objects.

4.1 Testing Environment

IFC files conforming to the IFC2X3 schema version were used. All comparisons were done on a single computer (Operating System: Windows Vista, Intel Centrino Duo processor 1.7 GHz, 2GB RAM) to avoid relative differences. The virus scanner and the auto defragmentation functionalities were turned off to minimize any external influence on performance. Moreover, tests were repeated one hundred times and the average values were taken to neutralize any external influences that might be caused by other processes running on the computer at the same time or the extra overhead caused by initiating the JVM (Java Virtual Machine) at the first runs. The effect of multiple threading programming was also investigated and found to be insignificant for standalone tests.

The software used is ArchiCAD version 11 from Graphisoft. A single software package has been intentionally used to avoid relative differences of IFC output from different software sources, as it is well known that IFC files from various commercial software vendors are subject to size optimizations that can reach 66.7 % reduction in size in some cases (Pazlar & Turk, 2006).

A software tool for parsing and interpreting the IFCxml 2X3 models to Java early binding classes has been written by the author based on the Java DOM (Document Object Model) technology. The tool is also capable of converting the IFCxml 2X3 model to valid IFC STEP-P21 files. The effect of conversion results in a reduction of size in comparison to the file produced from the same application that produced the XML file.

The tests show first a comparison in size between the Native ArchiCAD 11, IFC STEP-21, parsed and interpreted IFC early binding Java, parsed IFC STEP-21 (un-interpreted) and the IFCxml files of the same IFC 2X3 model.

The Java early binding of the IFC model takes place in two phases: 1) Parsing. 2) Interpretation. Therefore, a Java object model is serialized after parsing without interpretation (without binding) and is referred to as "Un-interpreted Java". In the meantime, the same model is serialized after interpretation and is bound to pre-defined library of IFC Java classes (i.e. early binding) and is referred to as "Interpreted Java".

Each file format was compressed using the WinZip functionality except for the object serialization for the interpreted and un-interpreted Java models. Both of them were compressed using the gZip Java functionality which is most appropriate for compressing and decompressing data at run time (on the fly) (Mahmoud, 2002). The gZip compression originally comes from the UNIX world. It compresses files but does not archive them. There is no need to archive files in the serialization of IFC Java model as it consists of one main object (IFC model). Thus, the gZip was found to be more appropriate to be implemented on a single stream of data.

4.2 IFC Testing Models

The test cases include a wide spectrum of complexity of IFC models ranging from a single wall to models of real projects' degree of complexity.

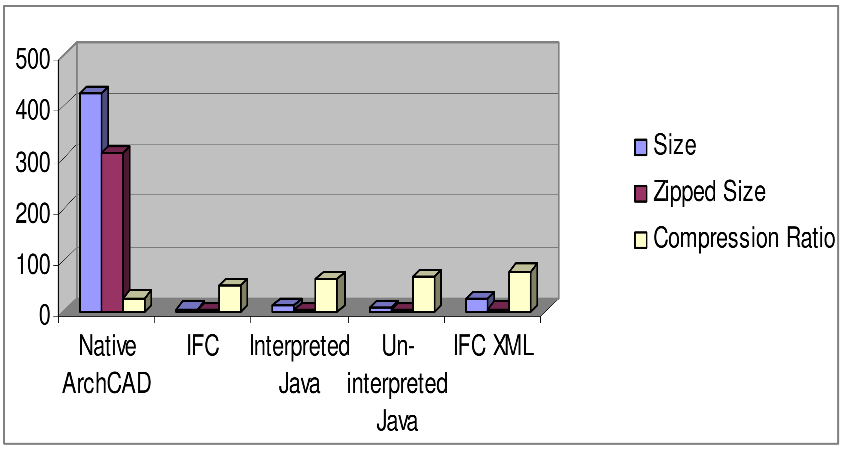


FIG. 7: Test Case 1: Size comparison of different exchange formats

It has also been taken into account that the structure of the IFC model could influence the results. Therefore, the test cases include cases where the geometrical description of IFC elements depends on Brep (Boundary Representation) and other models that depend on Swept Area Solids, CSG (Construction Solid Geometry) and Boolean operations.

The size of testing files ranges from 6 KB for the simplest model to 18 MB for the most complex one.

4.3 Test Case 1: A single IfcWallStandardCase

Test case 1 is the simplest test case. It is created by the author using ArchiCAD 11. The sizes of the native ArchiCAD file, the IFC STEP-P21 file, the serialized interpreted Java Object Model, the serialized un-interpreted Java Object Model and the IFCxml model with their zipped sizes and compression ratios are shown in *TABLE 2* and compared in *FIG. 7*.

TABLE 2: Test Case 1: File sizes and their compression values

Model Format	Size	Zipped Size	Compression Ratio
	KB	KB	
Native ArchCAD	425	309	27,29 %
IFC 2X3	6	3	50 %
Interpreted Java	11	4	63,63 %
Un-interpreted Java	9	3	66,66 %
IFC XML 2X3	24	5	79,16 %

It is clear that the native ArchiCAD file in this case is 70 times greater than the IFC file. This might be attributed to the reason that the file contains a lot of overhead information that is related to ArchiCAD software and not the Building Information Model. We will notice in the next text cases that the size of the native format gets smaller than the IFC STEP format as the underlying models grow bigger.

It can also be noticed that the IFCxml format of the model is 4 times bigger than the IFC STEP file with the highest compression ratio (79%), while the Interpreted Java file is 1.8 times bigger and the Un-interpreted Java file is 1.5 times bigger.

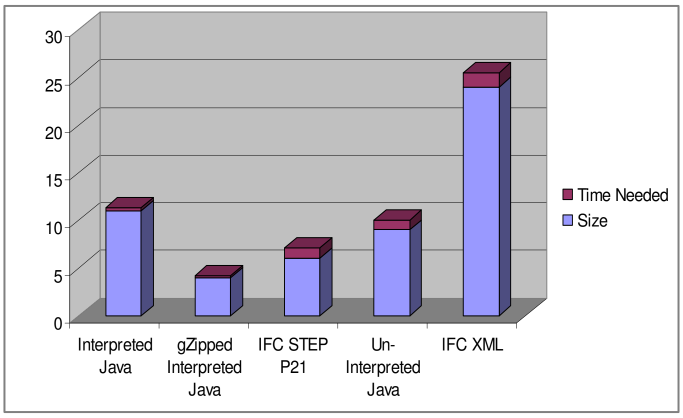


FIG. 8: Test Case1: De-Serialition speeds in relation to file sizes

TABLE 3: Test case 1: Files Sizes and related De-serialization speeds (FIG. 8)

Format	Size	Time Needed
	KB	Seconds
Interpreted Java	11	0.3
gZipped Interpreted Java	4	0.23
IFC STEP P21	6	1.1
Un-Interpreted Java	9	1.0
IFC XML	24	1.5

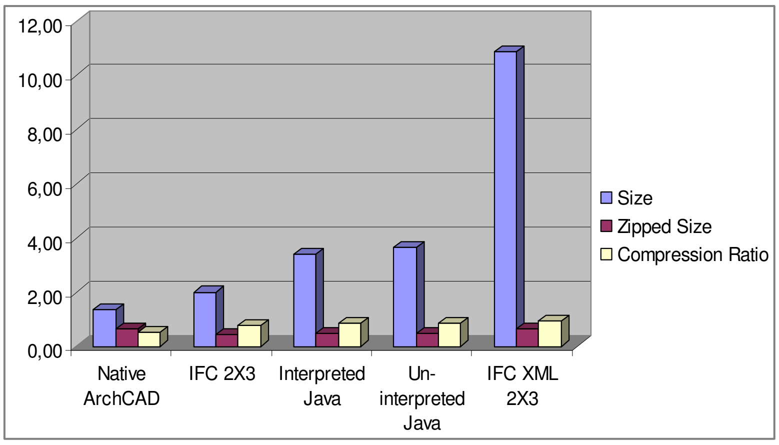


FIG. 9: Test Case 2: Size comparison of different exchange formats

By examining the speed of de-serialization and interpretation of the STEP-IFC format, the Interpreted Java serialization, the Un-Interpreted Java Serialization and IFC XML, the results were found to be according to the *TABLE 3* and *FIG. 8*.

The main outstanding result is that the de-serialization of the gZipped Interpreted Java IFC model took the shortest time (0.23 seconds) to get the IFC model in the form of early binding Java classes. In the meantime, it has the smallest size (4 KBs). The savings in time against the IFC STEP-P21 model parsing and interpretation time is nearly 80%, where as the time saving in comparison with the IFCxml version of the model is nearly 85%.

TABLE 4: Test Case 2: File sizes in addition to compression values and ratios (FIG. 9)

Model Format	Size	Zipped Size	Compression Ratio
	MB	MB	
Native ArchCAD	1.34	0.636	52 %
IFC 2X3	2.00	0.434	78 %
Interpreted Java	3.38	0.489	85 %
Un-interpreted Java	3.659	0.491	86 %
IFC XML 2X3	10.87	0.649	94 %

4.4 Test Case 2: An Office Building

This test case is a rather complex model of an office building. The model is downloaded from the internet (FZK, 2008). From *TABLE 4* and *FIG. 9*, it can be concluded that: Although the highest compression ration could be achieved on the IFC XML model (94 %), it still remains to be the largest in compressed model sizes (0.649 MB).

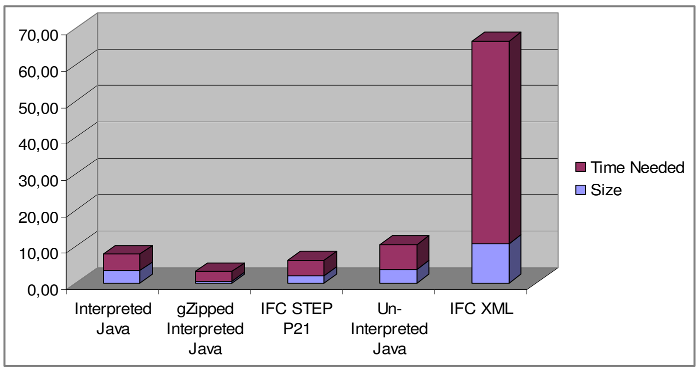


FIG. 10: Test Case 2: De-serialization speeds in relation to file sizes

TABLE 5: Test Case 2: File Sizes and related De-serialisation speeds (FIG. 10)

Format	Size	Time Needed
	MB	Seconds
Interpreted Java	3.38	4.5
gZipped Interpreted Java	0.489	2.6
IFC STEP P21	2.00	4.2
Un-Interpreted Java	3.659	6.8
IFC XML	10.87	55.6

From FIG. 10 and TABLE 5 it could be seen that the gZipped early binding IFC Java serialization has achieved both the smallest model size (after compression) and the least time needed for serialization. Moreover, it is faster than reading from an IFC STEP ISO 10303-P21 file by 62%.

4.5 Test Case 3: NHS Office

The test case is also a complex model of an NHS Building, *FIG. 12*. The model can be downloaded from (Graphisoft, 2008).

FIG. 12: A 3D perspective view of the NHS project

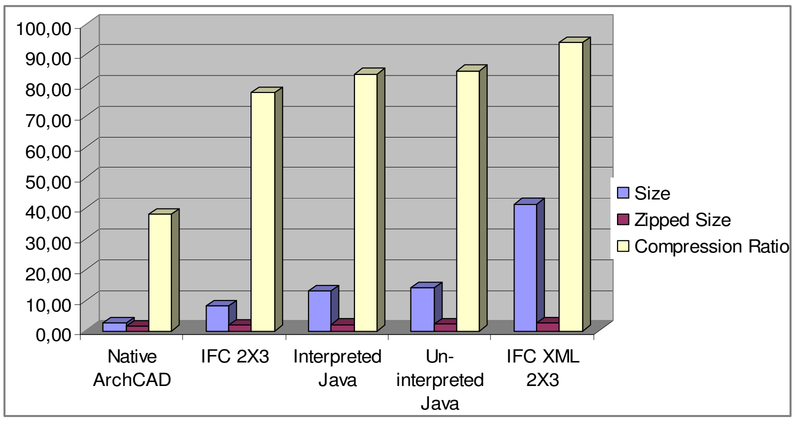


FIG. 11: Test Case 3: Size comparison of different exchange formats

TABLE 6: Test Case 3: File sizes in addition to compression values and ratios (FIG. 11)

Model Format	Size	Zipped Size	Compression Ratio
	MB	MB	
Native ArchCAD	26.3	23.7	9.8%
IFC 2X3	17.9	3.5	80.4 %
Interpreted Java	27.6	4.0	85.5 %
Un-interpreted Java	30.5	4.0	86.9 %
IFC XML 2X3	86.8	4.7	94.6 %

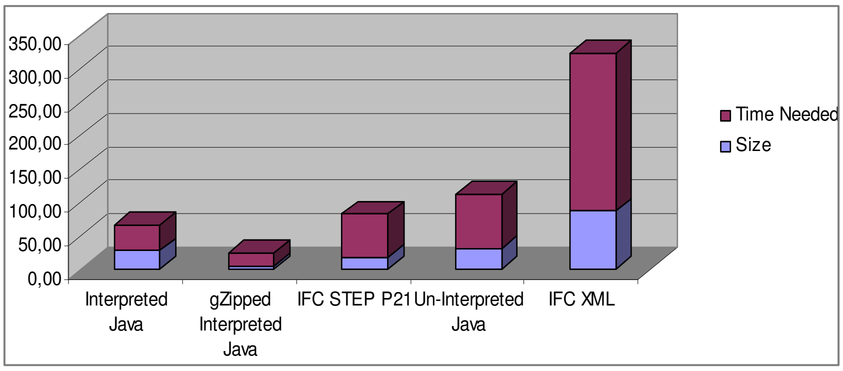


FIG. 13: Test Case 3: De-serialization speeds in relation to file sizes

TABLE 7: Test Case 3: File Sizes and related De-serialisation speeds (FIG. 13)

Format	Size	Time Needed
	MB	Seconds
Interpreted Java	27.6	38
gZipped Interpreted Java	4.0	20
IFC STEP P21	17.9	65
Un-Interpreted Java	30.5	81.5
IFC XML	86.8	237

As it can be seen from *TABLE 6* and *FIG. 11*, the zipping of the IFC model at this size starts to achieve substantial reduction on the model size in all cases except for the ArchiCAD native format.

4.6 Test Case 4: Ettenheim City

This test case is shown through *FIG. 14*, FIG. 15 and *FIG. 16* as well as TABLE 8 and TABLE 10. It is taken as an example for the relation between IFC and GIS (Geographical Information System). It contains 195 buildings, 1489 walls, 380 windows and 329 roof slabs (IfcSlab Roof). It can be downloaded from (FZK, 2008).

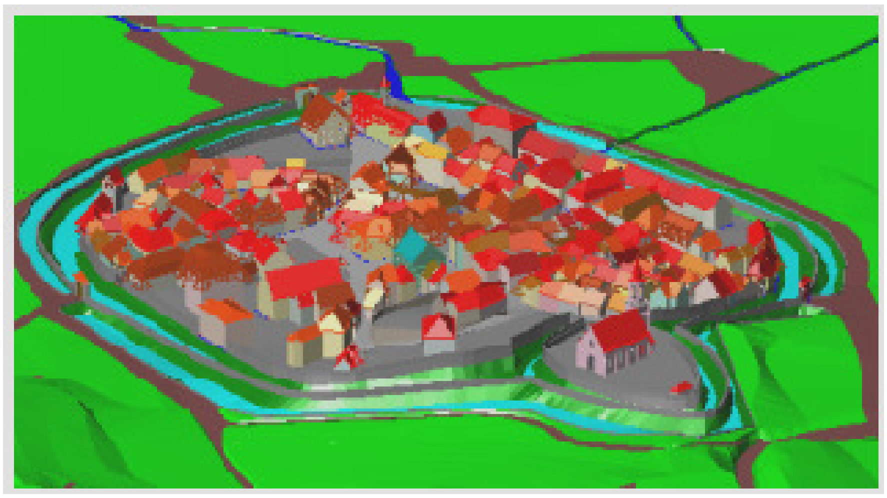


FIG. 14: A perspective view of Ettenheim city

TABLE 8: Test Case 4: File sizes in addition to compression values and ratios (FIG. 15)

Model Format	Size	Zipped Size	Compression Ratio
	MB	MB	
Native ArchCAD	2.55	1.6	38%
IFC 2X3	8.04	1.8	77.6 %
Interpreted Java	12.9	2.1	83.7 %
Un-interpreted Java	14.1	2.1	84.7 %
IFC XML 2X3	41.3	2.5	94.0 %

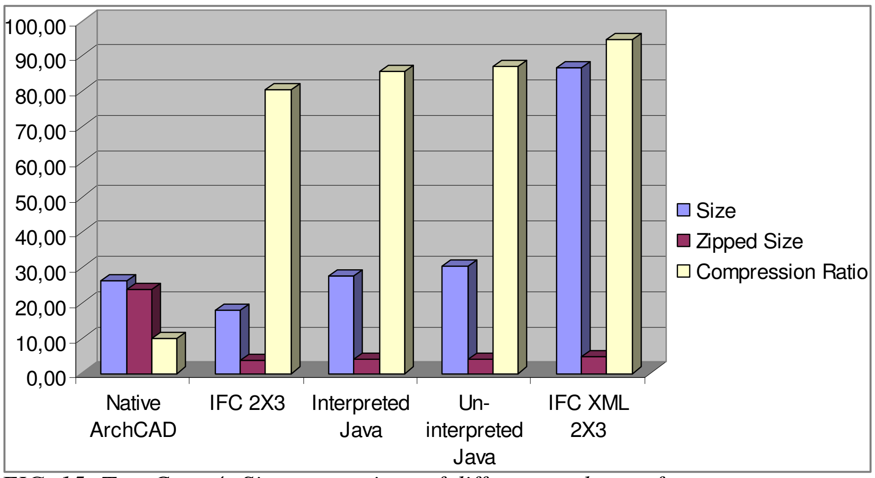


FIG. 15: Test Case 4: Size comparison of different exchange formats

TABLE 9: Test Case 4: File Sizes and related De-serialization speeds (FIG. 16)

Format	Size	Time Needed
	MB	Seconds
Interpreted Java	12.9	18
gZipped Interpreted Java	2.1	9
IFC STEP P21	8.04	54
Un-Interpreted Java	14.1	67
IFC XML	41.3	249

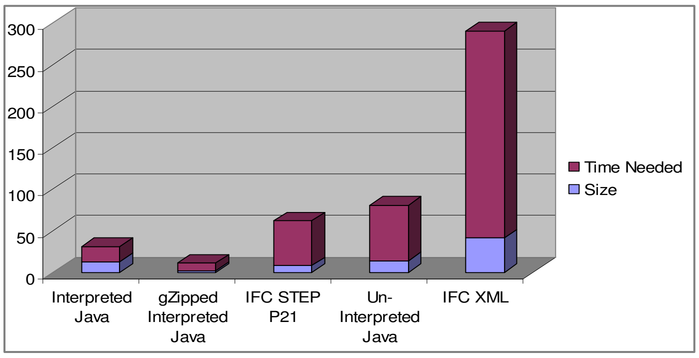


FIG. 16: Test Case 4: File sizes and related De-serialization speeds.

It is quite clear that the native CAD format in this case is much smaller than the IFC/STEP format and its interpretation. Again the gZipped interpreted Java achieves a better de-serialization speed and a smaller size over the IFC/STEP-21 format.

4.7 Test Cases Analysis

FIG. 17 shows a relative comparison for obtaining Java object at runtime according to the four test cases. It is quite clear that until the size of 2MB, there is no clear difference in performance. As soon as the sizes of the models grow more than 2MB, then the differences in performance tend to be much clearer. Although the IFC XML file has the highest compression ratio (about 95%), it proves to be inconvenient for the purpose of this research work. Further more, it is usually between 4-7 times bigger than the corresponding IFC STEP-21 file. To improve the readability of *FIG. 17*, *FIG. 18* is added, where the XML comparison is removed.

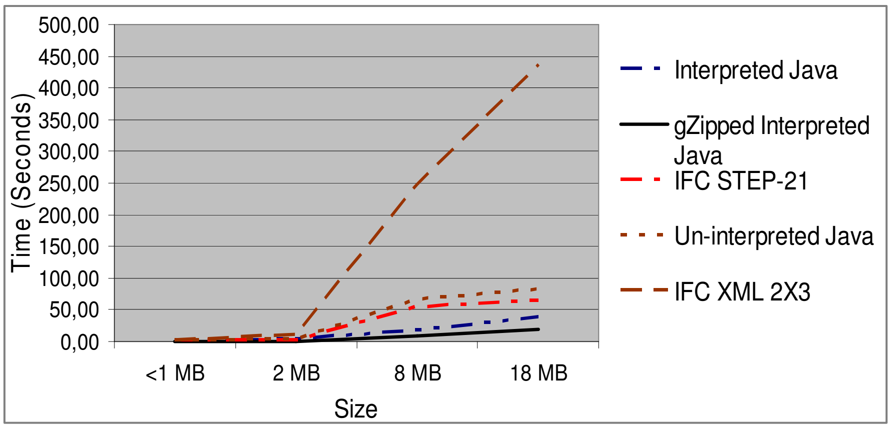


FIG. 17: Comparative analysis for the test cases

FIG. 18 also shows that slopes of the curves adapt a different pattern after the size of 2MB. The best performance is achieved by the gZipped Interpreted Java object de-serialization. It showed that it can save up to nearly 83% of the time required to get hold of the IFC early binding objects at runtime in comparison with the parsing and interpretation of the IFC STEP-21 files.

The gZipped Interpreted Java, in addition to the performance superiority has another advantage over IFC STEP-21 format. It requires in average 77% less persistent memory space. However, it could be argued that disk space is becoming no longer a problem.

The interpreted Java can save time over the un-interpreted variant because it saves the interpretation and binding phase of the IFC model at runtime.

The ArchiCAD native format was used only as an indication of the size of the model. It does not serve the purposes of this research work, as it is not possible to obtain a binding through a programming language using the native CAD format. Thus, the native format was not included in any performance measurement test. The test cases have shown that it is difficult to speculate whether the native CAD format is bigger in size than the IFC

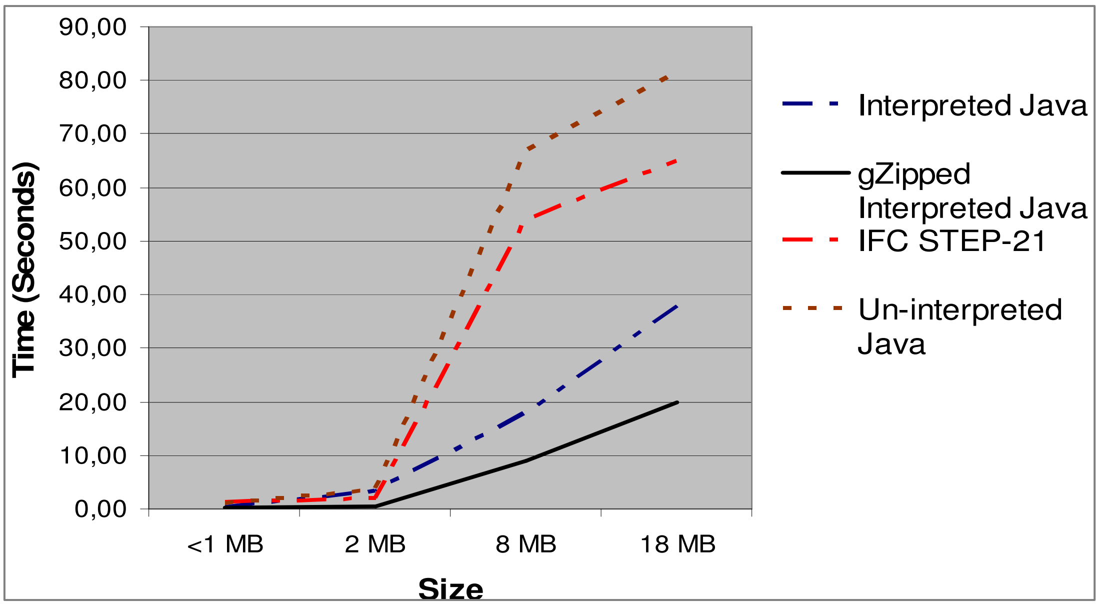


FIG. 18: Relative performance for obtaining runtime objects excluding the XML format

STEP-21 format or not. For IFC models smaller than 2 MB, the STEP-21 file would mostly be smaller than the native CAD format. However, as the model grows more than 2MB, the probability that the IFC model exceeds the size of the native format grows. There are also cases like test case 3, where the IFC STEP-21 is smaller. Many factors influence the size of the IFC STEP model in comparison to the native CAD format. Among these factors are: 1) The way geometry is described (e.g. using swept solids or Brep). 2) How repetition of same elements with different locations is handled. 3) Amount of semantic information (non geometry) in the model and so forth.

When writing an IFC XML parser, it was discovered that there is a very high potential for optimization of the STEP-21 files. There are often unnecessary sequences of referencing between relative locations of objects that could be optimized.

5. CONCLUSIONS AND RECOMMENDATION FOR FURTHER RESEARCH

The ability of the central database or model server that is responsible for the management of different versions of the BIM (Building Information Model) to perform queries, create partial models, integrate partial models (merging and updating) and shift between different versions of the model and its objects efficiently is considered to be an essential success factor.

In this connection, the idea of mapping the entire BIM / IFC model to a relational database has proved to impact the database’s performance, especially with data models greater than 2MB. Hence, a need was established to conduct a research work to overcome this performance problem.

The suggested solution approach is mainly built on capturing and filtering IFC/BIM domain data that is necessary for the conduction of queries and creation of needed business objects. It is envisaged that any query would result in a set of GUIDs (Global Unique Identifiers) that refer to objects within files (Model Versions) that are referenced by the database.

Various types of possible serialization and de-serialization of the IFC model were examined in terms of: 1) The file sizes, 2) The speed of serialization and de-serialization of the IFC model.

In order to be able to perform this analysis, STEP-21, IfcXml parsers and interpreters were developed by the author together with the needed tools for mapping IFC (ISO/PAS 16739:2005) objects to the underlying relational database.

Four IFC test cases with different degrees of complexities were used to compare the performance of obtaining runtime object models using interpreted and un-interpreted early bindings of Java objects in addition to IFC XML 2X3 and the STEP-21 formats.

The gZipping effect was tested on the serialized interpreted IFC Java early binding objects and has proved superiority in both performance and persistent memory needs.

Although the work in this paper has succeeded in achieving better performance values, there is still a strong need to be able to handle partial IFC models that are manageable within reasonable size units.

The introduced solution achieves relative improvement, but does not overcome the problems and deficiencies in performance associated with large size models that can reach gigabytes.

The areas for further research are very wide. Among these areas are: 1) The interpretation of the results obtained from queries to valid partial IFC models that can be communicated with other partners, compared and integrated with other models. 2) There is a need to establish data exchange protocols between the local workbench at the client side and the central model server bearing in mind the needs of object versioning.

ACKNOWLEDGEMENTS

The result presented in this paper is part of InPro (http://www.inpro-project.eu) - an integrated project co-funded by the European Commission within the Sixth Framework Programme (2002-2006).

6. REFERENCES

EDM Model Server (2009). JOTNE EPM Technology, Norway. Available at: http://www.epmtech.jotne.com/

Eurostep Share a Space (2009). IFC PLCS model server, Sweden. Available at: http://www.eurostep.com/

FZK Forschungszentrum Karlsruhe (2008). Institute für Angewandte Informatik. Available at http://iai-typo3.iai.fzk.de/www-extern/index.php?id=799&L=0

Graphisoft (2008). Virtual Building Solutions: NHS office. Available at http://www.gshelp.de/EDU/Content/05-Anwender/05-04-Beispiele.html

IAI (2007). International Alliance for Interoperability, IFC/ifcXML Specifications. Available at: http://www.iai-international.org/Model/IFC(ifcXML)Specs.html

IBM (2008). Business Objects, Designing and developing business objects. Available at: *http://publib.boulder.ibm.com/infocenter/wbihelp/v6rxmx/index.jsp?topic=/com.ibm.wbia_developer.doc/doc/connector_dev_java/java19.htm*

IMS (2002). IFC Model Server Development Project, VTT, Finland. Available at: http://cic.vtt.fi/projects/ifcsvr

ISO/PAS 16739 (2005). Industry Foundation Classes, Release 2X, Platform specifications. Available at http://iso.nocrew.org

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

ISO 10303-21 STEP (1994). Industrial Automation Systems and Integration — Product Data Representation and Exchange — Part 21: Implementation Methods: Clear Text En-coding of the Exchange Structure, ISO 10303-21:1994 (E), ISO, Geneva, 1994.

JavaCC (2005). Java Compiler Compiler. The Java Parser Generator, The source for Java Technology Collaboration, Java .NET. Available at: https://javacc.dev.java.net

Mahmoud Q. H. (2002). Sun Developer Network (SDN), Compressing and Decompressing data using Java. February 2002. Available at: http://java.sun.com/developer/technicalArticles/Programming/compression/

McCluskey G. (1998). Sun Developer Network (SDN), Using Java Reflection, Article, January. Available at http://java.sun.com/developer/technicalArticles/ALT/Reflection/

Microsoft (2007). Office Online, Microsoft Office Access 2007 database. Available at: http://office.microsoft.com/de-ch/access/FX100487571031.aspx

Pazlar T. and Turk Z. (2006). Analysis of the geometric data exchange using the IFC, In Proceedings of the ECPPM conference on eWork and eBusiness in Architecture, Engineering and Construction (Martinez & Scherer (editors)), Tylor & Francis Group, London, ISBN 0-415-41622-1

Sun (2008). The Java Tutorials, Retrieving Values from Result Sets. Available at: http://java.sun.com/docs/books/tutorial/jdbc/basics/retrieving.html