Efficient binary serialization of IFC models using HDF5

Krijnen, T.F.; Beetz, J.

Published in:
ICCCBE2016: 16th International Conference on Computing in Civil and Building Engineering, Osaka, July 6-8, 2016

Document license:
Unspecified

Published: 01/01/2016

Document Version
Author's version before peer-review

Efficient Binary Serialization of IFC Models Using HDF5

Thomas Krijnen[a], Jakob Beetz[b]

[a] Technical University of Eindhoven (TU/e), Eindhoven, The Netherlands

Abstract:

The Industry Foundation Classes (IFC) are a common file-based open standard to describe Building Information Models. An IFC file can describe a building model to a level of detail suitable for production use and unite information pertaining to all stakeholders involved in a construction project. IFC files can possibly constitute up to gigabytes of data. Processing the full extent of this data can be time consuming. Considering the multi-disciplinary nature of our industry it may also be unnecessary for the use case at hand. Therefore, the retrieval of relevant subsets, whether spatially, based on discipline, or others, is necessary to effectively consume such datasets in downstream applications.

However, prevalent encoding forms of IFC models are text-based. And even though, in terms of file size, the most prevalent encoding, called IFC-SPF, can be rather efficient, by nature, it does not facilitate random access seeking in the file and no ordering is imposed to the definition of elements in the file. Therefore, at worst, the entire file needs to be traversed in order to find instances of interest. Furthermore, text-based data is slow to parse in comparison to its binary equivalent.

This paper introduces a binary serialization for IFC models as an alternative to prevalent text-based formats. It is based on an existing open standard called HDF5. An implementation for the translation of conventional IFC instance models into HDF5 is provided under and open source license. HDF5 is a binary and hierarchical data format. The hierarchical nature allows random access to specific instances. Other benefits include transparent compression and mechanisms for linking and mounting external files. The compressed HDF5 format yields a significant reduction of file sizes as compared to IFC-SPF models. In three use cases is assessed that extracting data from the model, can occur in near-constant time in relation to the size of the model, contrary to linear time using IFC-SPF models.

The translation into HDF5 files follows an existing ISO standardized mapping from EXPRESS instance models, the parent standard of IFC. The self-documenting nature of HDF5 enables incorporating additional attributes that are not part of the schema. In order to improve visualisation one can cache calculated information such as triangulated geometry for complex CSG geometries that are computationally complex to compute. In addition, incorporating inverse attribute values as part of the instantiation allows to further optimize the generation of subgraphs.

Keywords: BIM, IFC, EXPRESS, HDF5, binary storage, file size, performance

1. INTRODUCTION

The Industry Foundation Classes (IFC) are aimed at to providing a comprehensive set of modeling constructs by which all disciplines involved in the entire life cycle of a construction project can express and store relevant information. Due to this cross-domain nature, a variety of information is populated and IFC instance files can be of considerable magnitude, possibly up to gigabytes of data. The retrieval of subsets, based on the spatial location of elements in a building or a particular discipline, is necessary to effectively consume such datasets in downstream applications (Weise et al., 2003). In the past, a number of query languages have been developed to formalize systems by which information can be extracted from IFC files. This can be done on the level of attributes (Mazairac & Beetz, 2013) or topological relationships (Daum & Borrmann, 2014). For the curation of metadata pertaining to IFC models and the buildings they describe, automated systems are developed that extract this information (Beetz et al., 2013). These use cases have in common that they are aimed at extracting a subset of the model and present this to the user.

However, to date, the most common serialization format of IFC instance models is the ISO 10303 part 21 (Step Physical File Format, SPF, p21). It has been designed to maximize interoperability, not storage and retrieval efficiency. The use cases of efficiently computing subgraphs or extracting relevant information sketched above are not feasible as SPF instance models grow in size. The primary reason for this is the lack of structuring in the way entity instances are laid out in an SPF serialization. It represents one continuous stream of instances, without the possibility to arbitrarily seek random locations in the file prior to parsing it completely. Furthermore, parsing inefficiency is increased by the plain text representation of floating point numerals, which have to be

parsed and translated into their machine equivalent. In SPF, storage space is not efficiently used due to the full entity names being used redundantly for every instantiation of data. At the same time, extensions to the IFC schema, such as IfcBridge, IfcAlignment or point cloud data structures extensions are being proposed (Krijnen et al., 2015) and integrations with heterogeneous forms of data (Liu & Akinci, 2009) are being developed that further increase the requirements for storage and retrieval efficiency.

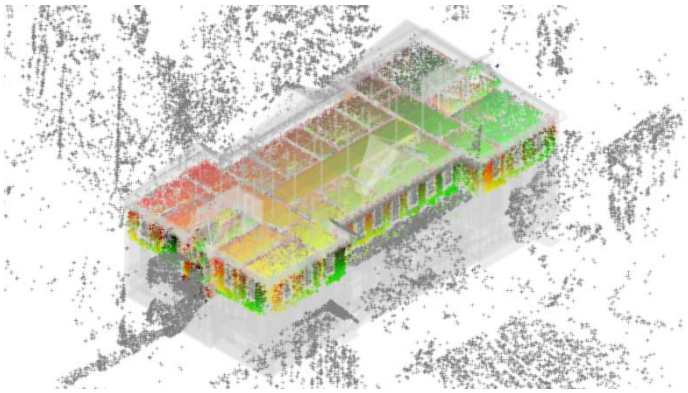

Figure 1. Example of an IFC model with point clouds serialized as IFC-HDF. Source: Krijnen et al. (2015)

On the other hand, the EXPRESS modelling language, in which the IFC schema is specified is versatile and flexible, featuring advanced concepts such a Turing-complete formulation of schema level constraints, multiple inheritance, combinations of several subtypes and algebraic data types, referred to as a SELECT type. These concepts can be succinctly expressed in a text-based p21 file, as it poses no limitations on the length of records or valuation of attributes. Consequently it poses the need for external semantic validation tools for seemingly trivial validation tasks, such as asserting that instance attributes are of the correct type (Lee et al., 2015).

Listing 1. Definition of an entity type in EXPRESS showing a built-in constraint check for the correct use of data types

```
ENTITY IfcCartesianPoint
 SUBTYPE OF (IfcPoint);
    Coordinates : LIST [1:3] OF IfcLengthMeasure;
 DERIVE
    Dim : IfcDimensionCount := HIINDEX(Coordinates);
 WHERE
    CP2Dor3D : HIINDEX(Coordinates) >= 2;
END_ENTITY;
```

2. HDF5

The Hierarchical Data Format (HDF) (Folk et al., 2011) is an open standard for efficiently handling large datasets. HDF is used for more than 20 years in different engineering and scientific communities to cope with large amounts of data including e.g. physics, astronomy and medical datasets. It is an open standard with mature portable implementations in many languages that allows efficient I/O. It provides flexible mechanisms for data modeling in a transparent, self-documenting way that makes it an accepted candidate for long-term preservation (Bertin-Mahieux et al., 2011; Gray et al., 2005) and interoperability (Choi & Folk, 2007a; Dougherty et al., 2009) use cases. With the ISO 10303 part 26, a standard exists that specifies mappings of STEP schemas from EXPRESS into HDF 5 as well as the serialization of instance model populations.

Apart from the storage space advantages stemming from a binary storage format, HDF also adds a number of additional advantages over part 21 SPF workflows. Firstly, the underlying data schema and meta-data is transported along with the instance files. This guarantees a correct valuation of attributes in terms of their datatype. It also allows for storing additional attributes that are not part of the schema specification. For example pre-computed geometrical triangulations for efficient visualization or an instantiation of inverse attributes for efficient graph traversal. Most notably, the hierarchical nature of HDF and its indexing mechanisms, allow fast random access and slices through large datasets and facilitate lazy loading approaches.

Technically an HDF5 file can be considered akin to how a file system can be organized. An HDF5 file spans a collection of objects, which are retrieved by means of their membership to groups. Group names can be chained to construct paths to access datasets, similarly to a file system. This constitutes the hierarchical nature of HDF5

and the performance gains introduced in Section 3.2 stem to a large extent from this organizing mechanism.

Datasets are types of objects in a HDF5 file and hold entries of numeric, textual or compound data fields. Datasets carry a reference to a datatype, which describes the content of the data, and to a dataspace, which describes the dimensions of the data. Filters can be applied to datasets that governs for example compression or the calculation of checksums. Processing these filters upon storage and retrieval is governed by the HDF5 library, hence datasets can be transparently compressed and decompressed.

HDF5 offers a variety of native datatypes in a familiar idiom, such as integers and floating point numerals. As opposed to the EXPRESS modeling language, these numeric datatypes have a predefined and fixed width and precision. Enumerations are offered to describe a value adhering to a set of predefined textual values, similarly to EXPRESS offers. Arrays of fixed size and variable length fields are used to express aggregates. Compounds span an ordered collection of named members, which are themselves fully typed attributes. HDF5 does not have a native mechanism for optional fields as the retrieval efficiency stems from the fact that all fields have a predefined and fixed length. This fixed length requirement also has implications for variable length aggregates, such as lists, as they are stored in a separate data structure, referred to as a heap, with 32 bytes auxiliary space per variable length attribute (HDF5 group, 2003).

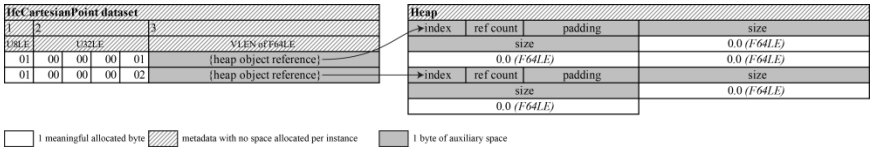

Figure 2. Instances with variable length aggregate attributes as stored in an HDF5 file

The compound datatype field is what constitutes the self-documenting nature of HDF5. Contrary to SPF-serialized models, the attribute names are contained in the file and are therefore easily accessible when viewing HDF5 data with general purpose tools. The self-documenting attribute names do not incur a storage space penalty since they are only stored once per datatype.

2.1 EXPRESS HDF5 mapping

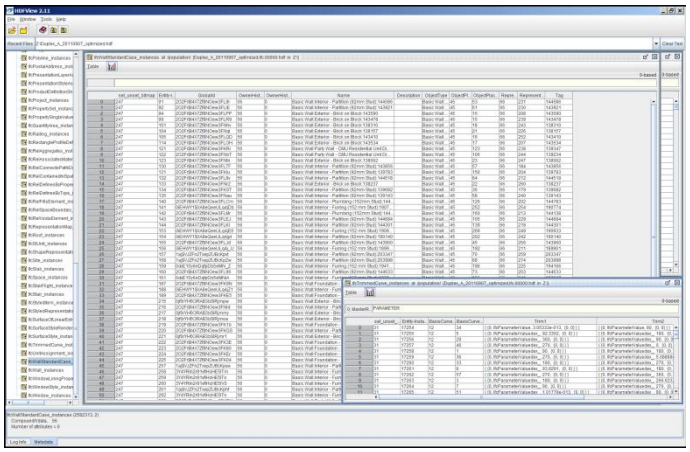

Figure 3. An IFC-HDF model opened in the freely available generic "HDFView" application

A standardized mapping, ISO 10303-26 exists to create HDF5 files to describe EXPRESS populations. Compound datatypes form the cardinal building block of this mapping as they are used to represent entity types. Also instance references and SELECT types are defined as HDF5 compounds. An instance reference consists of a pair of unsigned integers, referring to a dataset and the row in the dataset that describes the instance. The respective EXPRESS schema is captured in the instantiation as part of a group with committed datatypes. Model populations are also encapsulated in a group. Hence, an HDF5-serialized model can contain several instance models as well as several schemas. The complete mapping can be found in the ISO 10303-26 standard. An example of a textual description of an IFC datatype in HDF5 is provided in Listing 2.

Listing 2. HDF5 Datatype as reported by the "HDF5 dump" utility

```
DATATYPE "IfcAnnotationFillArea" H5T_COMPOUND {
```

```
   H5T_STD_I16LE "set_unset_bitmap";
   H5T_STD_I32LE "Entity-Instance-Identifier";
   H5T_COMPOUND {
      H5T_STD_I16LE "_HDF5_dataset_index_";
      H5T_STD_I32LE "_HDF5_instance_index_";
   } "OuterBoundary";
   H5T_VLEN { H5T_COMPOUND {
      H5T_STD_I16LE "_HDF5_dataset_index_";
      H5T_STD_I32LE "_HDF5_instance_index_";
   }} "InnerBoundaries";
}
```

2.2 HDF5 Serialization Options

As discussed in section 2, the nature of HDF5 allows for efficient retrieval as individual entity instances can be located in constant time because of their fixed width in the data structure. In contrast, instances in IFC-SPF models begin at arbitrary locations in the file without any imposed ordering or structure. At the same time, the necessity for fixed length fields comes with a storage space penalty in the heap object structure and references to them for variable length attributes and in the allocation of space for all possible select instantiations. This is illustrated in Figure 4, which segregates the relative file size contribution of instances per entity type for an individual model. Using a logarithmic scale, the bars represent the difference in file size in uncompressed IFC-HDF as compared to the amount these instances account for in the original IFC-SPF model. This chart is obtained from only a single model, but does point at the generic notion that variable length attributes, which include 'Name' and 'Description' strings of all rooted entities (those sub-classed from 'IfcRoot'), and large select structures do incur some storage overhead. Note however, that this is far from prohibitive, because this overhead for select structures is effectively eliminated by the combination of the transparently applied filters. Secondly, the storage overhead does not incur a measurable penalty in accessing the data, see Section 3.2.

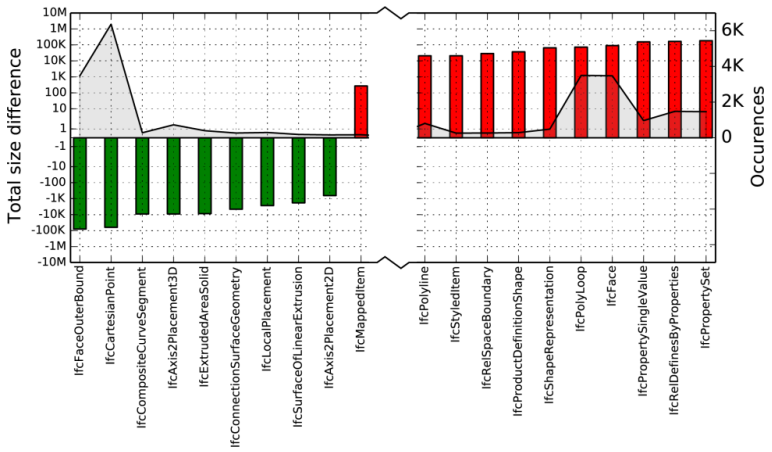

Figure 4. Relative contribution to file size differences segregated per entity type

As can also be seen from Figure 4, Cartesian point instances constitute a large part of the model. Without losing semantic meaning an amendment can be made to the mapping for variable length attributes of a known upper bound. An array instantiation can be chosen instead with out-of-bounds elements set to a placeholder value, for example not-a-number (NaN). The gains of this are threefold: firstly, less auxiliary bytes are populated as the heap object structure and reference thereto no longer needs to be populated. Secondly, (compression) filters do not operate on the heap, hence, this proposal allows point datasets to also benefit from transparent compression. Lastly, since the heap object is located in another segment of the HDF5 another I/O operation is eliminated by consolidating the data in the dataset itself. Applicable entities where similar rules can be applied are limited to 12 entity attributes in the IFC2x3 schema, including prevalent IfcCartesianPoint.Coordinates and IfcDirection.DirectionRatios. A graphical depiction following this enhanced mapping scheme is given in Figure 5. As opposed to the reference to a variable length object on the heap, the entire aggregate is now consolidated into the dataset, with potentially one obsolete value set to NaN. This strategy is assessed as well in Folk & Choi (2004) in which the suitability of HDF5 is assessed for geospatial features of varying complexity.

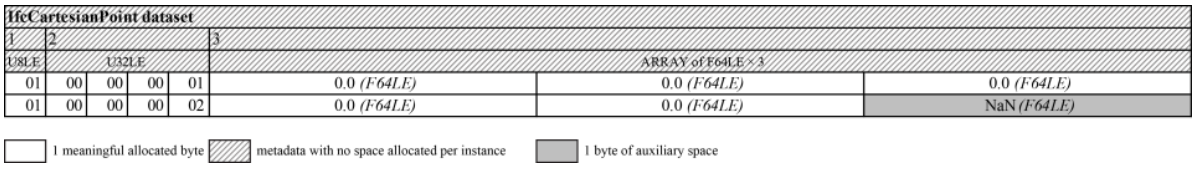

Figure 5. Modified structure for entities with aggregate attributes with upper bound

Another serialization option, which is suggested but only scarcely documented in the standard, is the option to populate inverse attributes. Inverse attributes are relations in the opposite direction between entities that refer to each other by means of an explicit attribute. In the case of the IfcObjectPlacement listed in Listing 3 an abstract entity is introduced that governs the placement of an object in a model by assigning it to the ObjectPlacement attribute of an IfcProduct. Not only does the inverse relationship PlacesObject allow to trace back to the product that uses the placement, it also restricts the cardinality of this relationship in the sense that an IfcObjectPlacement has to be referenced by one and only one IfcProduct instance.

Inverse attributes are not serialized as part of any of the current prevalent IFC serialization formats. It would not be of much use anyway, as typically the entire graph has to be parsed and the implementation can create inverse relationship instances along the way. The HDF5 serialization is radically different in the sense that it is designed not to demand the entire model to be parsed, but rather to allow efficient computation of subgraphs and easily extracting relevant content. Hence, incorporating inverse attribute values as part of the serialization is useful. The results show that doing so incurs only a negligible increase in file size, see Section 3.1.

Listing 3. An EXPRESS entity with inverse attributes

```
ENTITY IfcObjectPlacement
 ABSTRACT SUPERTYPE OF (ONEOF
    (IfcGridPlacement
    ,IfcLocalPlacement));
 INVERSE
    PlacesObject : SET [1:1] OF IfcProduct FOR ObjectPlacement;
    ReferencedByPlacements : SET [0:?] OF IfcLocalPlacement FOR
       ↪ PlacementRelTo;
END_ENTITY;
```

Lastly, several serialization options are related to the binary nature of HDF5. They are crucial aspects and future research should formulate guidelines on these aspects in order to guarantee interoperable results. The most pertinent being the alignment of datatypes within a compound structure. Hardware platforms and compiler software impose hard restrictions on the alignment to memory boundaries of objects. This alignment requirement varies based on the size of the object and is a non-negative integral power of two. Attempting to read or write to a memory address that is not properly aligned results in undefined behaviour and yields incorrect results. If datatypes within a HDF5 compound do not align to the restrictions imposed by the platform the object needs to be copied first to a location in memory that does satisfy this requirement. This requirement is potentially unknown at the time of writing the HDF5 file for other exotic systems interpreting the file later on, but in reality reasonable assumptions can be made that are appropriate for common architectures.

3. RESULTS AND VALIDATION

An implementation of the EXPRESS HDF5 mapping is created in a C++ prototype, built on top of IfcOpenShell, an open source toolkit for working with IFC files. Results are presented for the file sizes of HDF5 serialized models and access and processing times in three scenarios. The models are taken from an on-line repository of building models (Tamke et al. 2015) combined with some industry models of several hundreds of megabytes in size as IFC-SPF.

3.1 File Size

File sizes for various serialization options are given in Figure 6, along with a trend line which is fit through the data. As the length of entity instance identifiers increases with the magnitude of the file[1] the trend line for IFC-SPF follows a polynomial line. For IFC-HDF the population follows a straight line, but with a relatively large constant factor, which reflects the overhead in allocation of metadata and alignment. Compressed HDF5 tends to get more efficient with an increasing size as the compression algorithm can exploit the similarities in

[1] Compare for example "#1=IFCFACE((#2,#3));" and "#10000001=IFCFACE((#10000002,#10000003));"

between the instances. Adding inverse attribute values to the compressed HDF5 instantiation yields a negligible increase in file size. Compressing IFC-SPF data is not considered as it does not offer means for partial retrieval.

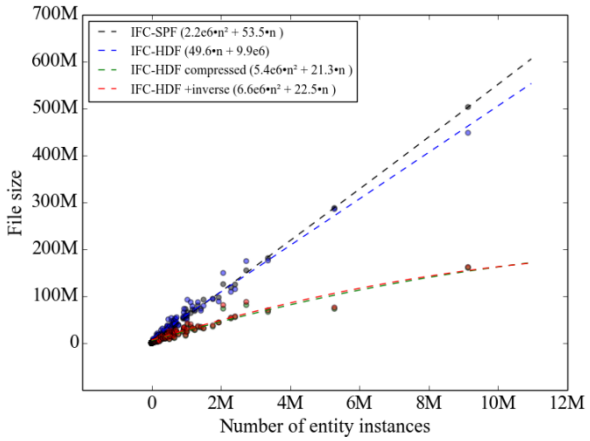

Figure 6. File sizes of SPF- and HDF-serialized models

3.2 Access and processing times:

In order to assess processing times in various use cases, three programs are presented with increasing levels of complexity. For parsing IFC-SPF files, IfcOpenShell is used. For parsing HDF5 only the HDF5 C++ library itself is used. The self-documenting compound member names are sufficient to interpret the data and yield readable source code. One can argue that this results in an unfair comparison as the overhead of the generic IFC-SPF library is not reflected in the parsing logic of the HDF files for these specific purposes. Partly this is true, however, with HDF5 being a containerized and hierarchically ordered file format it is fundamentally different from SPF and there is no way to tailor the IFC-SPF parsing to the three use cases as it is possible in IFC-HDF. Furthermore, the HDF5 library poses an overhead as it schedules how chunks of datasets are read, allocated and cached in memory. These notions make a completely unbiased comparison somewhat of an intractable problem. The comparison in this section therefore does not so much act as a means to formulate exact reproducible performance gains when a switch to HDF5 serialized instance files would be made. The comparison merely reflects on the inherent performance characteristics of these files. The three use cases are compiled into an optimized 64bit windows executable using Microsoft Visual Studio 2010.

(1) Metadata extraction: Find the total number of entity instances and the number of distinct entity types

For descriptive purposes it can be desirable to extract metadata that pertains to technical details of the IFC file. The number of distinct entity types instantiated in a file reflects the diversity of information collected in the model. Comparing results between operating on SPF data and HDF data hints at the fundamental differences in how the models are stored. SPF is a single continuous stream of entity instances without support for random access reading. On the other hand, HDF is a hierarchically decomposed storage model, with instances organized based on their entity type and with metadata describing the amount of instances and their layout within the file. A depiction of the results is omitted for brevity as the results were consistent for all files tested: extracting the data from an HDF5 fell below the resolution of the used timer, as only metadata in the file headers were accessed.

(2) Access explicit instance attributes: Find the largest window in a model

This example reflects on the use case of a simple query for IFC instance models. In this particular case, a single IfcWindow instance is requested for which no other instance in the model yields a higher value for the product OverallHeight × OverallHeight. When implemented for a SPF file, the entire file needs to be parsed as there is no other method that guarantees visiting all IfcWindow instances. In HDF5 this is fundamentally different as instances are grouped by their type in a known location in the file. Furthermore, this use case highlights the other important characteristic of binary storage, which is the fact that floating point numerals can be retrieved faster than from a decimal text representation in SPF files.

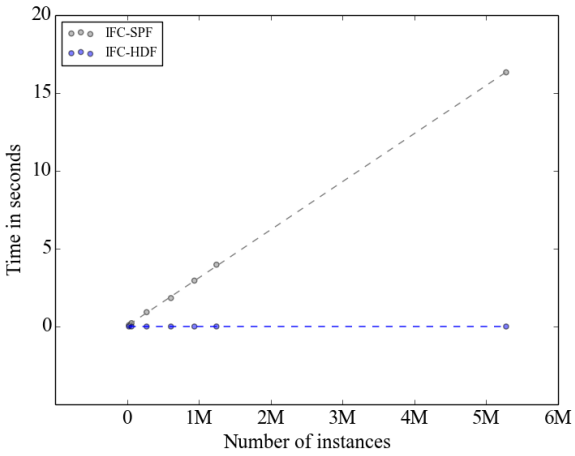

Figure 7. Time required to find the largest window in SPF and HDF serialized models

(3) Graph traversal: Identify all exterior walls in a model

On top of attributes defined as part of the entity definition in the schema, which were queried for the use case above, the IFC schema allows for extensible product-level metadata storage based on key-value pairs. These IfcPropertySets are related to building elements by means of an objectified relationship. Therefore, the traversal from building element to property includes several indirections, namely: {IfcWall, IfcWallStandardCase} ↷ {IfcRelDefinesByProperties} ↷ {IfcPropertySet} ↷ {IfcPropertySingleValue}. For the latter, when the Name is set to 'IsExternal' and the NominalValue attribute set to the boolean instance value 'true', the wall is identified as an exterior wall. As this example features the most complex traversal it is also most susceptible to subtleties in implementation and instantiation and with the most trade-offs. Implementations have to decide whether to read an entire dataset at once, or reading slices that contain instances to visit, balancing memory usage and I/O performance. Both also affected by dataset-level options such as compression and chunk size. In addition, the original hypothesis was that the performance for this use case scenario could be further improved by the option to instantiate inverse attributes. however, no performance gain was observed, likely due to the fact that datasets are reasonable in size and fit entirely into memory, as instances are not contiguously allocated the strategy of following instantiated inverse attributes resulted in many fragmented read operations.

Table 1. Results obtained

	IFC-SPF		IFC-HDF	
instances	time (s)	read (kb)	time (s)	read (kb)
27529	0.09360	1617	0.012480	820
62930	0.35568	4003	0.034320	2545
273138	1.60369	17716	0.112321	14445
611535	1.86265	42728	0.015600	1603
938124	3.01706	51618	0.124801	27408
1244987	3.99363	66950	0.015600	2275
5276285	16.39882	281376	0.028080	2736

4. CONCLUSION

With the serialization format introduced in this paper, profound reductions of access and processing times have been obtained, leading to near constant time retrieval in all tested use cases. An overall reduction of file sizes, regardless of serialization options, was obtained only for larger files within the dataset. However, with the application of transparent compression algorithms, natively part of HDF5, file size reductions are also obtained for smaller files. Comparing the results with an earlier report by Choi & Folk (2007b) on isolated STEP schema entities, it shows how freeform constructs like the ones employed in the IFC schema are sometimes more efficiently and succinctly expressed in IFC-SPF. Especially for smaller file sizes, storage overhead induced by HDF5 can be significant.

Not all practical aspects have been conclusively decided to propose a standardized mapping that guarantees

interoperable results. Many of these can be decided upon on the level of individual files. Considerations such as compression level, chunk size and numeric precision heavily impact performance characteristics of file size and access times, but consuming applications do not have to deal with these aspects manually as decompression, datatype transliteration and access and caching of chunks are handled by the HDF5 library transparently. Other aspects on the other hand such as alignment are architecture and compiler specific and are therefore harder to standardize.

Given the minimal increase in file size, an instantiation of inverse attributes is recommended. The IFC schema is very heavy on inverse attributes due to relational semantics between products. However, quantifiable performance gains were not obtained when traversing graphs using the instantiated inverse attributes. This is likely a consequence of the nature of the queries, size of the datasets and attributes involved. For example, if inverse attributes are considered referring to relating entity types with several subtypes, such as IfcRelAssociates, with 6 subtypes in the IFC4 schema, more datasets will need to be traversed without instantiated inverse attributes.

The implementation of the IFC-SPF to HDF5 conversion is available on-line under an open source license[2].

REFERENCES

Beetz, J., Dietze, D., Edvardsen, D. F., Fetahu, B., Gadiraju, U., Krijnen, T. (2013) *D3.3: Semantic Digital Archive Prototype*. Retrieved from http://duraark.eu/wp-content/uploads/2014/08/DURAARK_D3_3_3.pdf on Dec 2015.

Bertin-Mahieux, T., Ellis, D. P. W., Whitman, B., & Lamere, P. (2011). *The Million Song Dataset*. ISMIR 2011: Proceedings of the 12th International Society for Music Information Retrieval Conference, October 24-28, 2011, Miami, Florida, 591–596.

Choi, V., & Folk, M. (2007a) *Investigations into using HDF5 as an alternative to STEP for finite element modeling data*. In Proceedings of the 9th NASA-ESA Workshop on Product Data Exchange, National Aeronautics and Space Administration, Santa Barbara, CA.

Choi, V., & Folk, M. (2007b) *Investigations into using HDF5 for product model data – B-Spline and Cartesian Point data in HDF5*. Retrieved from https://www.hdfgroup.org/projects/nara/tech_report_on_B-spline_HDF5_study.pdf on Dec 2015.

Daum, S., Borrmann, A. (2014) *Processing of Topological BIM Queries using Boundary Representation Based Methods*, Advanced Engineering Informatics, Volume 28, Issue 4, October 2014, Pages 272-286, ISSN 1474-0346.

Dougherty, M. T., Folk, M. J., Zadok, E., Bernstein, H. J., Bernstein, F. C., Eliceiri, K. W., … Best, C. (2009). *Unifying Biological Image Formats with HDF5*. Commun. ACM, 52(10), 42–47.

Folk, M., Choi, V. (2004) *Scientific formats for geospatial data preservation - A study of suitability and performance*. Retrieved from https://www.hdfgroup.org/projects/nara/Sci_fmts_and_geodata_HDF.pdf on Dec 2015.

Folk, M., Heber, G., Koziol, Q., Pourmal, E., & Robinson, D. (2011). *An overview of the HDF5 technology suite and its applications*. In Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases (pp. 36–47). New York, NY, USA: ACM. doi:10.1145/1966895.1966900

Gray, J., Liu, D. T., Nieto-Santisteban, M., Szalay, A., DeWitt, D. J., & Heber, G. (2005). *Scientific Data Management in the Coming Decade*. SIGMOD Rec., 34(4), 34–41. doi:10.1145/1107499.1107503

HDF5 Group (2003) *Variable-Length Datatypes in HDF5*. Retrieved from https://www.hdfgroup.org/HDF5/doc/TechNotes/VLTypes.html on Dec 2015

Krijnen, T., Beetz, J., Ochmann, S., Vock, R., Wessel, R. (2015). *Towards extending IFC with point cloud data*, In proceedings of 22nd EG-ICE International Workshop, Juli 2015.

Lee, Y.-C., Eastman, C. M., Lee, J.-K. (2015). *Validations for ensuring the interoperability of data exchange of a building information model*, Automation in Construction, Volume 58, October 2015, Pages 176-195.

Liu, X., & Akinci, B. (2009). *Requirements and evaluation of standards for integration of sensor data with building information models*. Computing in Civil Engineering, 10.

Mazairac, M., Beetz, J. (2013) *BIMQL – An open query language for building information models, Advanced Engineering Informatics*, Volume 27, Issue 4, October 2013, Pages 444-456, ISSN 1474-0346.

Tamke, M. et al. (2015) *DURAARK Datasets*. Retrieved from http://data.duraark.eu/ on Dec 2015

Weise, M., Katranuschkov, P., & Scherer, R. J. (2003). *Generalised model subset definition schema*. CIB REPORT, 284, 440.

[2] https://github.com/ISBE-TUe/IfcOpenShell-HDF5